NISTIR 4654

Development of an Optical Disk System for the Automated Retrieval of EASEAR Records

Natalie Willman

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Advanced Systems Division
Computer Systems Laboratory
Gaithersburg, MD 20899

Sponsored by:

Social Security Administration
Baltimore, Maryland

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

NIST

NISTIR
QC100
.U56
4654
1991

NISTIR 4654

Development of an Optical Disk System for the Automated Retrieval of EASEAR Records

Natalie Willman

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Advanced Sjystems Division
Computer Systems Laboratory
Gaithersburg, MD 20899

Sponsored by:

Social Security Administration
Baltimore, Maryland

August 1991

Acknowledgments

Many people, in addition to the author of this report, have contributed to the program outlined in this publication, and the author is most appreciative of that assistance. Special thanks are due to the following individuals from NIST: Dana Grubb (Manager, Data Storage Group), Fernando Podio, Mark Williamson, Lloyd Gilmore and Diane Honeycutt.

The support and information provided by the project team at SSA are also appreciated. Special thanks are due to the following individuals from SSA: Malcolm Ewell, Ed Bulson, Marty Mahon, Joan Piasecki, Janice Whelchel, Ann Dempsey, Charlie Fuller, Perry Briscoe, Michael Macer and Michael Banks.

Table of Contents

Table of Figures

1. Introduction

The Social Security Administration (SSA) maintains records of the yearly wages earned by every person in the United States. Each year, approximately 2.5 gigabytes of data are collected on self-employed wage earners and stored in the EASEAR file. Another 47 gigabytes of data are collected on other wage earners and stored in the EAMATE file. The records are currently stored on over 110,000 rolls of microfilm. It takes over 400 scouts to retrieve information needed by the professional staff. Often, the needed roll of film is missing from the file due to being currently in use, misfiled, or misplaced. In addition, the information obtained from the microfilm is not always the most current, and decisions are sometimes based on obsolete data. Advances in peripheral mass storage technology during the 1980s (e.g. magneto-optic recording) now allow for alternative approaches to data storage and retrieval. The agency believes that an automated retrieval system would provide a more accurate, timely, and cost effective means of retrieving the information.

The focus of this document is the evolution and design of the EASEAR pilot application. The full pilot consists of three phases. Phase One of the project consists of the development of an automated retrieval pilot system for two years of the EASEAR records. In Phase Two of the pilot system, one year of the EAMATE records will be added to the pilot system, and in Phase three, a provision will be made for the connection of the pilot system to the SSA mainframe. The goals of the project are as follows: to demonstrate the feasibility of a full scale automated records retrieval system, to study the impact of an automated system on the users, and to gain insight into the factors which may impact the design of a full scale system.

This pilot system, both hardware and software, is intended solely for use in a pilot environment. The NIST developed software, while being tested to the best of our abilities, is not guaranteed to be error free. The pilot system design was intended to minimize cost while placing maximum emphasis on establishing the feasibility of an automated system and studying the factors that would impact the design of a full scale system. The pilot system design used does not imply any recommendations for a full scale system design. Any products named are for descriptive purposes and do not imply a recommendation for a final implementation.

2. Pilot system Components

The EASEAR pilot system is an integration of NIST developed software and commercially available software and hardware. In this way, the pilot system could be tailored to focus on the areas that were of interest to SSA and which could influence a later implementation of a full scale system. The following section discusses the hardware and software components of the pilot system.

2.1 Hardware Components

The EASEAR pilot system file server consists of a Compaq 386/20 with a 200 MB hard drive and 12 MB of RAM. The file server is daisy chained, via a Western Digital 7000 FASST-2 SCSI host bus adapter and external SCSI cables, to a Hewlett-Packard 88780B 12.7 mm (½ in) open reel tape drive and a Hewlett-Packard C1710A rack mountable optical disk library with two 130 mm (5¼ in) magneto-optic drives and 32 cartridge slots. The file server also contains an Arnet Smartport-8 multiport adapter board connected to four 19200 baud modems. The power to the file server and its peripherals is regulated by a Sola MCR series 2.0 kVA surge suppressor/voltage regulator.

The workstations in the pilot employ Intel 80286 or 80386 microprocessors in a 100% IBM compatible AT with color monitors and 101 keyboards. The workstations communicate with the file sever using 19200 baud modems (see Figure 1).

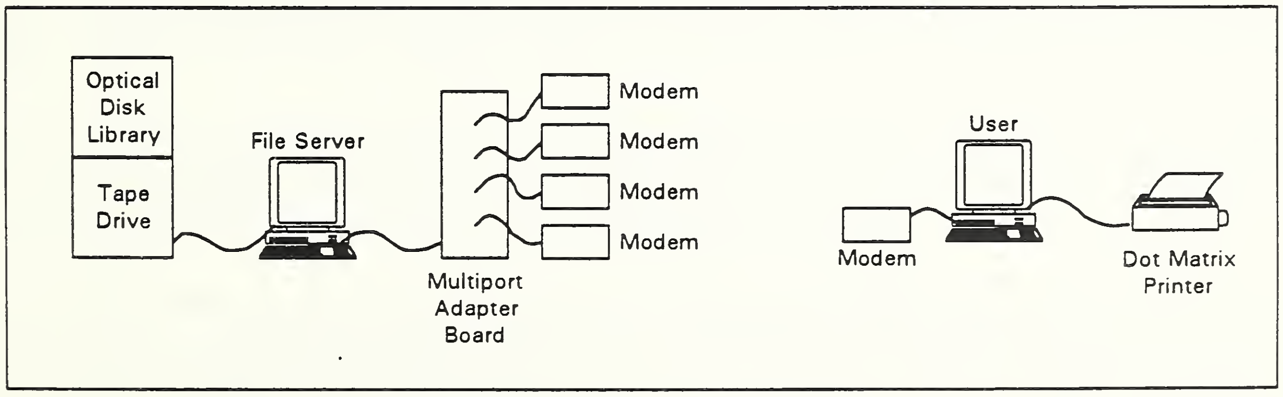


Figure 1. System Components

2.2 Software Components

The operating system for the file server is Interactive UNIX System V Release 3.2, Version. 2.2. With the exception of the optical disk library cartridge handler, the file server's peripherals are controlled by Columbia Data Products software drivers and UNIX utilities. NIST developed a software driver and scheduling algorithm to handle the requests for optical disk cartridges and the movement of the optical disk library cartridge handler. In addition, NIST developed an EASEAR user interface to allow the user to query for and to browse through the EASEAR record data. The Informix ESQL/C embedded SQL database engine is used to manage the EASEAR record data.

The operating system for the workstations is DOS 3.1 or above. Rappôrt, a DOS/UNIX bridge developed by Tundra Software, is used to communicate with the UNIX file server.

3. Pilot System Design

In order to provide SSA with the most feedback on the problems involved in the development of an automated records retrieval system, the design of the pilot system reflects a great effort to remain vendor-independent, while still consisting largely of "off-the-shelf" products. In the choice of components, NIST attempted to closely follow the existing and emerging standards in the following issues: removable optical disk media, Small Computer Systems Interface (SCSI) devices and programming interfaces, Structured Query Language (SQL) databases, and operating systems (see Reference Section).

The main goals of the pilot system were to demonstrate the feasibility of an automated retrieval system and to study the impact of the system on the users. Considering these goals, it was determined that the most crucial areas in the design of the pilot system were the operating system, the integration of a mass storage capability, the indexing methods used to retrieve records, and the user interface. This section of the document concentrates on the operating system and mass storage devices, while issues concerning the user interface and indexing methods are discussed in the software section of this document.

3.1 Operating System

The main criteria for the file server operating system was that it provided multiuser access, a POSIX-like interface, and supported removable optical disk media. With these criteria in mind, the following operating systems were considered: OS/2 extended edition, Novell Netware and UNIX. Based on the factors outlined below, NIST chose UNIX for the file server operating system.

While OS/2 would merge well with SSA's existing IBM mainframe hardware, it is still a fairly new operating system. Along with several other minor flaws in its current version, OS/2 does not currently provide adequate support for SCSI optical disk drives.

While Novell Netware has the most dominant networking operating system on the market, at this time it does not support removable optical media. According to Novell's development engineers, Novell will not guarantee that their operating system will work with removable optical media, and actually strongly recommended that Novell Netware and removable optical media not be used together.

UNIX had many advantages over the other operating systems. It is a well established operating system with a large market share, considerable power and flexibility, and an interface which supports the spirit of POSIX. UNIX supports optical media with no compromise to its efficiency, and was repeatedly recommended by industry developers for use with this type of media, drives and libraries.

Implementations of the UNIX operating system available for Intel 80386 microprocessor based systems were considered. NIST chose to use Interactive Systems Corporation's (ISC) Interactive UNIX System V Release 3.2.

3.2 Mass Storage Devices and Integration

The integration of a mass storage capability into the pilot system involved the consideration of a large number of issues. The following are the major areas that were researched: mass storage media, mass storage device integration methods, open reel tape drives, and automated library systems.

3.2.1 Mass Storage Media and Drives

Optical disk media currently offers the highest density removable, direct-access media and was chosen as the mass storage media for the pilot. The specific optical media selected was 130 mm (5¼ in) magneto-optic double sided media with 512 byte sectors and a capacity of 600 megabytes per cartridge. An international standard (ISO/IEC 10089) exists for this type of media, and the cost of the media, drives, and library systems were suitable for a pilot.

3.2.2 Device Integration Method

The device integration method includes the hardware interface as well as the software interface. The following sections discusses the Small Computer Systems Interface (SCSI), the pilot system hardware interface, and the Common Access Method (CAM), the pilot system software interface.

3.2.2.1 Small Computer Systems Interface

The Small Computer Systems Interface (SCSI) is a specification that defines a high-performance, general-purpose interface that allows a system to communicate with a wide variety of peripheral devices. Due to its flexibility, SCSI is a popular method for the integration of external mass storage devices. However, the interface has so many options that the integration of devices can be very difficult. In many cases, the physical connectors, as well as the command set used to control the devices, are vendor specific. Many "plug and play" subsystems have involved the use of proprietary boards and software, which lock a system into one manufacturer.

In order to promote interchangeability between vendors, Task Group X3T9.2, Lower Level Interfaces, has developed the SCSI-1 and SCSI-2 standards, and is currently working on a SCSI-3 standard. These standards provide specifications on the physical connectors and the command set, as well as many other characteristics that are necessary to allow for interchangeability.

SCSI compliant devices are daisy chained together and controlled by a single host bus adapter (HBA). Each device is assigned a target ID, via which the host bus adapter routes commands to the device. NIST chose to use the Western Digital 7000 FASST-2 SCSI host bus adapter.

3.2.2.2 Common Access Method

Another major problem in the integration of SCSI compliant devices is the programming interface. The Common Access Method (CAM) specification, being developed by the Task Group X3T9.2, defines a standard programming interface between an operating system and a host bus adapter. This interface provides a standard format for SCSI commands, regardless of the type of the device. For a device which is SCSI-compliant, a device driver following this specification would be written using high-level standardized commands rather than device specific SCSI commands. This enables the physical devices to be interchanged without having to rewrite code. This document is sufficiently complete that several software developers are already developing programming interfaces which comply to the CAM draft specification. NIST chose to use the Columbia Data Products SDLP CAM-compliant programming interface.

3.2.3 Open Reel Tape Drive

A 12.7 mm (½ in) open reel 256 cpmm (6250 cpi) 9-track tape drive was also an integral part of the pilot system as the EASEAR data was provided from SSA on this type of media. The Hewlett-Packard 88780B tape drive was selected as it provided an option for a SCSI interface.

3.2.4 Optical Disk Library System

In order to store all of the data intended for the pilot system, and to provide a response time acceptable for the number of users in the pilot, a library with approximately 20 gigabytes of storage capacity and two rewritable magneto-optic drives was required. It was also required that the optical drives and library cartridge handler be controlled by SCSI-1 or SCSI-2 commands. NIST chose to use two Hewlett-Packard C1710A rack-mountable library systems.

4. Software Design

The EASEAR pilot system software consists of a mix of off-the-shelf packages and software developed by NIST (see Figure 2). The off-the-shelf software consists of the ESQL/C database engine, the software drivers for the optical disk drives, as well as several UNIX utilities. The software written at NIST consists of two major programs, the EASEAR user interface and the library cartridge handler device driver. The software is written in the C Language, using a modular design approach, and compiled with the Interactive UNIX C compiler. The user interface is pre-compiled by the Informix ESQL/C precompiler, which allows the C language program to make SQL queries to the database engine.

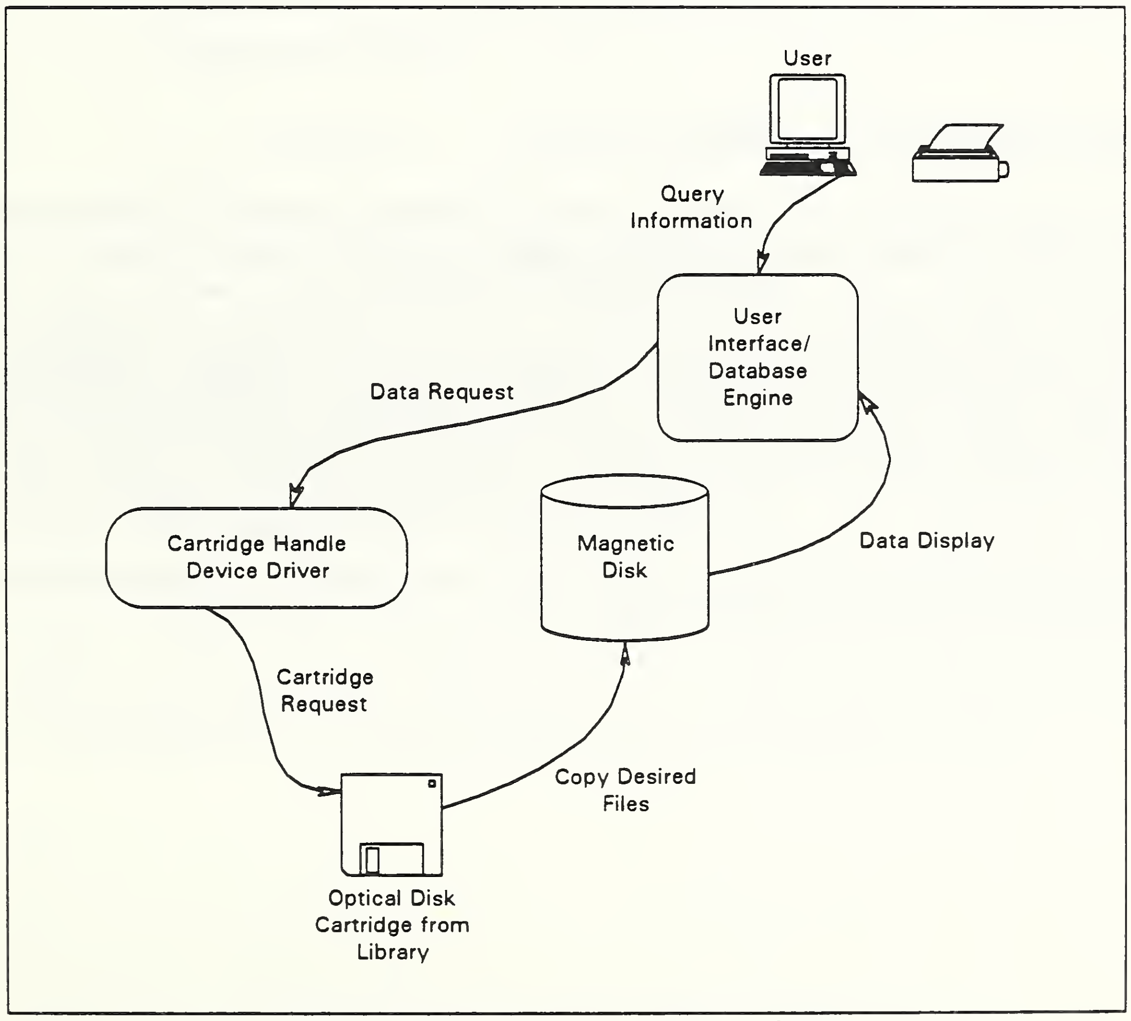

User
Query Information
User Interface/ Database Engine
Data Request
Cartridge Handle Device Driver
Magnetic Disk
Data Display
Cartridge Request
Copy Desired Files
Optical Disk Cartridge from Library

Figure 2. System Software Flow of Data

4.1 Overview of the EASEAR User Interface

Before writing the pilot system user interface, several visits were made to the Metro West branch of the Social Security Administration. From information obtained on a tour of the manual retrieval system currently in use, a prototype interface was designed and implemented for an IBM PC compatible running DOS. This prototype interface was used by approximately 20 users at SSA who were asked to offer suggestions to improve the user interface design. From these comments, the prototype was refined and taken back to SSA for a final prototype test.

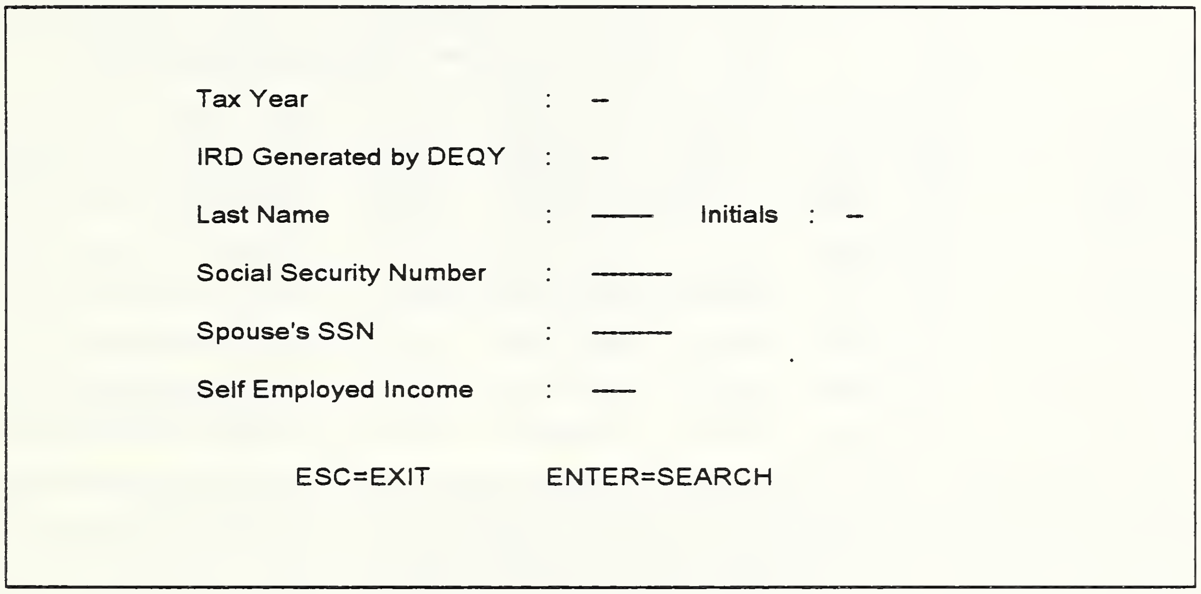
The EASEAR user interface provides record retrieval and viewing capabilities to the user. The user enters certain information on the record for which he or she is searching. This information is reordered into an SQL query, and records that match the query are retrieved. A limited number of fields from each record are displayed, and the user is allowed to browse through the records, select a record to be displayed in full, and select records to be printed. In addition to these functions, the user interface collects certain management information to help assess the results of the pilot system test. The listing of this code, "easear.ec", can be found in Appendix D.

4.1.1 General Screen Characteristics

The user interface consists of four types of screens: the query screen, the browse screen, the detail screen, and the error screen. The first screen, the query screen, is used to enter the parameters that define a query. The next screen, the browse screen, is used to view a subset of information on every record that matches the query. Both screens have a blue background and yellow text, and, in order to make available as much information as possible, are the size of the entire display. The detail screen, used to display full information on a selected record, and the error screen, used to report an error condition or status information to the user, have a red background and yellow text. These two screens are smaller than the display as only limited information is necessary, and overlay either the query or browse screen. With the exclusion of the error screen, there are prompts on the bottom of each of the screens outlining options available to the user while at the current screen.

4.1.2 The Query Screen

The query screen appears upon entry into the program, and prompts the user to enter parameters for a record search (see Figure 3). It is mandatory that the user specify the tax year, the internal revenue district (IRD), and at least four letters of the wage earner's last name. Optional information includes the initials, the social security number, the spouse's social security number and the amount of self employed income.



```
Tax Year                   :  _
IRD Generated by DEQY      :  _
Last Name                  :  ____     Initials  :  _
Social Security Number     :  ______
Spouse's SSN               :  ______
Self Employed Income       :  ___

        ESC=EXIT           ENTER=SEARCH
```

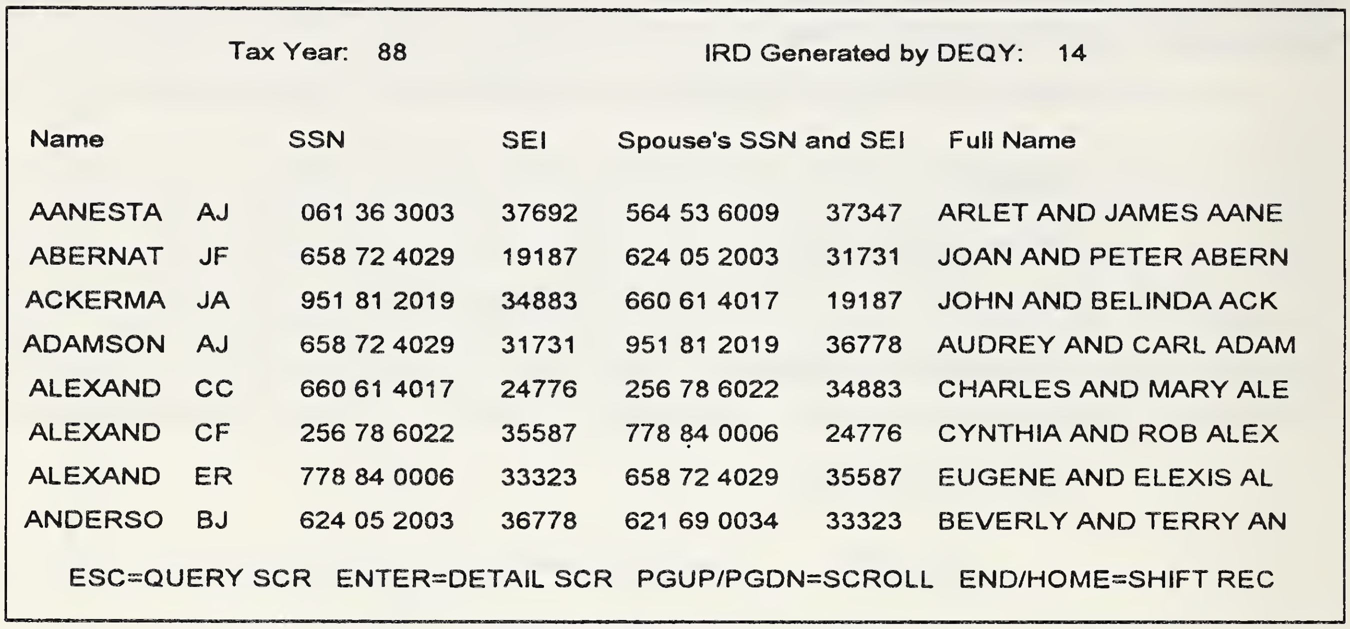
Figure 3. The query screen

If the user places a period at the end of the specified last name, only records in which the last name matches the specified pattern are retrieved. Otherwise, all records with the last name beginning with the specified character sequence are retrieved. If the user enters self employed income information, all records are retrieved in which the self employed income is in the range of +/- one dollar of the figure designated as a query parameter.

When the user has entered the maximum number of characters for a response to a parameter, the cursor jumps to the next field. The user also has the use of the arrow and tab keys to move between fields. The Page Up key is used to view the tax year and the IRD from the previous query. When the user presses enter, the program retrieves the records matching the query and proceeds to the browse screen. If the user presses escape, the program terminates.

4.1.3 The Browse Screen

The browse screen displays partial information on all of the records that match the user's query (see Figure 4). The information displayed is intended to be enough for the user to refine his or her search in order to request detailed information on a few records.



```
              Tax Year:   88                              IRD Generated by DEQY:   14

 Name                 SSN              SEI      Spouse's SSN and SEI      Full Name

 AANESTA    AJ     061 36 3003      37692     564 53 6009     37347    ARLET AND JAMES AANE
 ABERNAT    JF     658 72 4029      19187     624 05 2003     31731    JOAN AND PETER ABERN
 ACKERMA    JA     951 81 2019      34883     660 61 4017     19187    JOHN AND BELINDA ACK
 ADAMSON    AJ     658 72 4029      31731     951 81 2019     36778    AUDREY AND CARL ADAM
 ALEXAND    CC     660 61 4017      24776     256 78 6022     34883    CHARLES AND MARY ALE
 ALEXAND    CF     256 78 6022      35587     778 84 0006     24776    CYNTHIA AND ROB ALEX
 ALEXAND    ER     778 84 0006      33323     658 72 4029     35587    EUGENE AND ELEXIS AL
 ANDERSO    BJ     624 05 2003      36778     621 69 0034     33323    BEVERLY AND TERRY AN

    ESC=QUERY SCR   ENTER=DETAIL SCR   PGUP/PGDN=SCROLL   END/HOME=SHIFT REC
```

Figure 4. The browse screen

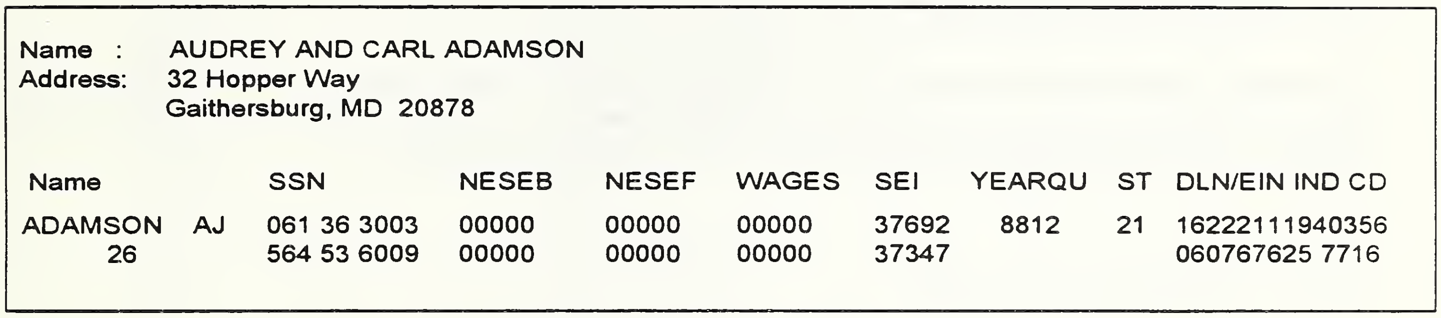
The the arrow keys are used to browse through the page of records currently on the screen, and the Page Up and Page Down keys are used to retrieve a previous page or a new page of matches.

The End key is used to view any portion of the record that is too long for the standard display, and the Home key is used to restore the line.

The user is able to view all records that match the query, up to the point where system performance would be impacted. By pressing Esc, the user is returned to the query screen. If the user presses Enter while the cursor is positioned over a record, he or she proceeds to the detail screen.

4.1.4 The Detail Screen

The detail screen displays all the information in the current record as it would appear on the microfilm (see Figure 5). If the user presses Enter, the record information is sent to the dot matrix printer at the user's workstation. If the user presses Print Screen, the entire screen is printed.

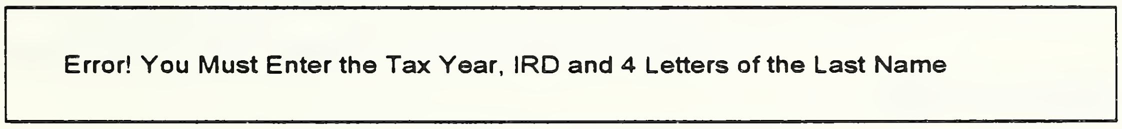


```
Name  :   AUDREY AND CARL ADAMSON
Address:  32 Hopper Way
          Gaithersburg, MD  20878

 Name            SSN          NESEB    NESEF    WAGES    SEI     YEARQU   ST   DLN/EIN IND CD
ADAMSON   AJ    061 36 3003   00000    00000    00000    37692    8812    21   16222111940356
     26         564 53 6009   00000    00000    00000    37347                 060767625 7716
```

Figure 5. The detail screen

4.1.5 The Error Screens

The error screens are used to inform the user that an error condition has occurred (see figure 6). When an error screen appears, the user presses Esc to exit the screen. Some error messages are caused by an improper entry, and return the user to the current screen. Others are system error messages, and cause program termination. Refer to Appendix C for a list of error messages.



```
Error! You Must Enter the Tax Year, IRD and 4 Letters of the Last Name
```

Figure 6. The error screen

4.1.6 Management Information

The EASEAR user interface maintains certain management information in order to help assess the results of the pilot system test. It collects the information, and stores it in a file. A separate program uses the information stored by the user interface and prints the following:

- Total logins to the pilot system
- Total time elapsed during all logins
- Average time elapsed during a login
- Total number of queries
- Total number of 1988 queries
- Total number of 1989 queries
- Total time elapsed during all queries
- Total time elapsed during 1988 queries
- Total time elapsed during 1989 queries
- Average time elapsed during a query
- Average time elapsed during a 1988 query
- Average time elapsed during a 1989 query
- Total requests for detailed records
- Total requests for 1988 detailed records
- Total requests for 1989 detailed records
- Total requests for record printouts
- Total requests for 1988 record printouts
- Total requests for 1989 record printouts

4.2 User Interface Software Organization

The following subsections discuss the data structures implemented in the user interface, as well as the high level software organization.

4.2.1 Software Data Structures

The major data components involved in the user interface consist of seven data structures: the query parameters, the browse data, the full record data, the Informix host variables, the request message, the acknowledge message, and the management information.

4.2.1.1 The Query Parameters

The Query Parameters data structure is implemented as an array of size two of the user-defined C structure INFO, and is used to store search parameters entered by the user. One element of the array is used to store the current query parameters, while the other array element is used to store the previous query parameters. The structure contains the following data:

- Tax Year
- IRD
- Last name
- Initials
- Social Security Number
- Spouse's Social Security Number
- Self Employed Income

4.2.1.2 The Browse Data

The Browse data structure is implemented as an array of size MAXSCR (number of matches displayed on the screen in the user interface) of the user-defined C structure QUERY, and is used to store the fields of the record that the user views in the browse screen. The structure contains the following data:

- Tax Year
- IRD
- Last Name
- Initials
- Social Security Number
- Self Employed Income
- Spouse's Social Security Number
- Spouse's Self Employed Income
- Full Name as Reported on the 1040 form

4.2.1.3 The Full Record Data

The Full Record data structure is implemented as an array of size MAXSCR (number of matches displayed on the screen in the user interface) of the user-defined C structure REC, and is used to store the remainder of the record fields which the user views in a detail screen. The structure contains the following data:

- NESEB
- NESEF
- Wages
- Month
- Quarter
- Option Code
- DLN
- IWP
- Error Code
- Spouse's NESEB
- Spouse's NESEF
- Spouse's Wages
- Date
- EIN
- Industry Code
- Street Address
- City and State
- Zip Code

4.2.1.4 The Informix Host Variables

The Informix Host variables data structure is implemented as the user-defined C structure RECEIVE, and is precompiled by the Informix ESQL/C database engine. It is used as a set of variables for the passing and receiving of data between the user interface and the embedded database engine. This is a separately defined data structure because the database engine has problems dealing with an array of structures serving as a host variable. It contains all of the information in the QUERY and REC user-defined structures.

4.2.1.5 The Request Message

The Request Message data structure is implemented as the user-defined C structure MSGBUF, and is used to pass requests for mounting and unmounting cartridges to the cartridge handler device driver. The structure contains the following data:

- Message Type (mount or unmount)
- User Number
- Library Number
- Cartridge Number
- Cartridge Side

4.2.1.6 The Acknowledge Message

The Acknowledge Message data structure is implemented as the user-defined C structure MSGACK, and is used to receive acknowledgement of a mounted cartridge from the cartridge handler device driver. The structure contains the following data:

- Message Type (Acknowledge)
- Directory to which the cartridge has been mounted

4.2.1.7 The Management Information

The Management Information data structure is implemented as the user-defined C structure MANAGEINFO, and is used to collect information that will be used to assess the effectiveness of the pilot system. The structure contains the following data:

- Number of logins to the pilot system
- Length of time the user has been logged into the pilot system
- Number of 1988 and 1989 queries
- Length of time the user is involved in a 1988 and 1989 query
- Number of 1988 and 1989 detailed screens viewed
- Number of 1988 and 1989 printouts requested

A separate program, "print_mi.c", uses this information to calculate and print the parameters listed in section 4.1.6. The code for this program is listed in Appendix D.

4.2.2 High Level Software Organization

The user interface software organization consists of five distinct phases: the initialization and control of the user environment, the entry of the user query, the selection of the matches to the query, the display and viewing of the browse fields, and the display of detailed record information (see Figure 7). These phases are discussed in the following sections.

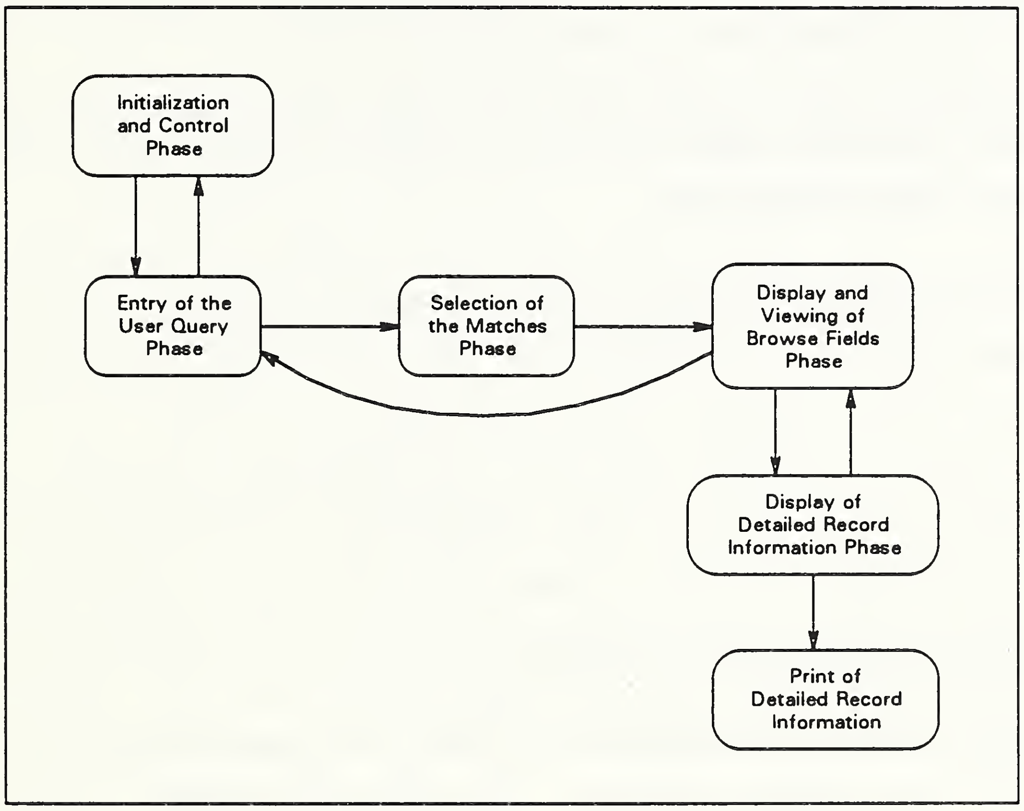

Initialization and Control Phase
Entry of the User Query Phase
Selection of the Matches Phase
Display and Viewing of Browse Fields Phase
Display of Detailed Record Information Phase
Print of Detailed Record Information

Figure 7. User interface software control flow

4.2.2.1 Initialization and Control of the User Interface Environment

The user interface uses a set of library routines in the UNIX compiler referred to as the "curses" library. These routines provide windowing, screen handling, and cursor control functions, and are used to draw and manage the user interface screens for character oriented displays. In the initialization phase, the screen windows are set up, the management information is initialized, the user number is determined based on the port the user is logged into, the length of login timer is started, and the login count is incremented.

The entry of the query phase is then entered by a call to the get_query() function. Control is returned to initialization phase when the get_query() function returns an escape character, signalling the end of a session. At this point, the login timer is ended, the management information is collated and written to a file, the windowing environment is exited and the program terminates.

4.2.2.2 Entry of the User Query

In the Query Entry Phase, the get_query() function performs the following steps:

- initializes the length of query timer
- increments the query count
- draws the query screen
- traps and interprets each character entered by the user

If the user presses Esc, the program returns to the initialization phase and the query timer is discarded. If the user presses Enter, the select_matches() function is called, and the program proceeds to the match selection phase.

All characters, excluding escape and enter, pass to the function interp(), which either processes the input or routes it to the correct lower level function. If the character is an arrow key, the character passes to the proc_arrow() function, which moves the cursor in the direction of the arrow. If the character is a tab key, the cursor moves to the next field. If the character is a Page Up key, the tax year and the IRD from the previous query are displayed. If the character is a backspace, it passes to the function del_char(), which deletes the last character entered, from both the screen and the INFO data structure. Otherwise, the character passes to the function add_char(), which adds the character to both the screen and the INFO data structure.

4.2.2.3 Selection of the Matches to the Query

In the Selection of Query Matches Phase, the INFO data structure containing the user query passes to the select_matches() function. The last name parameter is checked by the check_search_type() function to determine if the search type is prefix or exact match. The assign_user_files() function is called to allocate files for use by this user, and the req_user_data() function is called to request files from the optical disk library.

The req_user_data() function queries the indexing tables for files that contain the IRD, last name and initials, if specified, that match the data in the INFO structure. If the number of files that meet this criteria is greater than twenty, an error occurs and the user is asked to refine his or her query and control returns to the entry of the query phase. Otherwise, the cartridge containing the files is requested from the cartridge handler scheduling algorithm. Upon receipt of an acknowledge message, the files are copied from the cartridge to the magnetic hard disk and decompressed, and the scheduling algorithm is notified that the use of this cartridge is complete. The program then returns to the select_matches() function.

At this point, the data in the INFO structure are concatenated into a string which forms the SQL query. The user's database is then opened, and all matches to the query are retrieved. If no matches are found, an error message is displayed, and program returns to get_query function() and the entry of the query phase. If matches are found, the display_matches() function is called, and control of the program continues to the display and viewing of the browse fields.

4.2.2.4 Display and Viewing of Browse Fields

In the Display and Viewing of Browse Fields Phase, the display_matches() function draws the query screen and each character entered by the user is trapped and interpreted. If the user presses Esc, the length of query timer is ended, and the program returns to the get_query() function and the query entry phase. If the user presses enter, the disp_rec() function is called, and the program proceeds to the detail view phase.

All characters, excluding escape and enter, are passed to the function browse(), which either processes the input or route it to the correct lower level function. If the character is an arrow key, the cursor is moved in the direction of the arrow. If the character is a page up, the view_previous_matches() function is called and the previous page of matches is retrieved, if applicable. If the character is a page down, the view_next_matches() function is called and the next set of matches is retrieved, if applicable. If the character is an end key, the move_line() function is called and the rightmost portion of the current record is displayed. If the character is a home key, the restore_line() function is called and the leftmost portion of the current record is displayed.

4.2.2.5 Display of Detailed Record Information

In the display of detailed record information phase, the disp_rec() function draws the detail screen, increments the detail view count, and both traps and interprets each character entered by the user. If the user presses Esc, the program returns to the display_matches() function and the Query Entry Phase. If the user presses Enter, the program calls the print_rec() function, increments the print count, and prints the record. All other characters are disregarded.

4.3 Overview of the Cartridge Handler Device Driver

The Helwett-Packard optical disk library system contains two 130 mm (5¼ in) magneto-optic drives, a cartridge handler device and 32 cartridge slots. The Columbia Data Products software drivers, in combination with several UNIX utilities, provide adequate support for the two drives.

No off-the-shelf software support existed for the cartridge handler device in this environment, therefore it was necessary to develop the software at NIST. The device driver consists of two major algorithms: a scheduler algorithm and a control algorithm. The scheduler algorithm creates and maintains a messaging interface with all copies of the EASEAR user interface that are running, and handles their requests for optical disk cartridges. The control algorithm deals with the movement of the cartridge handler device. The listing of this code, "devhandler.c", can be found in Appendix D.

The cartridge handler device driver must be run with system administrator privileges in the background. This is necessary in order to have the correct permissions to mount and unmount file systems.

4.3.1 The Scheduler Algorithm

The scheduler algorithm communicates with the EASEAR user interface through a form of UNIX interprocess communication referred to as messaging. After setting up the messaging interface, the scheduler waits for a request from the user interface. The messages are one of two types - a request to mount a cartridge or to unmount a cartridge.

Before checking the incoming messages, the scheduler first checks to see if any drives are available. If no drives are currently free, the scheduler waits until an unmount request is placed in the queue and services it before retrieving any mount requests.

When a mount request is received, the scheduler checks to see if the cartridge is available. If so, a drive is allocated, the cartridge is moved into the drive and mounted, and an acknowledge message is sent to the user interface. If the requested cartridge is currently in use, the request is placed in the wait queue.

When an unmount request is received, the scheduler first checks the wait queue to see if any requests are pending for the cartridge in question. If there are no requests, the cartridge is unmounted and returned to its library slot. If a request is pending, the scheduler ascertains which side of the cartridge is requested, flips the cartridge if necessary, and sends an acknowledge message to user interface.

4.3.2 The Control Algorithm

The cartridge handler device is controlled through the Columbia Data Products Standard Level Device Protocol (SDLP) interface. This interface, for supported types of devices, removes the necessity to send SCSI commands to the device. Instead, higher level commands are sent to an SDLP driver which generates the SCSI commands. As the cartridge handler is not currently a supported device for the SDLP interface, it was necessary to use the pass-through mode to bypass the SDLP interface and send SCSI-2 commands directly to the cartridge handler.

When the scheduling algorithm requests that a cartridge be moved, the control algorithm creates a command descriptor block (CDB) that contains the SCSI-2 command for move medium, and sends an SDLP command to the Columbia Data Products SDLP driver requesting that this CDB be passed through to the cartridge handler device.

4.4 Cartridge Handler Device Driver Software Organization

The following subsections discuss the data structures implemented in the cartridge handler device driver, and the high level software organization.

4.4.1 Software Data Structures

The major data components involved in the cartridge handler device driver consist of six data structures: the message queues, the request message, the acknowledge message, the drive status information, the command descriptor block, and the SDLP command information.

4.4.1.1 The Message Queues

The Message Queues are implemented as a C system-defined structure, and are managed by the UNIX operating system. The messages in the queue are implemented as a system-defined linked list that is also managed by the UNIX operating system. The exact format of these data structures is found in the UNIX C library header files "ipc.h" and "msg.h". For the purposes of this documentation, the queues and messages are viewed as a structure pointer that is returned by the system.

The cartridge handler device driver utilizes three message queues -- the incoming messages queue, the wait queue, and the acknowledge queue. The incoming messages queue is used to receive cartridge requests from the user interface. The wait queue is used to hold cartridge requests that cannot be filled immediately due to the cartridge being in use. The acknowledge queue is used to send acknowledgement of an available cartridge to the user interface.

These queues are created by the scheduler but are not released by the program as the scheduler operates continuously. If, for any reason, the scheduler is terminated, it is necessary to release the messaging queues before restarting the program. There is a small program, "killq.c", which releases the queues. The listing of this code is located in Appendix D.

4.4.1.2 The Request Message

The Request Message data structure is implemented as the user-defined C structure MSGBUF, and is used to receive requests from the user interface for the mounting and unmounting of cartridges. The structure contains the following data:

- Message Type (mount or unmount)
- User Number
- Library Number
- Cartridge Number
- Cartridge Side

4.4.1.3 The Acknowledge Message

The Acknowledge Message data structure is implemented as the user-defined C structure MSGACK, and is used to send acknowledgement of a mounted cartridge to the user interface. The structure contains the following data:

- Message Type (Acknowledge)
- Directory to which the cartridge has been mounted

4.4.1.4 The Drive Status Information

The Drive Status information is implemented as an array size MAXDRIVES (the number of drives in the pilot system) of the user-defined C structure DRINFO, and is used to maintain the current state of each of the drives, as well as system and address information on the drive. The following information is kept for each of the drives:

- The SDLP unit number of the jukebox in which it resides
- The SDLP unit number of the drive
- The file name of the software driver for the drive
- The directory to which this drive mounts a cartridge
- A flag indicating whether or not the drive is in use
- The cartridge number of the cartridge currently in the drive
- The side of the cartridge currently in the drive

4.4.1.5 The Command Descriptor Block

The Command Descriptor Block is implemented as an array size 12 of characters, and is used to store the move medium SCSI command and related information needed by the cartridge handler device. It contains the following information:

- SCSI command (0xA5 - move media command)
- Logical Unit Number of target device
- Cartridge Handler Device Address
- Source Element Address
- Destination Element Address
- Invert Flag

4.4.1.6 The SDLP Command Information

The SDLP Command Information is implemented as a user-defined C structure, and is used to store information needed by the Columbia Data Products SDLP driver in order to execute a pass-through move medium SCSI command. The structure contains the following information:

- Command for the SDLP device driver to execute (0xff - passthrough)
- SDLP unit number of the device for which this command is intended
- Address of the return buffer (for commands that return information)
- Length in bytes of the return buffer
- Address of the Command Descriptor Block containing SCSI command
- Length in bytes of the Command Descriptor Block
- I/O direction of the command (read or write)
- Type (always 0)
- Error Status

4.4.2 High Level Software Organization

The cartridge handler device driver organization consists of three distinct phases: the initialization of the data structures, the receipt of a request, and the processing of a request (see figure 8). These phases are discussed in the following sections.

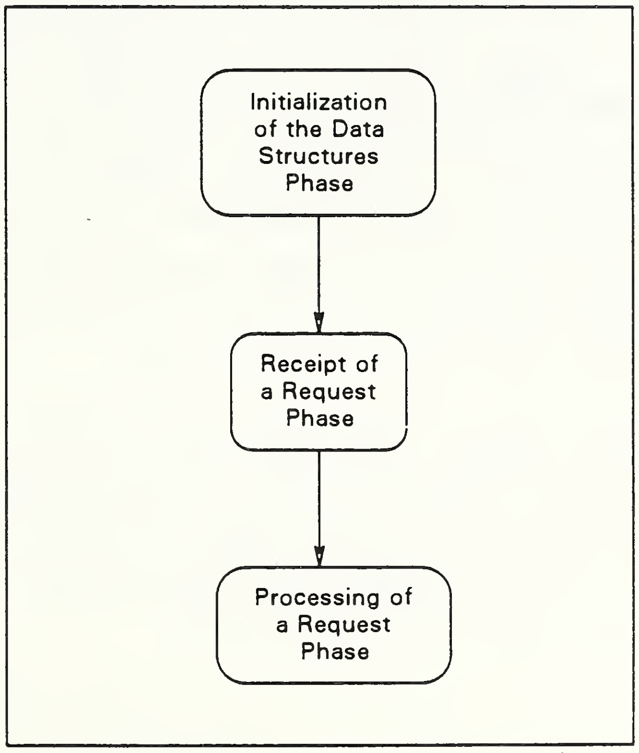

Initialization of the Data Structures Phase
Receipt of a Request Phase
Processing of a Request Phase

Figure 8. Cartridge handler device driver software control flow

4.4.2.1 Initialization of the Data Structures

In the initialization phase, the messaging environment is created and the control data structures are initialized. The incoming messages, wait, and acknowledge queues are created, the SDLP device driver is opened, and the DRINFO data structure is initialized to contain starting information for each of the drives. Control passes to the receipt of a request phase, which runs continuously.

4.4.2.2 Receipt of a Request

In the receipt of a request phase, the DRINFO data structure is checked to determine if any drives are available. If a drive is free, the program calls the msgrcv() function and waits for the next message to be placed in the queue. If no drives are free, the msgrcv() function waits until the next unmount message is placed in the queue. Once a message is received and stored in the MSGBUF data structure, control passes to the processing of a request phase.

4.4.2.3 Processing of a Request

In the processing of a request phase, the MSGBUF data structure is checked to determine which kind of request has been received.

If the request is a mount request, the DRINFO data structure is checked to determine if the requested cartridge is free. If it is currently in use, the MSGBUF data structure is sent to the msgsnd() function and the message is placed in the wait queue. If the cartridge is free, the free_drive() and mark_busy() functions are called to allocate a drive, and the mov_disk() function is called in order to move the cartridge to the allocated drive. At this point, a hardware response delay is caused to allow the cartridge to be inserted into the drive, and then the mount_disk() function is called to mount the file system. If the process is not delayed, the mount_disk() function will fail. Once the file system is mounted, the MSGACK data structure is filled and sent to the msgsnd() function and the message is placed in the acknowledge queue. At this point, control is returned to the receipt of a message phase.

If the request is an unmount request, the determine_drive() function is called to ascertain which drive contains the cartridge to unmount. The check_wait_queue() function is then called to see if any messages in the wait queue need this cartridge. If not, the mark_idle() function is called to change the status of the drive to idle, the unmt_disk() function is called to unmount the file system, and the ret_disk() function is called to return the cartridge to the library slot. At this point, control is returned to the receipt of a message phase.

If a message in the wait queue is waiting for this cartridge, the MSGBUF data structure is filled with the pending message, and is checked to determine if the current side of the cartridge is needed. If so, the MSGACK data structure is filled and sent to the msgsnd() function and the message is placed in the acknowledge queue. If the opposite side is needed, the unmt_disk() function is called to unmount the file system, the mark_busy() function is called to update the cartridge information stored in the DRINFO data structure, the flip_disk() function is called to flip the cartridge in the drive, and, after a short hardware response delay, the mount_disk() function is called to mount the file system. The MSGACK data structure is filled and sent to the msgsnd() function and the message is placed in the acknowledge queue. At this point, control is returned to the receipt of a message phase.

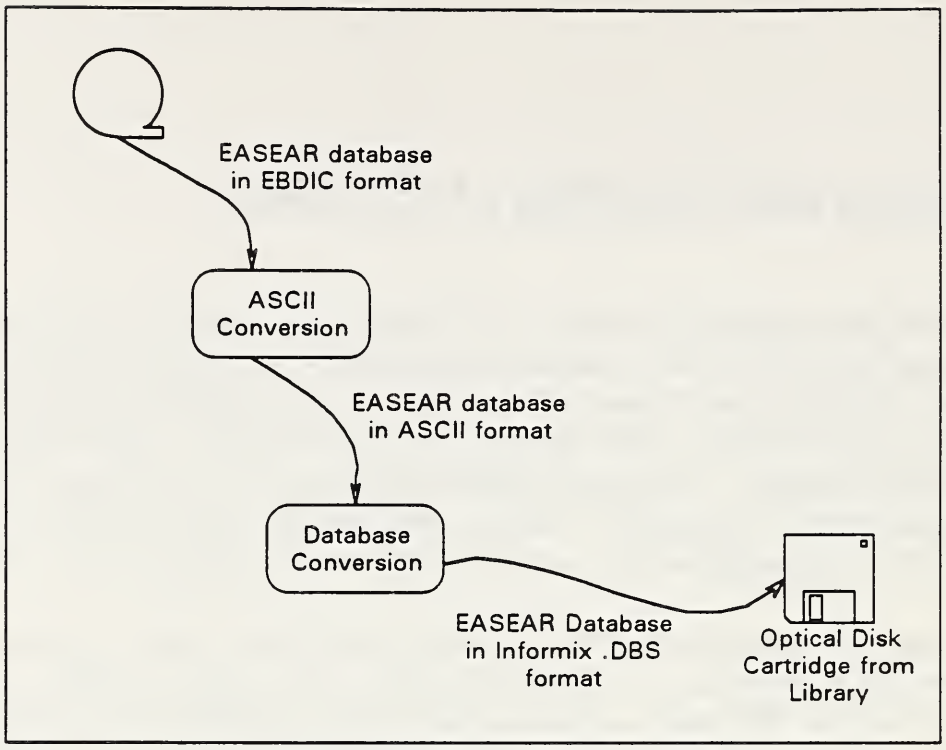
5. Conversion and Handling of SSA Data

The EASEAR data was received from SSA on 12.7 mm (½ in) open reel 256 cpmm (6250 cpi) 9-track tape. Each year of the data contained approximately 11 million records, and was stored as a multi-volume file split over 17 tapes. The data was in the EBCDIC character set, with each tape containing a header section, a data section, and a trailer section. Each section is separated from the others with filemarks. The tapes contained between 60 to 107 blocks, with 31824 bytes each and separated with a '<' character. The data itself was split into records of 234 bytes each, and was organized in increasing numeric order by IRD, and in alphabetic order within IRD.

The EASEAR data was manipulated at NIST until it was finally stored as database files of 300 records using the ASCII character set. Each of the files contains only one IRD, and is assigned a name beginning with the IRD and a sequential file number starting at one for each IRD. The files are stored on 130 mm (5¼ in) magneto-optic cartridges in a compressed format. In its final compressed state, one year of data consumes approximately one gigabyte of optical disk storage.

The steps in the conversion of the data include the following (see Figure 9):

- transfer the data from tape to optical disk cartridge
- convert the data to the ASCII character set
- analyze the data to determine which IRD's resided in each of the files
- separate the data into files of 300 records, with only one IRD per file
- convert the data to Informix Relational Database Table format
- compress the database tables


EASEAR database in EBDIC format
ASCII Conversion
EASEAR database in ASCII format
Database Conversion
EASEAR Database in Informix .DBS format
Optical Disk Cartridge from Library
Figure 9. Data conversion process

5.1 Basic Procedures

There are some basic system procedures required for the system manager to convert the data. These include inserting a cartridge into a library drive, ejecting a cartridge from a library drive, logging into an account on the UNIX system, gaining system administrator privileges, formatting an optical disk cartridge, placing a file system on an optical disk cartridge, mounting a file system, unmounting a file system and logging out of an account.

5.1.1 Inserting a Cartridge into a Library Drive

Insert the cartridge into the library mailbox slot. The side of the cartridge to be used by the drive should be facing to the right. Press the **load** button and then use the **next** and **previous** buttons until the desired drive number is shown in the display. Press the **enter** button. The cartridge is loaded into the selected drive.

5.1.2 Ejecting a Cartridge from a Library Drive

Press the **eject** button to remove the cartridge from the drive, and use the **next** and **previous** buttons to select the correct drive. Press the **enter** button. The cartridge is unloaded from the selected drive and placed in the library mailbox slot.

5.1.3 Logging into an Account on the UNIX System

At the *login:* prompt, type the account name. The system prompts for the password. Once the valid password is entered, the system prompt, a %, is displayed.

5.1.4 Gaining System Administrator Privileges

At the system prompt, type: **su root**. The system prompts for the password to this account. Once the valid password is entered, the root system prompt, a #, is displayed.

Note: The super-user account is a privileged account on the UNIX system. While logged in as the super-user, there are **no** restrictions on what can be done to the system. **Extreme caution should be exercised while in this account.**

5.1.5 Formatting an Optical Disk Cartridge

Place the cartridge into the library drive, and log into the system. At the system prompt, type: **/usr/sst/dutil**. At the menu, select the device corresponding to the drive in which the cartridge resides. When prompted to select an action, choose option 7, Format. When asked to enter an interleave value, enter **0**, Default interleave value. When asked whether to continue with format or to quit, choose **Yes**, continue with format.

The screen says: *Formatting Device #...* and when the format is complete, the screen says: *Format of Device # Successful, press any key.*

When asked to select an action, choose option **9** to exit, then choose Q to quit. At the "%" prompt, log out of the system and remove the cartridge from the library drive.

5.1.6 Placing a File System on an Optical Disk Cartridge

Place the cartridge into the library drive, and log into the system. At the system prompt, if the cartridge is in drive number one, type: **mkfs /dev/sstb0 576998**. If the cartridge is in drive number two, type: **mkfs /dev/sstb1 576998**.

The screen prompts with *mkfs /dev/sstbx (Del if wrong)* and pauses for 10 seconds. Press the delete key if the wrong device name has been entered, and re-enter the command as above.

If the process is not stopped, the screen displays file size information such as logical block size, number of logical blocks, etc., and places a file system on the cartridge. This takes approximately 15 minutes. When completed, the screen displays the total available blocks and returns to the "%" prompt. Log out of the system and remove the cartridge from the library drive.

5.1.7 Mounting a File System

To mount a file system, the system manager must have system privileges. Insert the cartridge into the library drive. At the # prompt, type: **mount /dev/sstbx /directory**, where **x** is 0 if the cartridge is in drive one or 1 if the cartridge is in drive two, and **directory** is the name of the directory to which to mount the cartridge in the drive.

The system then mounts the file system on the cartridge to the specified directory. This takes approximately 5 minutes. When complete, the screen displays *warning: < > mounted as </drive0>*.

5.1.8 Unmounting a File System

To unmount a file system, the system manager must have system privileges. At the # prompt, type: **umount /dev/sstbx** where **x** is 0 if the cartridge is in drive one, or 1 if the cartridge is in drive two. The system then unmounts the file system. This takes approximately 5 minutes.

5.1.9 Logging out of an Account

At the system prompt, type: **exit** to leave the account.

5.2 Conversion from Tape to Optical Disk

The first step in the conversion of the EASEAR data was to copy the data from tape to optical disk. The data were also converted to the ASCII character set, and split into smaller files.

Moving the data to optical disk rather than leaving it on magnetic tape made the data conversion process much easier for our system as this provided direct-access retrieval of the data. The data were copied from the tape and converted to ASCII using the UNIX utility "dd". The actual code for this procedure is the shell script "downld", a listing of which is in Appendix D.

The script deals with the data in 50 block sets of data. The information is read from the tape, converted to ASCII, and stored in a file on the optical disk whose file name corresponds to the current value of the loop counter within the script. The resulting file size on the optical disk is approximately 1.5 megabytes.

5.2.1 System Procedures

Take the tape drive off-line, and press **Unload** to open the tape hatch. Remove the tape from its case and insert the tape into the drive. Close the hatch, and wait for the tape to load. The display on the drive reads *Ready* when complete.

Place the cartridge into the library drive, and log into the system. At the % prompt, type: **/usr/sst/dutil**. At the menu, when prompted to select a device, choose the number of the device corresponding to the tape drive. When prompted to specify an action, choose option **6**, File mark. When asked to go to a file mark or to write a filemark, choose option **g**, Go to a file mark.

The screen displays *Searching for filemark*, and once it is found, displays *Filemark found.* When asked to select an action, choose option **9** to exit and then choose option Q to quit.

Mount the destination cartridge to /drive0, and type: **downld (#sets)**, where a set is defined as 50 blocks of data on the tape. Round the value up to the nearest integer. This converts the tape data from EBCDIC to ASCII, and stores the data on the optical cartridge with a filename corresponding to the set number. The results of the transfer of each set are printed on the screen.

Unmount the cartridge from the library drive, remove the cartridge from the library drive, and log out of the system.

Take the tape drive off-line, and press the **rewind/unload** button on the drive twice to rewind and unload the tape. The tape hatch door opens when rewind/unload is complete.

5.3 Data Analysis

The next step in the data conversion was to analyze each of the files in order to determine which IRD's resided in each of the files, and to create a file listing the IRD's found in each file.

The program reads in each record, one at a time, and checks the value of the IRD in the record. If the IRD is the same as that of the previous record, the program continues to the next record. If the value is different from the previous record, the IRD and the current filename is written to a file which is named irdfile.x, where x stands for the tape number (see Figure 10). This analysis is done to expedite the next process, the separating of the data into smaller files by IRD. The code for this procedure is the C program "check.c", listed in Appendix D.

```
01    /drive2/0
01    /drive2/1
01    /drive2/2
01    /drive2/3
01    /drive2/4
01    /drive2/5
01    /drive2/6
01    /drive2/7
01    /drive2/8
01    /drive2/9
01    /drive2/10
02    /drive2/10
02    /drive2/11
```

Figure 10. IRD file

5.3.1 System Procedures

Insert the cartridge into the library drive, log into the system, and mount the cartridge to /drive0.

At the % prompt type: **ls -l /drive0**. This provides a directory of the files on the cartridge. Look for the highest numbered file, and add one to this number. This is the number of files to analyze.

At the % prompt, type: **check**. The program prompts for the number of files to analyze, and the tape number from which this data originated.

The program analyzes each of the files in turn and a list of the IRD's contained in each file is written to the file "irdfile.x", where x corresponds to the tape number.

Unmount the file system, remove the cartridge from the library drive, and log out of the system.

5.4 Separation of the Data

The data was then separated into files of 300 records in length. This number of records was chosen to provide maximum effectiveness for compression and decompression, both in speed and size, as well as trying to transfer from optical disk cartridge the fewest number of records in order to meet a user's queries.

An index file, startend.x where x is the IRD, is generated for each of these files (see figure 11). The index file contains the starting last name and initials, the ending last name and initials, and the filename of each output file. The code for this procedure is the C program "sep.c" listed in Appendix D.

```
CAISSIE     AM   CAMPBEL  RS   /drive2/0132
CAMPBEL     RW   CANOLIE  AR   /drive2/0133
```

Figure 11. Index file

The program accepts as input a starting filename and the total number of files to search. For each of the files, the records are read from the input file and written to an output file. A record count is maintained, and when the number of records written to one output file reaches 300, a new output file is started. The output files are named with the IRD and a sequential file number that starts at zero for each IRD. An entry is made in the IRD's index file for the first record and the last record of each output file. Records that contain an IRD lower than the IRD for which the program is searching are skipped. When a record is encountered with an IRD higher than the search IRD, the program terminates.

5.4.1 System Procedures

Place the source cartridge in the library drive two, and the destination cartridge in the library drive one. Login to the system and mount the source cartridge to /drive2 and the destination cartridge to /drive1.

At the % prompt, type: **sep**. When prompted, enter the IRD for which to search, the total number of files to search, and the file number with which to start. This information may be obtained from the data analysis report generated in the previous step.

The data that corresponds to the selected IRD is extracted from the larger files and stored on the destination cartridge in small files. At the same time, indexing information for this IRD is stored in a file called "startend.x", where x corresponds to the IRD. If the data for this IRD is contained on more than one cartridge, the system manager is prompted to change cartridges. Go to another terminal, unmount and remove the current cartridge, insert and mount the next cartridge. The system manager is again be prompted for the number of files to search, and the file number with which to start.

Unmount the file systems, remove the cartridges from the library drives, and log out of the system.

5.5 Conversion of Data to Informix Database Table Format

The files separated by IRD are then converted to Informix database table format, and indexed by last name. The program accepts as input a list of file names, and for each file creates a database and an index file. The records are then read in one by one and stored in the table. The database table and index file are copied to the destination cartridge, and the table on the hard drive is dropped. The code for this procedure is the C program "conv.ec", listed in Appendix D.

The last name and initials index files generated in the separating step are converted to database table format and stored on the magnetic media. In addition, fields are added to these tables to indicate on which cartridge and side the data are stored. The code for this procedure is the C program "index.ec", listed in Appendix D.

5.5.1 System Procedures

Place the source cartridge in the library drive two, and the destination cartridge in the library drive one. Login to the system and mount the source cartridge to /drive2 and the destination cartridge to /drive1.

At the % prompt, type: **ls > /usr/pilot/names**. This generates the list of input file names that the program expects.

Type: **conv** Each file is converted and the database and index files are copied to the destination cartridge.

Unmount the file systems, remove the cartridges from the library drives, and log out of the system.

5.6 Compression of the Data

The last step in the data conversion process is the compression of the database and index files. The compression algorithm used is the Huffman encoding scheme, which is included in the UNIX operating system as a utility. The code for this step is the c program "compress.c" listed in Appendix D.

5.6.1 System Procedures

Place the source cartridge in the library drive two, and the destination cartridge in the library drive one. Login to the system and mount the source cartridge to /drive2 and the destination cartridge to /drive1.

At the % prompt, type: **cd /drive2**, then type: **ls > /usr/pilot/names** and then type: **cd /usr/pilot**. This generates the list of input file names that the program expects.

Type: **comp < names**. Each file is copied to the destination cartridge and compressed.

Unmount the file systems, remove the cartridges from the library drives, and log out of the system.

6. References

American National Standard for Information Systems - Small Computer Systems Interface (SCSI), ANSI X3.131-1990.

Federal Information Processing Standard Publication - Database Language SQL, FIPS Publication 127-1.

International Standard ISO/IEC 10089, Part A, Information Technology - 130 mm Rewritable Optical Disk Cartridges for Information Interchange.

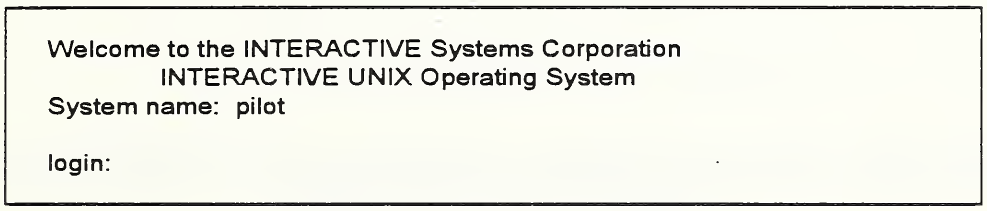
SCSI Common Access Method - Transport and SCSI Interface Draft Document, X3T9.2 90-186 Revision 2.3.

Federal Information Processing Standard Publication - POSIX: Portable Operating System Interface for Computer Environments, FIPS Publication 151-1.

Appendix A

User Instructions

A.1 Entering the System



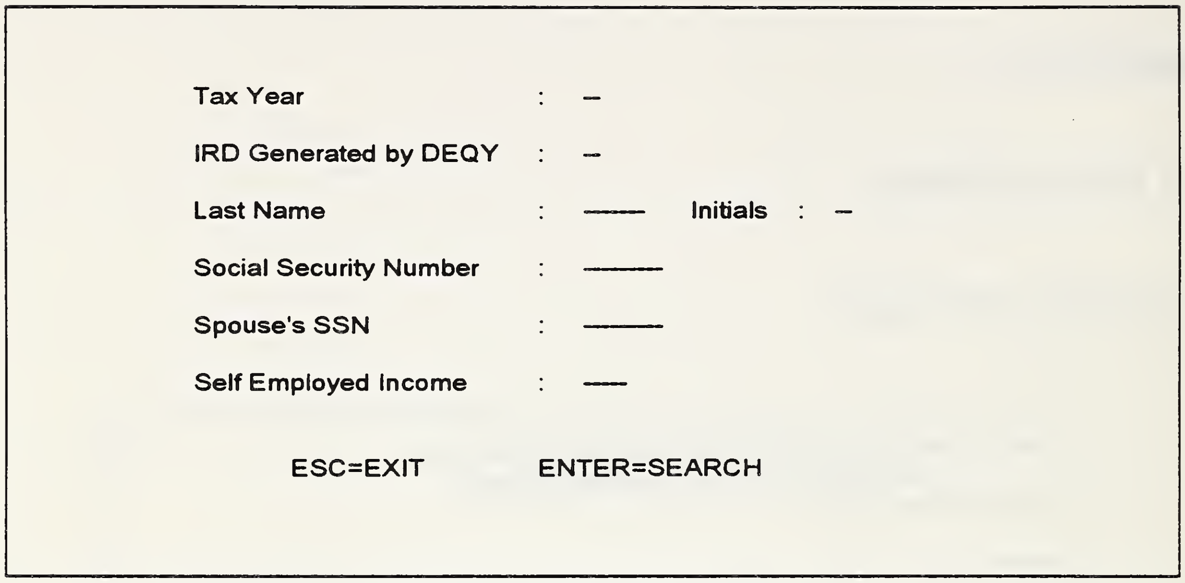
```
Welcome to the INTERACTIVE Systems Corporation
         INTERACTIVE UNIX Operating System
System name:  pilot

login:
```

Figure A1. The login screen

In Order to ...

- Login to the system, at the *login:* prompt, type the account name. At the *Password:* prompt, type the password for the account.

A.2 The Query Screen



```
Tax Year                   :  _

IRD Generated by DEQY      :  _

Last Name                  :  _____    Initials  :  _

Social Security Number     :  _______

Spouse's SSN               :  _______

Self Employed Income       :  ____

        ESC=EXIT           ENTER=SEARCH
```

Figure A2. The query screen

Mandatory Search Parameters:	Tax Year IRD generated by the DEQY system Last name
Optional Search Parameters:	Initials Social Security Number Spouse's Social Security Number Self Employed Income

Rules: A period is placed at the end of the last name field to indicate a search for an exact match of this last name. In other cases, the last name parameter is treated as a prefix, and a search is made for last names beginning with this sequence of characters.

At least four letters of the last name must be entered, unless a search for an exact match is indicated.

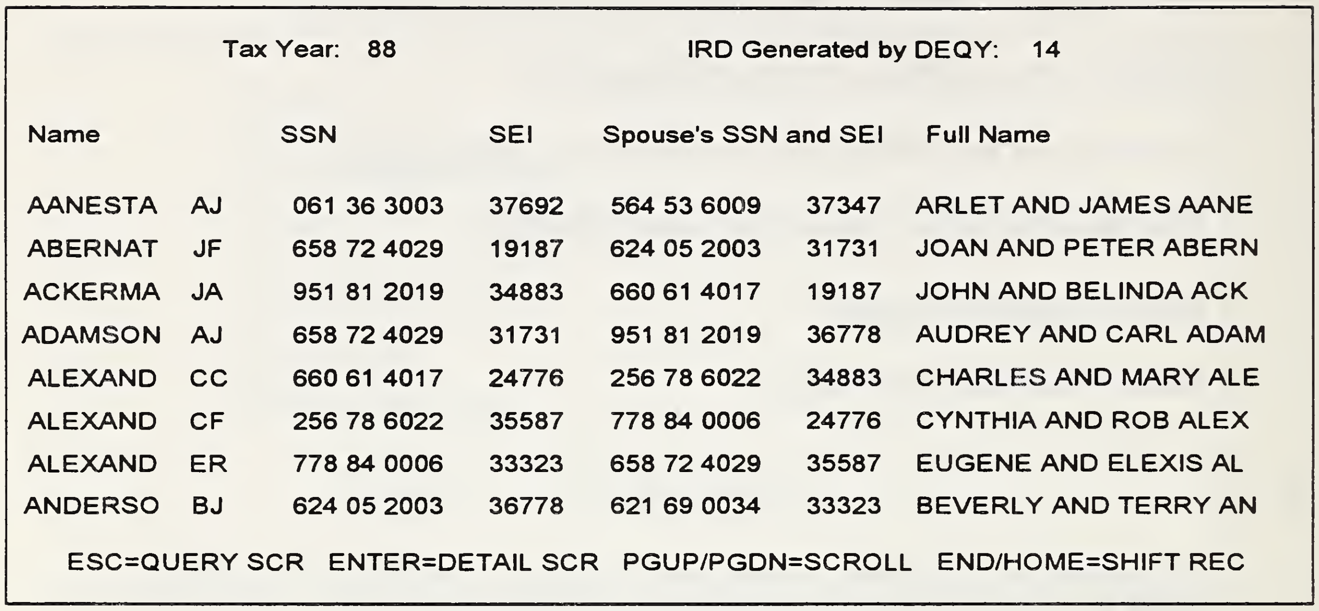
In Order to ...

- Move between fields or within a field, use the **Tab** and **Arrow** keys.
- Delete a wrong character, use the **Backspace** key.
- View the year and IRD from the previous query, press **Pg Up**.
- Print the information on the screen, press **Print Scrn**.
- Exit the program, press **Esc**.
- Retrieve matches, press **Return**.

If ...

- No records match the search parameters, a message is displayed. Press **Esc** to return to the query screen.
- More records match the search parameters than can be reasonably retrieved, a message is displayed asking to refine the search. Press **Esc** to return to the query screen.
- A reasonable number of matches are found, the records are displayed on the screen for the user to browse.

A.3 The Browse Screen



```
              Tax Year:   88                           IRD Generated by DEQY:   14

 Name              SSN            SEI      Spouse's SSN and SEI     Full Name

 AANESTA   AJ      061 36 3003    37692    564 53 6009    37347     ARLET AND JAMES AANE
 ABERNAT   JF      658 72 4029    19187    624 05 2003    31731     JOAN AND PETER ABERN
 ACKERMA   JA      951 81 2019    34883    660 61 4017    19187     JOHN AND BELINDA ACK
 ADAMSON   AJ      658 72 4029    31731    951 81 2019    36778     AUDREY AND CARL ADAM
 ALEXAND   CC      660 61 4017    24776    256 78 6022    34883     CHARLES AND MARY ALE
 ALEXAND   CF      256 78 6022    35587    778 84 0006    24776     CYNTHIA AND ROB ALEX
 ALEXAND   ER      778 84 0006    33323    658 72 4029    35587     EUGENE AND ELEXIS AL
 ANDERSO   BJ      624 05 2003    36778    621 69 0034    33323     BEVERLY AND TERRY AN

    ESC=QUERY SCR   ENTER=DETAIL SCR   PGUP/PGDN=SCROLL   END/HOME=SHIFT REC
```

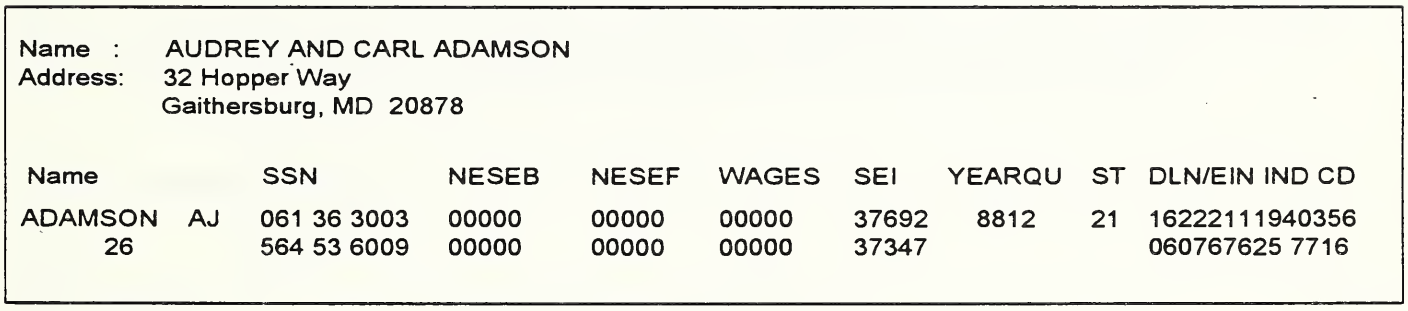
Figure A3. The browse screen

Browse Fields (Header):	Tax Year
	IRD generated by the DEQY system
Browse Fields (Record):	Last name
	Initials
	Social Security Number
	Self Employed Income
	Spouse's Social Security Number
	Spouse's Self Employed Income

In Order to ...

- Move between records within a page, use the **Arrow** keys.
- Move between pages of records, use the **Pg Up** and **Pg Dn** keys.
- View a portion of the record that is off the screen, press **End**.
- Return a record after viewing an off-screen portion, press **Home**.
- Print the information on the screen, press **Print Scrn**.
- Return to the Query screen, press **Esc**.
- View a record in detail, position the cursor to it and press **Return**.

A.4 The Detail Screen



```
Name  :   AUDREY AND CARL ADAMSON
Address:  32 Hopper Way
          Gaithersburg, MD  20878

 Name            SSN          NESEB    NESEF    WAGES    SEI     YEARQU  ST  DLN/EIN IND CD

ADAMSON   AJ    061 36 3003   00000    00000    00000    37692    8812   21  16222111940356
      26        564 53 6009   00000    00000    00000    37347               060767625 7716
```

Figure A4. The detail screen

Detail Fields (Header) :	Last name as Reported on 1040 form Street Address City, State and Zip Code
Detail Fields (Record) :	All fields as displayed on the microfilm, except the fields in the header

In Order to ...

- Print the information on the screen, press **Enter** or **Print Scrn**.

- Return to the Browse screen, press **Esc**.

Appendix B

Rappôrt Setup

B.1 Terminal Parameters

USING AN RS-232 CONNECTION

BAUD	:	19200
DATA BITS	:	7
STOP BITS	:	1
PARITY	:	EVEN
HANG UP	:	NO

After entering Rapport, the login prompt is displayed immediately. If it does not appear, press the **Enter** key. To exit, return to the login prompt and press **Alt-Spacebar** two times.

USING A MODEM LINE

BAUD	:	19200
DATA BITS	:	7
STOP BITS	:	1
PARITY	:	EVEN
HANG UP	:	NO

After entering Rapport, type: **ATZ** The screen displays OK. Then type: **ATDP (Phone #)** for pulse lines, or **ATDT (Phone #)** for tone lines. The modem dials the number, connects to the system, and the login prompt is displayed. To exit Rapport, return to the login prompt and type: **CONTROL-D** The computer responds with NO CARRIER. At this point, type: **ATH** to hang up, and the screen displays OK. Press **Alt-Spacebar** twice to exit Rapport.

B.2 File Transfer Parameters

PORT	:	COM 1 or COM 2
MAX. BAUD RATE	:	19200
BLOCK SIZE	:	1024
TTY BUFFER SIZE	:	512
RETRIES/BLOCK	:	4
TIMEOUT	:	5

B.3 Screen Colors

Normal	:	Blue background with yellow letters
Bold	:	Red background with yellow letters

Appendix C

System Error Messages

C.1 User Interface Error Messages

Too much data required for query -- Please refine query

The query entered by the user requires more than 20 files to be copied from the library, which could impact system performance due to the amount of time needed to copy and decompress that amount of files. The user must specify more letters of the last name, or enter initials upon which to search.

Error! Must enter tax year, IRD and 4 letters of last name

It is mandatory that the user enter the tax year, the IRD and at least four letters of the last name in order to request records. The user must return and provide any missing information. If the user enters less than four letters of the last name but ends the last name field with a period, this error message should not appear.

Error! Tax year must be 88 or 89

The data stored on the pilot system is only for the years of 1988 and 1989. If the user enters another year, this error message is displayed.

Error! Ird XX is invalid

The pilot system has data only for the IRD's that were found on the tapes of data sent to NIST. If the user specifies an IRD for which there is no data, this error message is displayed.

Invalid User -- Please go to an authorized terminal

In order to maintain management information and to keep each user's files separate, each user is assigned a user number based upon the terminal at which they are working. The terminal port is parsed into a user number, and if the resulting number is not in the range of the acceptable users, this error message is displayed.

Error opening database
Error preparing the query id
Error declaring the scroll cursor
Error opening the query

These error messages are displayed when the Informix ESQL/C database engine encounters problems with the query. Consult the ESQL/C manual for more documentation.

Cannot communicate with the Handler -- Please alert Computer room

The user interface requests cartridges from the optical disk library by sending a message to the cartridge handler device driver. If an error is returned by a messaging function, this error is displayed. This error could be cause by the handler program not being active on the file server.

C.2 Cartridge Handler Device Driver Error Messages

Error getting queue

This error message is displayed if the cartridge handler device driver could not allocate a message queue to allow communication between the user interface and the cartridge handler device driver.

Error sending acknowledge message to user

This error message is displayed if the cartridge handler device driver receives an error when trying to acknowledge to the user interface that the cartridge requested is now mounted and available.

Jukebox error occurred while moving the disk

This error message is displayed if the ioctl() call to the library cartridge handler returns an error.

Appendix D

Listing of the Code

```
/*
 *
 * Header file for Device Handler for
 * Jukebox
 *
 * util.h
 * by Natalie Willman
 *
 * April 2, 1991
 *
 */


/* Define Statements */
#define TRUE      1       /* Logical True                                    */
#define FALSE     0       /* Logical False                                   */
#define MAXDRIVES 2       /* Maximum number of drives that are controlled */
#define MAXDISKS  10      /* */
#define KEY1      1234L   /* Key for aquiring the service message queue   */
#define KEY2      5678L   /* Key for aquiring the wait message queue      */
#define KEY3      2345L   /* Key for aquiring acknowledge message queue   */
#define PERMS     0666    /* Permissions for queue - read/write to all    */
#define MSGLEN    14      /* Length of service message                     */
#define ACKLEN    10      /* Length of acknowledge message                 */
#define WAIT      0       /* Wait for the next message of MTYPE            */
#define NEXTMSG   0       /* Get the next message, regardless of MTYPE    */
#define MOUNT     100L    /* Message type to request mounting of a disk   */
#define UNMOUNT   200L    /* Message type to request unmounting of a disk */
#define ACK       300L    /* Message type to send an ack to the user      */


/* Structure to hold information passed to and returned from */
/* an ioctl() call to the sdlp device driver (pass thru mode */
struct IOCTL
   {
   unsigned int CMD;
   unsigned int UNIT;
   union
     {
     paddr_t ADDR;
     unsigned long DEVSIZ;
     } I1;
   union
     {
     unsigned int COUNT;
     unsigned int BLKSIZ;
     unsigned int TRANSPORT;
     } I2;
   union
     {
     unsigned long START;
     unsigned char DEVTYP;
     unsigned long CDBADDR;
     } I3;
   union
     {
     unsigned int DTSTAT;
     unsigned int IODIRECT;
     unsigned int DEST1;
     unsigned int NUMELEMENTS;
     unsigned int VALUE1;
     } I4;
   union
     {
      unsigned char GRPSIZ;
      unsigned int DEST2;
      unsigned int ELEMENTYPE;
      unsigned int VALUE2;
      } I5;
    union
      {
      unsigned char VARBLK;
      unsigned char INVT;
      } I6;
    unsigned int TYPE;
    unsigned int STATUS;
    };


/* Structure containing information on the drives in the system */
struct DRINFO
 {
 int jukeaddress;
 int driveaddress;
 char devname[15];
 char mountdir[10];
 int disk;
 int side;
 int busy;
 };


/* Template for message passing of service request messages   */
struct MSGBUF
  {
  long mtype;
  int  user;
  int  juke;
  int  disk;
  int side;
  };


/* Template for message passing of acknowledge messages       */
struct MSGACK
  {
  long mtype;
  char mtdir[10];
  };
```

```
/*
 *
 * Proto.h
 *
 * Header file for user interface,
 * easear.ec, contains #define statements,
 * and struture definitions
 *
 */



/* Define Statements */
#define ESC        27
#define ENTER      10
#define BACKSPACE  8
#define UPARROW    3
#define DOWNARROW  2
#define LEFTARROW  4
#define RIGHTARROW 5
#define TAB        9
#define PAGEDOWN   82
#define PAGEUP     83
#define END        104
#define HOME       6
#define ROWTAX     5
#define ROWIRD     7
#define ROWLAST    9
#define ROWINI     9
#define COLINI     57
#define ROWSSN     11
#define ROWSP      13
#define ROWSEI     15
#define MINCOL     36
#define MAXCOL     70
#define FIRSTROW   6
#define LASTROW    20
#define MINVIEW    5
#define MAXVIEW    77
#define MATCHSCR   8
#define GENERALWIN 0
#define DETAILWIN  1
#define ERRORMSG   2
#define KEY1       1234L
#define KEY3       2345L
#define MSGLEN     14
#define ACKLEN     10
#define WAIT       0
#define MOUNT      100L
#define UNMOUNT    200L
#define ACK        300L


/* Global pointers to windows and panels */
WINDOW *general, *detail, *errormsg;
PANEL *pgeneral, *pdetail, *perrormsg;


struct QUERY
   {
   char lastname[8];
   char initials[3];
   char ssno[10];
   char sei[6];
   char year[3];
```

```
    char ird[3];
    char fullname[50];
    char sp_ssno[10];
    char sp_sei[6];
    };

struct REC
    {
    char nesef[6];
    char neseb[6];
    char wages[6];
    char month[3];
    char se_quar[2];
    char option_cd[2];
    char dln[15];
    char iwp[3];
    char err_code[3];
    char sp_neseb[6];
    char sp_nesef[6];
    char sp_wages[6];
    char date[7];
    char ein[10];
    char ind_cd[5];
    char st_add[35];
    char city_st[22];
    char zip[6];
    };

struct INFO
   {
   char year[3];
   char ird[3];
   char lastname[8];
   char initials[3];
   char ssno[10];
   char sp_ssno[10];
   char sei[6];
   };

struct MSGBUF
   {
   long mtype;
   int user;
   int juke;
   int disk;
   int side;
   };

struct MSGACK
   {
   long mtype;
   char mtdir[10];
   };
```

```
/*
 * check.c
 *
 * by Natalie Willman
 *
 * This program opens a series of files as specified by the user, and
 * prints to a file the IRD's found in each file.
 */

/* Include files */
#include <stdio.h>
#include <string.h>

/* Define statements */
#define TRUE  1
#define FALSE 0


/*
 *
 *  MAIN PROGRAM
 *
 */

main()
  {
  int i, count;
  char filename[30], tape[4];


  /* Retrieve Information from the user on the files to analyze */
  printf("Enter the Total number of files on the disk:  ");
  scanf("%d%*c", &count);
  printf("\n\n");
  printf("Enter the Tape number from which this data originated:  ");
  scanf("%s%*c", tape);
  printf("\n\n");

  /* Initialize the filename array */
  sprintf(filename, "/drive0/0");

  /* Analyze each of the files */
  for(i = 0; i < count; i++)
    {
    printf("Analyzing File %s ...\n", filename);
    check_file(filename, tape);

    /* Create the next file name */
    sprintf(filename, "%s%d", "/drive0/", i+1);
    }
  }




 /*
  * Read in the records and check to see if the ird is different
  * from the ird of the last record read.  If so, the ird and
  * the filename are printed out to a file.
  *
  * Input:  Filename from which to read the data
  *         tape number of the tape from which the data originated
  *
 * Output: A file containing any changes in IRD
 *
 */

check_file(filename, tape)
  char filename[], tape[];
  {
  int i;
  char record[235];
  char ch, ird[3];
  char prevird[3], output[20];
  FILE *dat, *temp, *fopen();
  int rec, result, totalrec;


  /* Open the tape data file from which to read records */
  dat = fopen(filename, "r");

  /* Set up record counters and initialize the previous IRD variable */
  rec = 0;
  totalrec = 0;
  prevird[0] = '0';
  prevird[1] = '0';
  prevird[2] = 0;


  /* Read in each of the records in the file                        */
  do
     {
     /* Read in the record - 234 characters long */
     for(i = 0; i < 234; i++)
       {
       record[i] = fgetc(dat);
       ch = record[i];
       /* if EOF is returned at this point, it is most likely an error */
       /* sop print a notice to the error file                         */
       if(record[i] == EOF)
          {
          sprintf(output, "%s%s", "err.", tape);
          temp = fopen(output, "a");
          fprintf(output, "Error record %d, file %s\n", totalrec+1,filename);
          fclose(output);
          }

       /* Copy the IRD to a variable */
       if(i == 44)
          ird[0] = ch;
       if(i == 45)
          {
          ird[1] = ch;
          ird[2] = 0;
          }
       }

     /* set EOL character */
     record[234] = 0;

     /* Increment the total record count for this file */
     totalrec++;


     /* compare the ird just read to the previous ird */
     result = strcmp(ird, prevird);

     /* if the irds are different, print ird and filename to a file */
     if( (result != 0) && (ird[0] != EOF) )
```

```
        {
        sprintf(output, "%s%s", "irdfile.", tape);
        temp = fopen(output, "a");
        fprintf(temp, "%s %s\n", ird, filename);
        fclose(temp);
        }

    /* copy the current ird to the previous ird */
    prevird[0] = ird[0];
    prevird[1] = ird[1];
    prevird[2] = 0;

    /* skip over the EOB character if at the end of a block */
    if(rec == 135)
       {
       ch = fgetc(dat);
       rec = 0;
       }
    else
      rec++;
    }
  while(ch != EOF);

  /* close the data file */
  fclose(dat);
  }
```

```
/*
 * Compress.c
 *
 * This program copies a series of files from the source disk
 * to the destination disk and compresses them
 *
 */

#include <stdio.h>

main()
  {
  char command[200], filename[60], ird[5];
  char sourcefile[100], destfile[100];
  char ch;

  /* Read in the first filename */
  ch = scanf("%s%*c", filename);

  /* While there is still a remaining list */
  while(ch != EOF)
    {
    /* Create source and destination file names */
    strcpy(sourcefile, "/drive2/");
    strcat(sourcefile, filename);
    strcpy(destfile, "/drive1/");
    strcat(destfile, filename);

    /* Copy the file from the source disk to the destination disk */
    printf("Copying file %s to file %s\n", sourcefile, destfile);
    strcpy(command, "cp ");
    strcat(command, sourcefile);
    strcat(command, " ");
    strcat(command, destfile);
    printf("%s\n", command);
    system(command);

    /* Compress the file on the destination disk */
    strcpy(command, "pack ");
    strcat(command, destfile);
    printf("Compressing File %s\n", destfile);
    printf("%s\n", command);
    system(command);

    /* Get the next filename */
    ch = scanf("%s%*c", filename);
    }
  }
```

```
/*
 * conv.ec
 *
 * by Natalie Willman
 *
 * This program opens a list of files of easear records, and
 * converts them to Informix Database format
 *
 */

/* Define statements      */
#define TRUE  1
#define FALSE 0

/* Include Files          */
$include sqlca;
#include <stdio.h>
#include <string.h>


/* Structure Definitions */
struct QUERYINFO
   {
   char lastname[8];
   char initials[3];
   char ssno[10];
   char sei[6];
   char year[3];
   char ird[3];
   char fullname[63];
   char sp_ssno[10];
   char sp_sei[6];
   };

struct REC
   {
   char nesef[6];
   char neseb[6];
   char wages[6];
   char month[3];
   char se_quar[2];
   char dln[15];
   char iwp[3];
   char err_code[3];
   char option_cd[2];
   char sp_neseb[6];
   char sp_nesef[6];
   char sp_wages[6];
   char date_rcd[7];
   char ein[10];
   char ind_cd[5];
   char st_add[35];
   char city_st[22];
   char zip[6];
   };


 /*
  * Main Program
  *
  */

 main()
   {
   struct QUERYINFO querymatches, *query;
struct REC full_record, *record;
char filename[30], cpyname[10], command[300];
FILE *namefile, *fopen();
char ch;


printf("Make sure that the source disk is mounted as read only in\n");
printf("/drive2, and that the destination disk is mounted as\n");
printf("read/write in /drive1.  \n\n");
printf("Make sure that you have done a 'ls' of the source disk\n");
printf("and stored the file as 'names' in the current directory\n");

printf("Press any key to continue ...\n\n");
getchar();

/* Set up pointers */
record = &full_record;
query = &querymatches;

/* Open the database */
$ database junk;
if(sqlca.sqlcode)
  printf("ERROR\n");

/* Open the list of files to convert */
namefile = fopen("names", "r");

ch = fscanf(namefile, "%s%*c", cpyname);
while(ch != EOF)
 {
 strcpy(filename, "/drive2/");
 strcat(filename, cpyname);

 printf("Converting file %s ...\n", filename);

 /* Create the Database Table */
 $ create table test
   (
   lastname   char(7),
   initials   char(2),
   ssn        char(9),
   neseb      char(5),
   nesef      char(5),
   wages      char(5),
   sei        char(5),
   year       char(2),
   month      char(2),
   se_quar    char(1),
   option_cd  char(1),
   ird        char(2),
   dln        char(14),
   full_name  char(62),
   iwp        char(2),
   err_code   char(2),
   sp_ssn     char(9),
   sp_neseb   char(5),
   sp_nesef   char(5),
   sp_wages   char(5),
   sp_sei     char(5),
   date_rcd   char(6),
   ein        char(9),
   ind_cd     char(4),
   st_add     char(34),
   city_st    char(21),
   zip        char(5)
   );
```

```
    $ create index test on test (lastname);

    /* read in and convert the records */
    read_rec(query, record, filename);

    /* Copy the files to an optical disk */
    strcpy(command, "cp /usr/pilot/junk.dbs/test*.dat /drive1/");
    strcat(command, cpyname);
    strcat(command, ".dat");
    system(command);

    strcpy(command, "cp /usr/pilot/junk.dbs/test*.idx /drive1/");
    strcat(command, cpyname);
    strcat(command, ".idx");
    system(command);

    /* Drop the table -- if the table is not dropped, subsequent */
    /* records will be stored in reverse order                   */
    $ drop table test;

    /* Read in the next filename */
    ch = fscanf(namefile, "%s%*c", cpyname);
    }
   }




/*
 * Read in the records, splice them into fields, and convert
 * the data to .dbs format
 *
 * Input:  structures to hold the spliced data
 *         Filename of the file from which to read records
 *
 */

read_rec(query, record, filename)
  struct QUERYINFO *query;
  struct REC *record;
  char filename[];
  {
  int i;
  char rec[235];
  char ch;
  FILE *dat, *fopen();

  /* Open the file to read */
  dat = fopen(filename, "r");
  if(dat == NULL)
     {
     printf("ERROR opening file %s\n", filename);
     exit();
     }

  /* Read in each of the records in the file                          */
  do
    {
     /* Read in the record - 234 characters long */
     for(i = 0; i < 234; i++)
       {
       rec[i] = fgetc(dat);
       ch = rec[i];
       }
    if(ch != EOF)
      {
      /* Split the records into fields */
      splice_rec(rec, query, record);

      /* convert the records to .dbs format */
      conv_rec(query, record);
      }
    }
  while(ch != EOF);

  /* Close the file */
  fclose(dat);
  }



/*
 * Function to split the records into separate
 * fields
 *
 * Input:  The record as a string
 *         Structure to hold the spliced record
 *
 * Output: The record is split into fields and stored in the
 *         structures
 *
 */


splice_rec(rec, query, record)
  char rec[];
  struct QUERYINFO *query;
  struct REC *record;
  {
  int i;


  /* set up the EOL characters */
  query->lastname[7] = 0;
  query->initials[2] = 0;
  query->ssno[9] = 0;
  query->sei[5] = 0;
  query->year[2] = 0;
  query->ird[2] = 0;
  query->fullname[62] = 0;
  query->sp_ssno[9] = 0;
  query->sp_sei[5] = 0;
  record->nesef[5] = 0;
  record->neseb[5] = 0;
  record->wages[5] = 0;
  record->month[2] = 0;
  record->se_quar[1] = 0;
  record->dln[14] = 0;
  record->iwp[2] = 0;
  record->err_code[2] = 0;
  record->option_cd[1] = 0;
  record->sp_neseb[5] = 0;
  record->sp_nesef[5] = 0;
  record->sp_wages[5] = 0;
  record->date_rcd[6] = 0;
  record->ein[9] = 0;
  record->ind_cd[4] = 0;
  record->st_add[34] = 0;
  record->city_st[21] = 0;
  record->zip[5] = 0;
```

```
/* splice the field based on position in record string */
for(i = 0; i < 234; i++)
   {
   switch(i)
      {
      case 0: case 1: case 2: case 3: case 4: case 5: case 6:
           query->lastname[i] = rec[i];
           break;
      case 7: case 8:
           query->initials[i-7] = rec[i];
           break;
      case 9: case 10: case 11: case 12: case 13:
      case 14: case 15: case 16: case 17:
           query->ssno[i-9] = rec[i];
           break;
      case 18: case 19: case 20: case 21: case 22:
           record->neseb[i-18] = rec[i];
           break;
      case 23: case 24: case 25: case 26: case 27:
           record->nesef[i-23] = rec[i];
           break;
      case 28: case 29: case 30: case 31: case 32:
           record->wages[i-28] = rec[i];
           break;
      case 33: case 34: case 35: case 36: case 37:
           query->sei[i-33] = rec[i];
           break;
      case 38: case 39:
           query->year[i-38] = rec[i];
           break;
      case 40: case 41:
           record->month[i-40] = rec[i];
           break;
      case 42:
           record->se_quar[i-42] = rec[i];
           break;
      case 43:
           record->option_cd[i-43] = rec[i];
           break;
      case 44: case 45:
           query->ird[i-44] = rec[i];
           break;
      case 46: case 47: case 48: case 49: case 50: case 51:
      case 52: case 53: case 54: case 55: case 56: case 57:
      case 58: case 59:
           record->dln[i-46] = rec[i];
           break;
      case 60: case 61: case 62: case 63: case 64: case 65:
      case 66: case 67: case 68: case 69: case 70: case 71:
      case 72: case 73: case 74: case 75: case 76: case 77:
      case 78: case 79: case 80: case 81: case 82: case 83:
      case 84: case 85: case 86: case 87: case 88: case 89:
      case 90: case 91: case 92: case 93: case 94: case 95:
      case 96: case 97: case 98: case 99: case 100: case 101:
      case 102: case 103: case 104: case 105: case 106: case 107:
      case 108: case 109: case 110: case 111: case 112: case 113:
      case 114: case 115: case 116: case 117: case 118: case 119:
      case 120: case 121:
           query->fullname[i-60] = rec[i];
           break;
      case 122: case 123:
           record->iwp[i-122] = rec[i];
           break;
```

```
       case 124: case 125:
            record->err_code[i-124] = rec[i];
            break;
       case 126: case 127: case 128: case 129: case 130:
       case 131: case 132: case 133: case 134:
            query->sp_ssno[i-126] = rec[i];
            break;
       case 135: case 136: case 137: case 138: case 139:
            record->sp_neseb[i-135] = rec[i];
            break;
       case 140: case 141: case 142: case 143: case 144:
            record->sp_nesef[i-140] = rec[i];
            break;
       case 145: case 146: case 147: case 148: case 149:
            record->sp_wages[i-145] = rec[i];
            break;
       case 150: case 151: case 152: case 153: case 154:
            query->sp_sei[i-150] = rec[i];
            break;
       case 155: case 156: case 157: case 158: case 159: case 160:
            record->date_rcd[i-155] = rec[i];
            break;
       case 161: case 162: case 163: case 164: case 165: case 166:
       case 167: case 168: case 169:
            record->ein[i-161] = rec[i];
            break;
       case 170: case 171: case 172: case 173:
            record->ind_cd[i-170] = rec[i];
            break;
       case 174: case 175: case 176: case 177: case 178: case 179:
       case 180: case 181: case 182: case 183: case 184: case 185:
       case 186: case 187: case 188: case 189: case 190: case 191:
       case 192: case 193: case 194: case 195: case 196: case 197:
       case 198: case 199: case 200: case 201: case 202: case 203:
       case 204: case 205: case 206: case 207:
            record->st_add[i-174] = rec[i];
            break;
       case 208: case 209: case 210: case 211: case 212: case 213:
       case 214: case 215: case 216: case 217: case 218: case 219:
       case 220: case 221: case 222: case 223: case 224: case 225:
       case 226: case 227: case 228:
            record->city_st[i-208] = rec[i];
            break;
       case 229: case 230: case 231: case 232: case 233:
            record->zip[i-229] = rec[i];
            break;
       }
    }
  }


/*
 * Convert the record to .dbs format
 *
 * Input:  Structures containing the spliced record
 *
 * Output: The data is written to an Informix SQL Database Table
 *
 */

conv_rec(query, record)
  struct QUERYINFO *query;
  struct REC *record;
  {
  /* Declaration of Host Variables */
  $ char lastname[8];
 $ char initials[3];
 $ char ssno[10];
 $ char sei[6];
 $ char year[3];
 $ char ird[3];
 $ char fullname[63];
 $ char sp_ssno[10];
 $ char sp_sei[6];
 $ char nesef[6];
 $ char neseb[6];
 $ char wages[6];
 $ char month[3];
 $ char se_quar[2];
 $ char dln[15];
 $ char iwp[3];
 $ char err_code[3];
 $ char option_cd[2];
 $ char sp_neseb[6];
 $ char sp_nesef[6];
 $ char sp_wages[6];
 $ char date_rcd[7];
 $ char ein[10];
 $ char ind_cd[5];
 $ char st_add[35];
 $ char city_st[22];
 $ char zip[6];


 /* Copy the structure data to a host variable - this is due to a */
 /* bug in Informix which causes ESQL/C to have problems dealing  */
 /* with structures as host variables                             */
 stcopy(query->lastname, lastname);
 stcopy(query->initials, initials);
 stcopy(query->ssno, ssno);
 stcopy(query->sei, sei);
 stcopy(query->year, year);
 stcopy(query->ird, ird);
 stcopy(query->fullname, fullname);
 stcopy(query->sp_ssno, sp_ssno);
 stcopy(query->sp_sei, sp_sei);
 stcopy(record->nesef, nesef);
 stcopy(record->neseb, neseb);
 stcopy(record->wages, wages);
 stcopy(record->month, month);
 stcopy(record->se_quar, se_quar);
 stcopy(record->dln, dln);
 stcopy(record->iwp, iwp);
 stcopy(record->err_code, err_code);
 stcopy(record->option_cd, option_cd);
 stcopy(record->sp_neseb, sp_neseb);
 stcopy(record->sp_nesef, sp_nesef);
 stcopy(record->sp_wages, sp_wages);
 stcopy(record->date_rcd, date_rcd);
 stcopy(record->ein, ein);
 stcopy(record->ind_cd, ind_cd);
 stcopy(record->st_add, st_add);
 stcopy(record->city_st, city_st);
 stcopy(record->zip, zip);


 /* Insert record into database */
 $ INSERT INTO test (lastname, initials, ssn, neseb, nesef,
          wages, sei, year, month, se_quar, option_cd, ird,
          dln, full_name, iwp, err_code, sp_ssn, sp_neseb,
          sp_nesef, sp_wages, sp_sei, date_rcd, ein,
          ind_cd, st_add, city_st, zip) VALUES (
        $lastname, $initials,
        $ssno, $neseb,
        $nesef, $wages,
        $sei, $year,
        $month, $se_quar,
        $option_cd,
        $ird, $dln,
        $fullname, $iwp,
        $err_code,
        $sp_ssno, $sp_neseb,
        $sp_nesef, $sp_wages,
        $sp_sei, $date_rcd,
        $ein, $ind_cd,
        $st_add, $city_st,
        $zip);

/* Check to see that no error occured while writing to the database */
if(sqlca.sqlcode)
     {
     printf("ERROR occured! - Number %d\n", sqlca.sqlcode);
     exit();
     }
}
```

```
/*
 *
 * Program devhandler.c
 * by Natalie Willman
 *
 * April 2, 1991
 *
 * This is the device driver for the HP jukeboxes which
 * controls the logic of the queues
 *
 *
 */




/* Include Files */
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "util.h"

main()
 {
 int i, j;       /* Counters                                    */
 int res;
 int queueid;    /* queue id for messages to service            */
 int waitid;     /* queue id for messages waiting on a disk     */
 int ackid;      /* queue id for acknowledgement messages       */
 int freedrive;  /* Drive number of the drive to mount to       */
 int unmtdrive;  /* Drive number of the drive to unmount        */
 int empty;      /* boolean variable to indicate waiting msgs   */
 struct DRINFO drives[MAXDRIVES];  /* Array with drive info     */
 struct MSGBUF message, *pmessage; /* pointer to service msgs   */
 struct MSGACK msgack, *pmsgack;   /* pointer to ack messages   */

 /* assign the message structures to the pointer */
 pmessage = &message;
 pmsgack = &msgack;

 /* Initialize the drive information structure array */
 drives[0].jukeaddress = 0;
 drives[0].driveaddress = 1;
 strcpy(drives[0].devname, "/dev/sstb0");
 strcpy(drives[0].mountdir, "/drive0");
 drives[0].busy = FALSE;

 drives[1].jukeaddress = 0;
 drives[1].driveaddress = 2;
 strcpy(drives[1].devname, "/dev/sstb1");
 strcpy(drives[1].mountdir, "/drive1");
 drives[1].busy = FALSE;


 /* setup the message queues */
 queueid = msgget(KEY1, PERMS | IPC_CREAT);
 waitid = msgget(KEY2, PERMS | IPC_CREAT);
 ackid = msgget(KEY3, PERMS | IPC_CREAT);
```

```
/* if any of the queues cannot be created, error and exit */
if( (queueid < 0) || (waitid < 0) || (ackid < 0) )
   {
   printf("error getting the queue\n");
   exit(0);
   }


do
 {
 /* Check to see if both drives are currently busy - if so wait for */
 /* an unmount message                                              */
 if( (drives[0].busy == TRUE) && (drives[1].busy == TRUE) )
    {
    msgrcv(queueid, pmessage, MSGLEN, UNMOUNT, WAIT);
    print_msg(pmessage);
    unmtdrive = determine_drive(drives, pmessage);
    empty = check_wait_queue(pmessage, waitid);
    if(empty)
      {
      printf("Nothing in wait queue - returning disk\n");
      mark_idle(unmtdrive, drives);
      unmt_disk(drives, unmtdrive);
      ret_disk(pmessage, drives, unmtdrive);
      }
    else
      {
      print_msg(pmessage);
      if(pmessage->side == drives[unmtdrive].side)
        {
        printf("Same side retrieved from queue\n");
        strcpy(pmsgack->mtdir, drives[unmtdrive].mountdir);
        pmsgack->mtype = ACK + pmessage->user;
        res = msgsnd(ackid, pmsgack, ACKLEN, IPC_NOWAIT);
        if(res < 0)
          printf("ERROR sending acknowledge message to user\n%c", '\007');
        }
      else
        {
        printf("Opposite side retrieved from queue - flipping disk\n");
        unmt_disk(drives, unmtdrive);
        mark_busy(drives, pmessage, unmtdrive);
        flip_disk(drives, unmtdrive);
        /* Delay so that the mount will not fail due to disk not in drive */
        for(i = 0; i < 30000; i++)
          for(j = 0; j < 30; j++);
        mount_disk(drives, unmtdrive);
        strcpy(pmsgack->mtdir, drives[unmtdrive].mountdir);
        pmsgack->mtype = ACK + pmessage->user;
        res = msgsnd(ackid, pmsgack, ACKLEN, IPC_NOWAIT);
        if(res < 0)
          printf("ERROR sending acknowledge message to user\n%c", '\007');
        }
      }
    }
 else  /* if both drives are not busy */
    {
    msgrcv(queueid, pmessage, MSGLEN, NEXTMSG, WAIT);
    print_msg(pmessage);
    if(pmessage->mtype == UNMOUNT)
       {
       unmtdrive = determine_drive(drives, pmessage);
       empty = check_wait_queue(pmessage, waitid);
       if(empty)
         {
```

```
        printf("Nothing in wait queue - returning disk\n");
        mark_idle(unmtdrive, drives);
        unmt_disk(drives, unmtdrive);
        ret_disk(pmessage, drives, unmtdrive);
        }
      else
        {
        print_msg(pmessage);
        if(pmessage->side == drives[unmtdrive].side)
          {
          printf("Same side retrieved from queue\n");
          strcpy(pmsgack->mtdir, drives[unmtdrive].mountdir);
          pmsgack->mtype = ACK + pmessage->user;
          res = msgsnd(ackid,pmsgack,ACKLEN,IPC_NOWAIT);
          if(res < 0)
            printf("ERROR sending acknowledge message to user\n%c", '\007');
          }
        else
          {
          printf("Opposite side retrieved from queue - flipping disk\n");
          unmt_disk(drives, unmtdrive);
          mark_busy(drives, pmessage, unmtdrive);
          flip_disk(drives, unmtdrive);
          /* Delay so that the mount will not fail due to disk not in drive */
          for(i = 0; i < 30000; i++)
            for(j = 0; j < 30; j++);
          mount_disk(drives, unmtdrive);
          strcpy(pmsgack->mtdir, drives[unmtdrive].mountdir);
          pmsgack->mtype = ACK + pmessage->user;
          res = msgsnd(ackid,pmsgack,ACKLEN,IPC_NOWAIT);
          if(res < 0)
            printf("ERROR sending acknowledge message to user\n%c", '\007');
          }
        }
      }
    else if(((pmessage->disk == drives[0].disk) && (drives[0].busy == TRUE)) ||
         ((pmessage->disk == drives[1].disk) && (drives[1].busy == TRUE)))
      {
      printf("request placed in wait queue - disk not available\n");
      pmessage->mtype = MOUNT+pmessage->disk;
      res = msgsnd(waitid,pmessage,MSGLEN,IPC_NOWAIT);
      if(res < 0)
        printf("ERROR sending acknowledge message to user\n%c", '\007');
      }
    else
      {
      printf("A free drive is available, and the disk is available\n");
      freedrive = get_drive(drives);
      mark_busy(drives, pmessage, freedrive);
      mov_disk(pmessage, drives, freedrive);
      /* Delay so that the mount will not fail due to disk not in drive */
      for(i = 0; i < 30000; i++)
        for(j = 0; j < 30; j++);
      mount_disk(drives, freedrive);
      strcpy(pmsgack->mtdir, drives[freedrive].mountdir);
      pmsgack->mtype = ACK+pmessage->user;
      res = msgsnd(ackid, pmsgack, ACKLEN, IPC_NOWAIT);
      if(res < 0)
        printf("ERROR sending acknowledge message to user\n%c", '\007');
      }
    }
  }
while(TRUE);   /* INFINITE LOOP */
}
```

```
/*
 *
 * Function determine_drive()
 *
 * This function determines which drive contains
 * the disk referenced in the message pointed to by
 * the passed pointer
 *
 * Input:  Array containing drive information
 *         Pointer to the message
 *
 * Output: The drive number containing the requested disk
 *
 */

determine_drive(drives, pmessage)
  struct MSGBUF *pmessage;
  struct DRINFO drives[];
  {
  int unmtdrive;

  if(drives[0].disk == pmessage->disk)
     unmtdrive = 0;
  else
     unmtdrive = 1;

  printf("Removing disk from drive %d\n", unmtdrive+1);
  return(unmtdrive);
  }




/*
 *
 * Function get_drive()
 *
 * This function determines which drive is currently free
 *
 * Input:  Array containing drive information
 *
 * Output: The drive number that is free
 *
 */

get_drive(drives)
  struct DRINFO drives[];
  {
  int freedrive;

  if(drives[0].busy == FALSE)
      freedrive = 0;
   else
      freedrive = 1;

   printf("Placing disk in %d\n", freedrive+1);
   return(freedrive);
   }
```

```
/*
 *
 * Function mark_busy()
 *
 * This function marks the selected drive as busy
 *
 * Input:  Array containing drive information
 *         Pointer to message
 *         Drive number to mark as busy
 *
 * Output: The drive array is modified so that the drive
 *         is busy, and the disk number and side in the
 *         drive are stored in the array
 *
 */

mark_busy(drives, pmessage, freedrive)
  struct DRINFO drives[];
  struct MSGBUF *pmessage;
  int freedrive;
  {
  drives[freedrive].busy = TRUE;
  drives[freedrive].disk = pmessage->disk;
  drives[freedrive].side = pmessage->side;

  printf("Drive %d marked as busy and having disk %d side %d\n",
     drives[freedrive].driveaddress, drives[freedrive].disk,
     drives[freedrive].side);
  }




/*
 *
 * Function mark_idle()
 *
 * This function marks the selected drive as idle
 *
 * Input:  Array containing drive information
 *         Drive number to mark as idle
 *
 * Output: The drive array is modified so that the drive
 *         is marked as idle
 *
 */

mark_idle(unmtdrive, drives)
  int unmtdrive;
  struct DRINFO drives[];
  {
  drives[unmtdrive].busy = FALSE;
  printf("Drive %d marked as idle\n", drives[unmtdrive].driveaddress);
  }



/*
 *
 * Function flip_disk()
 *
 * This function sets the source and destination address of the
 * disk, as well as indicating the address of the changer device,
 * and passes it to a function to make the CDB and to execute
 * the SCSI instruction to flip the disk and replace it in the drive
 *
 * Input:  File descriptor for the sdlp device driver
 *         Pointer to the message
 *         Array containing drive information
 *         Drive the disk is to be placed in
 *
 * Output: The correct disk is flipped and replaced in the drive
 */

flip_disk(drives, freedrive)
  int freedrive;
  struct DRINFO drives[];
  {
  int srcint, destint, num;
  unsigned int movemed(), cdbsize, buffsize;
  char cdb[12], source, dest, invert;

  srcint = drives[freedrive].driveaddress;
  destint = drives[freedrive].driveaddress;
  source = srcint;      /* Set Source Address - Will drop the MSB      */
  dest = destint;       /* Set Destination Address - Will drop the MSB */
  num = drives[freedrive].jukeaddress; /* Set jukebox SCSI ID */

  if(drives[freedrive].side == 1)
    invert = TRUE;
  else
    invert = FALSE;

  printf("Source set at %d, Destination set at %d\n", srcint, destint);
  printf("SCSI address set at %d, flip set to %d\n", num, invert);


  buffsize = movemed(cdb, &cdbsize, source, dest, invert);
  execute(cdb, cdbsize, buffsize, num);
  }




/*
 *
 * Function mov_disk()
 *
 * This function sets the source and destination address of the
 * disk, as well as indicating the address of the changer device,
 * and passes it to a function to make the CDB and to execute
 * the SCSI instruction to move the disk to a drive
 *
 * Input:  File descriptor for the sdlp device driver
 *         Pointer to the message
 *         Array containing drive information
 *         Drive the disk is to be placed in
 *
 * Output: The correct disk is inserted into the drive
 */

mov_disk(pmessage, drives, freedrive)
  int freedrive;
  struct MSGBUF *pmessage;
  struct DRINFO drives[];
  {
  int srcint, destint, num;
  unsigned int movemed(), cdbsize, buffsize;
  char cdb[12], source, dest, invert;

  srcint = pmessage->disk + 10;
  destint = drives[freedrive].driveaddress;
  source = srcint;      /* Set Source Address - Will drop the MSB      */
  dest = destint;       /* Set Destination Address - Will drop the MSB */
  num = drives[freedrive].jukeaddress; /* Set jukebox SCSI ID */

  if(drives[freedrive].side == 1)
    invert = TRUE;
  else
    invert = FALSE;


  printf("Source set at %d, Destination set at %d\n", srcint, destint);
  printf("SCSI address set at %d, flip set to %d\n", num, invert);

  buffsize = movemed(cdb, &cdbsize, source, dest, invert);
  execute(cdb, cdbsize, buffsize, num);
  }



/*
 *
 * Function mount_disk()
 *
 * This function issues a mount command to mount the file system
 * of the disk in the current drive
 *
 * Input:  Array containing drive information
 *         Drive number of the drive containing the disk
 *
 * Output: The disk is mounted and available to the user
 *
 */

mount_disk(drives, freedrive)
  struct DRINFO drives[];
  int freedrive;
  {
  printf("Drive %s mounted to %s\n", drives[freedrive].devname,
         drives[freedrive].mountdir);
  mount(drives[freedrive].devname, drives[freedrive].mountdir, 1);
  }






/*
 *
 * Function ret_disk()
 *
 * This function sets the source and destination address of the
 * disk, as well as indicating the address of the changer device,
 * and passes it to a function to make the CDB and to execute
 * the SCSI instruction to return the disk to a storage slot
 *
 * Input:  File descriptor for the sdlp device driver
 *         Pointer to the message
 *         Array containing drive information
 *         Drive the disk is to be removed from
 *
 * Output: The disk is removed from the drive and placed back in
 *         its correct storage slot
 *
 */

ret_disk(pmessage, drives, unmtdrive)
  int unmtdrive;
  struct MSGBUF *pmessage;
  struct DRINFO drives[];
  {
  int srcint, destint, num;
  unsigned int movemed(), cdbsize, buffsize;
  char cdb[12], source, dest, invert;


  srcint = drives[unmtdrive].driveaddress;
  destint = pmessage->disk + 10;
  source = srcint;      /* Set Source Address - Will drop the MSB      */
  dest = destint;       /* Set Destination Address - Will drop the MSB */
  num = drives[unmtdrive].jukeaddress;  /* set jukebox SCSI ID */

  if(drives[unmtdrive].side == 1)
    invert = TRUE;
  else
    invert = FALSE;


  printf("Source set at %d, Destination set at %d\n", srcint, destint);
  printf("SCSI address set at %d, flip set to %d\n", num, invert);

  buffsize = movemed(cdb, &cdbsize, source, dest, invert);
  execute(cdb, cdbsize, buffsize, num);
  }





/*
 *
 * Function unmt_disk()
 *
 * This function issues an unmount command to unmount the file system
 * of the disk in the current drive
 *
 * Input:  Array containing drive information
 *         Drive number of the drive containing the disk
 *
 * Output: The disk is unmounted
 *
 */

unmt_disk(drives, unmtdrive)
  struct DRINFO drives[];
  int unmtdrive;
  {
  printf("drive %s unmounted\n", drives[unmtdrive].devname);
  umount(drives[unmtdrive].devname);
  }
```

```
/*
 * Function check_wait_queue()
 *
 * This function tries to retrieve a message from the wait queue.
 *
 * Input:  Array containing drive information
 *         Pointer to structure to hold a new message, if available
 *         Queue Id of the wait queue
 *
 * Output: The function returns TRUE if a message is found, and the
 *         pointer points to the new message.  Otherwise, FALSE is
 *         returned
 *
 */

check_wait_queue(pmessage, waitid)
  struct MSGBUF *pmessage;
  int waitid;
  {
  int msg, empty;

  msg = msgrcv(waitid, pmessage, MSGLEN, MOUNT+pmessage->disk, IPC_NOWAIT);

  if(msg < 0)
    empty = TRUE;
  else
    empty = FALSE;

  return(empty);
  }


/*
 *
 * Function print_msg()
 *
 * This function prints the contents of the current message
 * for diagnostic purposes
 *
 * Input: pointer to the message structure
 *
 * Output: contents of the message is printed out on the console screen
 *
 */

print_msg(pmessage)
  struct MSGBUF *pmessage;
  {
  printf("Message Received from User:\n");
  printf("Message type:  %d\n", pmessage->mtype);
  printf("User         :  %d\n", pmessage->user);
  printf("Jukebox #    :  %d\n", pmessage->juke);
  printf("Disk #       :  %d\n", pmessage->disk);
  printf("Side         :  %d\n", pmessage->side);
  printf("\n\n");
  }


/*
 * Sets the CDB for a Move Media Command
 *
 * Input:  array to hold the CDB
 *         CDB size
 *         source Address
 *         destination Address
 *         bit set for flipping the disk
 *
 * Output: CDB holds the Move Media command
 *         size of return buffer
 *
 */

unsigned int movemed(cdb, cdbsize, source, dest, invert)
  char cdb[];
  unsigned int *cdbsize;
  char source, dest, invert;
  {
  cdb[0] = 0xA5;     /* Command # 1A Hex                        */
  cdb[1] = 0x0;      /* Bits 5, 6, 7 are LUN                    */
  cdb[2] = 0x0;      /* Transport Element Address (MSB)         */
  cdb[3] = 0x0;      /* Transport Element Address (LSB)         */
  cdb[4] = 0x0;      /* Source Element Address (MSB)            */
  cdb[5] = source;   /* Source Element Address (LSB)            */
  cdb[6] = 0x0;      /* Destination Element Address (MSB) */
  cdb[7] = dest;     /* Destination Element Address (LSB) */
  cdb[8] = 0x0;      /* Reserved                                */
  cdb[9] = 0x0;      /* Reserved                                */
  cdb[10] = invert; /* Reserved - Bit 0 = 1 will flip          */
  cdb[11] = 0x0;     /* Control Byte                            */
  *cdbsize = 12;

  return(0);
  }




/*
 * Execute a SCSI command
 *
 * Input:  File descriptor to the sdlp device driver
 *         Array containing the CDB
 *         CDB size
 *         Buffer size of the return buffer
 *         SCSI id of the device to send the command to
 *
 * Output: SCSI command is executed, or error message is returned
 *
 */

execute(cdb, cdbsize, buffsize, num)
  char cdb[];
  unsigned int cdbsize, buffsize;
  int num;
  {
  struct IOCTL io;
  char buffer[512];
  int sdlp;

  io.CMD = 0xff;                               /* SCSI pass-thru command    */
  io.UNIT = num;                               /* SDLP Unit number          */
  io.I1.ADDR = (unsigned long) (&buffer[0]); /* Address of return buffer  */
  io.I2.COUNT = buffsize;                      /* length of return buffer   */
  io.I3.CDBADDR = (unsigned long) (&cdb[0]); /* Address of CDB            */
  io.I4.IODIRECT = 0;                          /* I/O Direction (Read/Write */
  io.I5.GRPSIZ = cdbsize;                      /* length of CDB             */
  io.TYPE = 0;                                 /* Type = 0                  */
  io.STATUS = 0;                               /* Status of No errors       */

  /* Open the Device Driver for the SDLP controller */
sdlp = open("/dev/sdlp0", O_RDWR);
if(sdlp < 0)
  {
  printf("Error opening SDLP driver for Jukebox!\n");
  exit(0);
  }

/* Execute CDB and check for errors */
if(ioctl(sdlp, 0xff, &io) < 0)
  {
  printf("Jukebox error occured while moving the disk\n");
  exit(0);
  }

/* close the device driver */
close(sdlp);
}
```

```
#
# Downld
#
# by Natalie Willman
#
# This shell script will read sets of 50 blocks of
# data from the tape, convert the data to ASCII, and
# store it in smaller files on an optical disk
# mounted to /drive0


# Print the number of sets of 50 blocks entered by the user
echo $1

# Initialize the counter and other variables
counter=0
full=$1

# While there are still sets left, print the set number, and
# download the set, converting it to ASCII, and increment the counter
while test $counter -lt $full
do
 echo $counter
 dd if=/dev/seqN0 of=/drive0/$counter ibs=31824 cbs=31824 conv=ascii count=50
 counter=`expr $counter + 1`
done

# Print the last set number (a partial set), and download and convert
# the records in that set
echo $counter
dd if=/dev/seqN0 of=/drive0/$counter ibs=31824 cbs=31824 conv=ascii
```

```
/*
 *
 * Easear.ec
 *
 * by Natalie Willman
 * April 2, 1991
 *
 * This is the user interface for the Pilot System for SSA
 *
 */


#include <curses.h>     /* Windowing functions for UNIX                  */
#include <panel.h>      /* Windowing functions for UNIX                  */
#include <string.h>     /* String manipulation functions                 */
#include <sys/types.h>  /* Used in Time Monitoring functions             */
#include <sys/ipc.h>    /* Interprocess communication structures         */
#include <sys/msg.h>    /* Messaging structures and functions            */
$include sqlca;         /* Informix ESQL/C header                        */
$include sqlda;         /* Informix ESQL/C header                        */
#include "proto.h"      /* #define statements and structure definitions */


/* Host variable to receive data from the database file - this is        */
/* done rather than making the true structures the host variables for    */
/* two reasons: to keep control of the arrays (so they are not            */
/* altered by esql, and because host variable structures cannot          */
/* be arrays -- bug in ESQL                                               */
$struct RECEIVE
  {
  char lastname[8];
  char initials[3];
  char ssno[10];
  char nesef[6];
  char neseb[6];
  char wages[6];
  char sei[6];
  char year[3];
  char month[3];
  char se_quar[2];
  char option_cd[2];
  char ird[3];
  char dln[15];
  char full_name[50];
  char iwp[3];
  char err_code[3];
  char sp_ssno[10];
  char sp_neseb[6];
  char sp_nesef[6];
  char sp_wages[6];
  char sp_sei[6];
  char date_rcvd[7];
  char ein[10];
  char ind_cd[5];
  char st_add[35];
  char city_st[22];
  char zip[6];
  } retrvar;

/* Management Information structure.                                      */
struct MANAGEINFO
  {
  long int logtime;     /* Time that the user is logged into the system   */
  long int logcount;    /* Count of logins to the system                  */
  long int qt88;        /* Time the user is entering 1988 queries            */
  long int qt89;        /* Time the user is entering 1989 queries            */

  long int qc88;        /* Count of 1988 queries                             */
  long int qc89;        /* Count of 1989 queries                             */

  long int dc88;        /* Count of 1988 Detailed information requests       */
  long int dc89;        /* Count of 1989 Detailed information requests       */

  long int pc88;        /* Count of 1988 Printout requests                   */
  long int pc89;        /* Count of 1989 Printout requests                   */

  time_t ltimer1;       /* Login timers                                      */
  time_t ltimer2;

  time_t qtimer1;       /* Query timers                                      */
  time_t qtimer2;
  };


/*
 * Main Program
 *
 */

main()
  {
  struct MANAGEINFO minfo, *pminfo;     /* management information structure  */
  struct QUERY querymatches[MATCHSCR]; /* contains browse info on matches   */
  struct REC full_record[MATCHSCR];    /* contains full info on matches     */
  struct INFO usrquery[2];             /* contains users query parameters   */
  char qinput, binput;                 /* Individual char of user input     */
  char get_query();                    /* function to get a user's query    */
  int nummatches, curmatch;            /* # of matches in memory, and       */
                                       /* array index of current match      */
  int user;                            /* User number - based on tty port   */
  int possible;                        /* Is the user's query possible      */

  /* Initialize the screen to use the ETI interface and setup windows        */
  /* Obtain the user number, and setup management information                */
  initscr();
  setup_windows();
  user = get_user_no();

  pminfo = &minfo;
  init_mi(pminfo);
  inc_log_count(pminfo);
  start_log_timer(pminfo);
  clrquery(usrquery, TRUE);

  /* Get the user's queries, until he presses ESCAPE, indicating the         */
  /* session is complete                                                     */
  do
   {
   start_query_timer(pminfo);
   qinput = get_query(usrquery);
   if(qinput != ESC)
     {
     /* Display the wait message                                             */
     disp_wait();

     /* select matches from the database                                     */
     possible = select_matches(usrquery, user);

     /* Remove the wait message                                              */
```

```
      rem_wait();

      if(possible == FALSE)
        {
        disp_err("Too much data required for query -- Please refine query");
        clrquery(usrquery, FALSE);
        }
      else
        {
        /* display screens of matches for user to browse                      */
        nummatches=display_matches(usrquery,querymatches,full_record,&curmatch);

        if(nummatches != 0)
           {
           /* allow user to page up and down through the matches              */
           binput = wgetch(general);
           while(binput != ESC)
              {
              browse(binput, &nummatches, &curmatch, usrquery, querymatches,
                 full_record, pminfo);
              binput = wgetch(general);
              }
           }
        else
           disp_err("NO MATCHES TO THIS QUERY -- PRESS ESC FOR QUERY SCR");

        /* When the user is done browsing, clear the current query            */
        inc_query_count(usrquery[0].year, pminfo);
        end_query_timer(usrquery[0].year, pminfo);
        clrquery(usrquery, FALSE);
        }
     }
   }
  while(qinput != ESC);


  /* Leave the ETI environment                                               */
  endwin();

  end_log_timer(pminfo);
  write_mi_stats(user, pminfo);
  }




/*****************************************************************************/
/*                                                                           */
/*                       Top Level Functions                                 */
/*                                                                           */
/*****************************************************************************/


/*
 * This function prompts the user for the query information
 * and stores it in the userquery structure
 *
 * Input:  pointer to structure to hold the users query (usrquery)
 * Output: The structure usrquery is filled with the search parmeters
 *         The indicator by the user whether to search for matches (ENTER)
 *               or quit the query session (ESC)
 */
```

```
char get_query(usrquery)
  struct INFO usrquery[];
  {
  int row, col;  /* Location of cursor on the screen                          */
  int done;      /* boolean variable to indicate when the query is entered */
  char input;    /* individual character of user's input                   */
  int lastlen;   /* Length of last name in the query                       */
  char errmsg[50]; /* Error message to print                               */

  /* Clear the background window */
  clrwin(GENERALWIN);

  /* Allow user to enter a query and check to make sure that the year, IRD */
  /* and at least 4 characters of the last name are entered                 */
  do
    {
     /* Prompt for the Tax Year                                             */
     disp_string(general, ROWTAX, 10, "Tax Year                    : --");

     /* Prompt for the IRD                                                  */
     disp_string(general, ROWIRD, 10, "IRD Generated by DEQY   : --");

     /* Prompt for the Last Name                                            */
     disp_string(general, ROWLAST, 10, "Last Name                  : -------");

     /* Prompt for the initials                                             */
     disp_string(general, ROWINI, 45, "  Initials: --");

     /* Prompt for the Social Security Number                               */
     disp_string(general, ROWSSN, 10, "Social Security Number  : ---------");

     /* Prompt for the Spouse's Social Security Number                      */
     disp_string(general, ROWSP, 10, "Spouse's SSN             : ---------");

     /* Prompt for the Self Employed Income                                 */
     disp_string(general, ROWSEI, 10, "Self Employed Income    : -----");

     /* Add user direction prompts                                          */
     disp_string(general, 22, 18, "ESC=EXIT   ENTER=SEARCH   PGUP=LAST QUERY");

     /* Move to the first prompt and refreshes the screen                   */
     wmove(general, ROWTAX, MINCOL);
     wrefresh(general);

     /* Obtain and interpret the User input until ESC (exit program)        */
     /* or ENTER (search on query) is given                                 */
     input = wgetch(general);
     while( (input != ESC) && (input != ENTER) )
         {
         interp(input, usrquery);
         input = wgetch(general);
         }


     /* check to see if the user entered enough information.  If so,       */
     /* the user is done entering the query.  If not, display an error     */
     /* message and loop until the user enters the correct information     */
     if( ( (strlen(usrquery[0].year) < 2) || (strlen(usrquery[0].ird) < 2) ||
       (strlen(usrquery[0].lastname) < 4) ) && (input != ESC) )
       {
       lastlen = strlen(usrquery[0].lastname);
       if(usrquery[0].lastname[lastlen-1] != '.')
         {
         disp_err("ERROR! MUST ENTER TAX YEAR, IRD, AND 4 LETTERS OF LAST NAME!");
         clrwin(GENERALWIN);
         done = FALSE;
```

```
          }
        }
     else if( (invalid_year(usrquery[0].year)) && (input != ESC) )
          {
          disp_err("ERROR! TAX YEAR MUST BE 88 OR 89");
          clrwin(GENERALWIN);
          done = FALSE;
          }
     else if( (invalid_ird(usrquery[0].ird)) && (input != ESC) )
          {
          sprintf(errmsg,"%s%s%s","ERROR! IRD ",usrquery[0].ird," IS INVALID!");
          disp_err(errmsg);
          clrwin(GENERALWIN);
          done = FALSE;
          }
     else    /* User has entered a correct query                              */
       done = TRUE;
    }
  while(done == FALSE);

  /* returns either ESC or ENTER, indicating to either exit or search      */
  return(input);
  }




/*
 * select_matches()
 *
 * Filter the database file to records matching the
 * user's query
 *
 * Input:  Users query information
 *         User number (tty port)
 *
 * Output: The database is filtered to contain records matching the query,
 *         and a cursor is set up to move through the database
 *
 */

select_matches(usrquery, user)
  struct INFO usrquery[];
  int user;
  {
  char query_last[12];   /* query for selection last name              */
  char selquery[200];    /* database query sent to INFORMIX SQL        */
  $char unionqry[4000];  /* union of databases to query                */
  char dbase[20][10];    /* names of databases to union                */
  int exact;             /* Boolean variable to indicate if search is */
                         /* exact or prefix                            */
  int sei;               /* integer representation of sei              */
  int len;               /* integer length of sei string               */
  int lower, upper;      /* upper and lower bounds for sei search      */
  char low[7], high[7];  /* upper and lower bounds for sei search      */
  int count, i;


  /* check to see if the search is an exact search on the last  */
  /* name or only a prefix search                               */
  exact = check_search_type(usrquery[0].lastname);

  /* select the file where the user's database information will */
  /* be stored and request the files from the jukebox           */
  assign_user_files(dbase, user);
```

```
count = req_user_data(user, usrquery, exact);
if(count > 20)
   return(FALSE);

/* create the query information for the last name            */
strcpy(query_last, "\"");
strcat(query_last, usrquery[0].lastname);
if(exact == FALSE)
    strcat(query_last, "*");
strcat(query_last, "\"");


/* create the where portion of the user's query              */
sprintf(selquery, "%s %s", "WHERE lastname matches", query_last);

/* If the user entered initials, add them to the query string */
if(strlen(usrquery[0].initials))
  {
  strcat(selquery, " AND initials matches \"");
  strcat(selquery, usrquery[0].initials);
  strcat(selquery, "\"");
  }

/* If the user entered a ssn, add it to the query string      */
if(strlen(usrquery[0].ssno))
  {
  strcat(selquery, " AND ssn matches \"");
  strcat(selquery, usrquery[0].ssno);
  strcat(selquery, "\"");
  }

/* If the user entered a spouse's ssn, add it to the query    */
if(strlen(usrquery[0].sp_ssno))
  {
  strcat(selquery, " AND sp_ssn matches \"");
  strcat(selquery, usrquery[0].sp_ssno);
  strcat(selquery, "\"");
  }

/* If the user entered a sei, add it to the query string      */
/* searching for +/- 1 of the sei value entered by the user   */
if(strlen(usrquery[0].sei))
  {
  sei = atoi(usrquery[0].sei);
  upper = sei + 1;
  lower = sei - 1;
  sprintf(low, "%d", lower);
  sprintf(high, "%d", upper);
  len = strlen(low);
  switch(len)
    {
    case 1:
         low[4] = low[0];
         low[3] = '0';
         low[2] = '0';
         low[1] = '0';
         low[0] = '0';
         low[5] = 0;
         break;
    case 2:
         low[4] = low[1];
         low[3] = low[0];
         low[2] = '0';
         low[1] = '0';
         low[0] = '0';
         low[5] = 0;
```

```
          break;
     case 3:
          low[4] = low[2];
          low[3] = low[1];
          low[2] = low[0];
          low[1] = '0';
          low[0] = '0';
          low[5] = 0;
          break;
     case 4:
          low[4] = low[3];
          low[3] = low[2];
          low[2] = low[1];
          low[1] = low[0];
          low[0] = '0';
          low[5] = 0;
          break;
     }
   len = strlen(high);
   switch(len)
     {
     case 1:
          high[4] = high[0];
          high[3] = '0';
          high[2] = '0';
          high[1] = '0';
          high[0] = '0';
          high[5] = 0;
          break;
     case 2:
          high[4] = high[1];
          high[3] = high[0];
          high[2] = '0';
          high[1] = '0';
          high[0] = '0';
          high[5] = 0;
          break;
     case 3:
          high[4] = high[2];
          high[3] = high[1];
          high[2] = high[0];
          high[1] = '0';
          high[0] = '0';
          high[5] = 0;
          break;
     case 4:
          high[4] = high[3];
          high[3] = high[2];
          high[2] = high[1];
          high[1] = high[0];
          high[0] = '0';
          high[5] = 0;
          break;
     }

   strcat(selquery, " AND sei between \"");
   strcat(selquery, low);
   strcat(selquery, "\"");
   strcat(selquery, " AND ");
   strcat(selquery, "\"");
   strcat(selquery, high);
   strcat(selquery, "\"");
   }


sprintf(unionqry, "%s %s %s", "SELECT * FROM", dbase[0], selquery);
```

```
/* string all of the databases to union */
for(i = 1; i < count; i++)
  {
  strcat(unionqry, " UNION ALL ");
  strcat(unionqry, "SELECT * FROM ");
  strcat(unionqry, dbase[i]);
  strcat(unionqry, " ");
  strcat(unionqry, selquery);
  }


/* select the easear database and check for status                */
switch(user)
  {
  case 1:
      $database user1;
      break;
  case 2:
      $database user2;
      break;
  case 3:
      $database user3;
      break;
  case 4:
      $database user4;
      break;
  case 5:
      $database user5;
      break;
  case 6:
      $database user6;
      break;
  case 7:
      $database user7;
      break;
  case 8:
      $database user8;
      break;
  }

if(sqlca.sqlcode)
  {
  disp_err("ERROR OPENING THE DATABASE");
  endwin();
  exit();
  }

/* prepare the query id from the user query string                */
$prepare qid from $unionqry;
if(sqlca.sqlcode)
  {
  disp_err("ERROR PREPARING THE QUERY ID");
  endwin();
  exit();
  }

/* Declare a scrolling cursor for the query                       */
$declare recptr scroll cursor for qid;
if(sqlca.sqlcode)
  {
  disp_err("ERROR DECLARING THE SCROLL CURSOR");
  endwin();
  exit();
  }

/* Open the query                                                 */
```

```
  $open recptr;
  if(sqlca.sqlcode)
    {
    disp_err("ERROR OPENING THE QUERY");
    endwin();
    exit();
    }
  return(TRUE);
  }




/*
 * Display Matches
 *
 * Input:  structure containing the user's query information
 *         arrays containing the query matches loaded in memory
 *         current match number
 *
 * Output: the matches are displayed on the user's screen
 */

int display_matches(usrquery, querymatches, full_record, curmatch)
  struct QUERY querymatches[];
  struct REC full_record[];
  struct INFO usrquery[];
  int *curmatch;
  {
  int nummatches, diff;
  $int topscr;

  /* display the header information and the first screen of matches */
  view_header(usrquery);

  /* load one screen of matches                                       */
  nummatches = load_matches(querymatches, full_record);

  if(nummatches != 0)
    {
     *curmatch = 0;
     /* Display the screen of matches                                 */
     disp_screen(querymatches, nummatches);

     /* Return the pointer to the top of the set of matches          */
     diff = MATCHSCR - nummatches;
     topscr = -((MATCHSCR - 1) - diff);
     $fetch relative $topscr recptr;
    }

  /* return the number of matches found                               */
  return(nummatches);
  }



/*
 *
 * Browse the initial matches
 *
 * Input:  user's directional information
 *         total number of matches in memory
 *         current match number
 *         user's query information
 *         array holding the current matches
 *
 * Output: the user's directional command is decoded and action is
 *         taken based upon it
 *
 */

browse(input, nummatch, curmatch, usrquery, querymatches, full_record, pminfo)
   int *curmatch, *nummatch;
   char input;
   struct MANAGEINFO *pminfo;
   struct QUERY querymatches[];
   struct INFO usrquery[];
   struct REC full_record[];
   {
   int row, col;
   int i;

   /* get current screen cursor position                                  */
   getyx(general, row, col);

   /* take action based upon user's command                               */
   switch(input)
     {
     case UPARROW:
          if(row > FIRSTROW)
             {
             *curmatch = *curmatch - 1;
             wmove(general, row-2, col);
             }
          break;
     case DOWNARROW:
          if( (row < LASTROW) && (*curmatch < (*nummatch-1)) )
             {
             *curmatch = *curmatch + 1;
             wmove(general, row+2, col);
             }
          break;
     case LEFTARROW:
          if(col > MINVIEW)
             wmove(general, row, col-1);
          break;
     case RIGHTARROW:
          if(col < MAXVIEW)
             wmove(general, row, col+1);
          else if(col == MAXVIEW)
             move_line(*curmatch, querymatches);
          break;
     case END:      /* user requests off the screen information           */
          move_line(*curmatch, querymatches);
          break;
     case HOME:     /* restore screen to default position                 */
          restore_line(*curmatch, querymatches);
          break;
     case ENTER:    /* user requests to see detailed info on a record */
          if(*nummatch != 0)
            {
            inc_detail_count(usrquery[0].year, pminfo);
            disp_rec(*curmatch, querymatches, full_record, pminfo);
            /* once the user is done, restore screen to browse mode   */
            restore_scr(*nummatch,*curmatch,usrquery,querymatches,full_record);
            }
          break;
     case PAGEUP:
          *nummatch = view_prev_matches(querymatches, full_record,
                                        curmatch, *nummatch);
          break;
     case PAGEDOWN:
          *nummatch = view_next_matches(querymatches, full_record,
                                     curmatch, *nummatch);
          break;
     }
    }



/*
 * Clear the old Query information, and place a copy of it in
 * the next array slot so if the user wishes it can be retrieved
 *
 * Input:  structure holding the query information
 *
 * Output: Main query array is cleared, and the storage query array
 *         contains the last query
 *
 */

clrquery(usrquery, first)
  struct INFO usrquery[];
  int first;
  {
  int i;

  /* copy the last query information to storage array                */
  if(!first)
    {
    strcpy(usrquery[1].year, usrquery[0].year);
    strcpy(usrquery[1].ird, usrquery[0].ird);
    strcpy(usrquery[1].lastname, usrquery[0].lastname);
    strcpy(usrquery[1].initials, usrquery[0].initials);
    strcpy(usrquery[1].ssno, usrquery[0].ssno);
    strcpy(usrquery[1].sp_ssno, usrquery[0].sp_ssno);
    strcpy(usrquery[1].sei, usrquery[0].sei);
    }

  /* clear the main query array                                        */
  for(i = 0; i < 3; i++)
    usrquery[0].year[i] = 0;

  for(i = 0; i < 3; i++)
    usrquery[0].ird[i] = 0;

  for(i = 0; i < 8; i++)
    usrquery[0].lastname[i] = 0;

  for(i = 0; i < 3; i++)
    usrquery[0].initials[i] = 0;

  for(i = 0; i < 10; i++)
    usrquery[0].ssno[i] = 0;

  for(i = 0; i < 10; i++)
    usrquery[0].sp_ssno[i] = 0;

  for(i = 0; i < 6; i++)
    usrquery[0].sei[i] = 0;

  /* close the database filter                                         */
  $close recptr;
  }
```

```
/*******************************************************************************/
/*                                                                             */
/*                     Small Functions Called by main()                        */
/*                                                                             */
/*******************************************************************************/


/*
 * Determine the user number of the current user (by tty port)
 * This is used to allow access to files, and for management information
 *
 * Output: user number is returned to the calling function
 */

int get_user_no()
  {
  char *tty, *ttyname();
  int user;

  tty = ttyname(0);        /* return where the standard out is going      */
  user = *(tty+9)  - 64;   /* get rid of "/dev/tty4" and point to letter */
                           /* 64 = @ - the ascii character before 'A'     */
  return(user);
  }



/*
 * Initialize the MI information structure
 *
 * Input: global structure minfo holding the management data
 *
 * Output: all values are set to 0 (not ANSI compiler)
 *
 */

init_mi(pminfo)
  struct MANAGEINFO *pminfo;
  {
  pminfo->logtime = 0;
  pminfo->logcount = 0;
  pminfo->qt88 = 0;
  pminfo->qt89 = 0;
  pminfo->qc88 = 0;
  pminfo->qc89 = 0;
  pminfo->dc88 = 0;
  pminfo->dc89 = 0;
  pminfo->pc88 = 0;
  pminfo->pc89 = 0;
  }



/*
 * Increment the login count
 *
 */

inc_log_count(pminfo)
  struct MANAGEINFO *pminfo;
  {
  pminfo->logcount++;
  }
```

```
/*
 * Start the timer for the login length
 *
 */

start_log_timer(pminfo)
  struct MANAGEINFO *pminfo;
  {
  pminfo->ltimer1 = time( (long *) 0);
  }



/*
 * End the timer for the login length
 *
 */

 end_log_timer(pminfo)
  struct MANAGEINFO *pminfo;
   {
   long int timer;
   long int min;

   pminfo->ltimer2 = time( (long *) 0);
   timer = pminfo->ltimer2 - pminfo->ltimer1;

   min = (long int) (timer/60) + 1;
   pminfo->logtime += min;
   }




/*
 * Increment the query count
 *
 */

inc_query_count(year, pminfo)
  char year[];
  struct MANAGEINFO *pminfo;
  {
  if(year[1] == '8')
     pminfo->qc88++;
  else if(year[1] == '9')
     pminfo->qc89++;
  }




/*
 * Start the timer for the query length
 *
 */

start_query_timer(pminfo)
  struct MANAGEINFO *pminfo;
  {
  pminfo->qtimer1 = time( (long *) 0);
  }
```

```
/*
 * End the timer for the query length
 *
 */

 end_query_timer(year, pminfo)
   struct MANAGEINFO *pminfo;
   char year[];
   {
   long int timer;
   long int min;

   pminfo->qtimer2 = time( (long *) 0);
   timer = pminfo->qtimer2 - pminfo->qtimer1;

   min = (long int) (timer/60) + 1;
   if(year[1] == '8')
      pminfo->qt88 += min;
   else if(year[1] == '9')
      pminfo->qt89 += min;
   }




/*
 * Increment the detail count
 *
 */

inc_detail_count(year, pminfo)
  char year[];
  struct MANAGEINFO *pminfo;
  {
  if(year[1] == '8')
     pminfo->dc88++;
  else if(year[1] == '9')
     pminfo->dc89++;
  }



/*
 * Increment the print count
 *
 */

inc_print_count(year, pminfo)
  char year[];
  struct MANAGEINFO *pminfo;
  {
  if(year[1] == '8')
     pminfo->pc88++;
  else if(year[1] == '9')
     pminfo->pc89++;
  }
```

```
/*
 * Write the management data to the disk
 *
 * Input: the user number
 *
 * Output: Management information is written to a disk file
 *
 */

write_mi_stats(user, pminfo)
  int user;
  struct MANAGEINFO *pminfo;
  {
  FILE *fopen(), *mi;
  char filename[20];
  struct MANAGEINFO tmp;

  /* Create mi_file for this user */
  sprintf(filename, "%s%d", "mi_info.", user);

  /* Check to see if file exists, if so -- read in the current stats */
  mi = fopen(filename, "r");
  if(mi != NULL)
    {
    fscanf(mi, "%d%*c", &tmp.logtime);
    fscanf(mi, "%d%*c", &tmp.logcount);
    fscanf(mi, "%d%*c", &tmp.qt88);
    fscanf(mi, "%d%*c", &tmp.qt89);
    fscanf(mi, "%d%*c", &tmp.qc88);
    fscanf(mi, "%d%*c", &tmp.qc89);
    fscanf(mi, "%d%*c", &tmp.dc88);
    fscanf(mi, "%d%*c", &tmp.dc89);
    fscanf(mi, "%d%*c", &tmp.pc88);
    fscanf(mi, "%d%*c", &tmp.pc89);

    /* Add new statistics */
    pminfo->logtime += tmp.logtime;
    pminfo->logcount += tmp.logcount;
    pminfo->qt88 += tmp.qt88;
    pminfo->qt89 += tmp.qt89;
    pminfo->qc88 += tmp.qc88;
    pminfo->qc89 += tmp.qc89;
    pminfo->dc88 += tmp.dc88;
    pminfo->dc89 += tmp.dc89;
    pminfo->pc88 += tmp.pc88;
    pminfo->pc89 += tmp.pc89;
    }
  fclose(mi);

  /* Open the file destroying old data if any, and write new data */
  mi = fopen(filename, "w");
  fprintf(mi, "%d\n", pminfo->logtime);
  fprintf(mi, "%d\n", pminfo->logcount);
  fprintf(mi, "%d\n", pminfo->qt88);
  fprintf(mi, "%d\n", pminfo->qt89);
  fprintf(mi, "%d\n", pminfo->qc88);
  fprintf(mi, "%d\n", pminfo->qc89);
  fprintf(mi, "%d\n", pminfo->dc88);
  fprintf(mi, "%d\n", pminfo->dc89);
  fprintf(mi, "%d\n", pminfo->pc88);
  fprintf(mi, "%d\n", pminfo->pc89);
  fclose(mi);
  }
```

```
/*****************************************************************************/
/*                                                                           */
/*                    Functions Called by Get_Query()                        */
/*                                                                           */
/*****************************************************************************/



/*
 * Determines if the user has entered an invalid year
 *
 * Input:  Array containing the year as a string
 *
 * Output: True if the year is invalid, false if not
 *
 */

invalid_year(year)
  char year[];
  {
  int different;

  different = strcmp("88", year);
  if(different)
    different = strcmp("89", year);

  if(different)
    return(TRUE);
  else
    return(FALSE);
  }


/*
 * Determines if the user has entered an invalid IRD
 *
 * Input:  Array containing the IRD as a string
 *
 * Output: True if the IRD is invalid, false if not
 *
 */

invalid_ird(ird)
  char ird[];
  {
  int irdint;

  irdint = atoi(ird);
  switch(irdint)
    {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
    case 11:
    case 21:
    case 23:
    case 31:
    case 35:
    case 36:
```

```
    case 38:
    case 39:
    case 41:
    case 42:
    case 43:
    case 45:
    case 46:
    case 47:
    case 48:
    case 51:
    case 52:
    case 54:
    case 55:
    case 56:
    case 57:
    case 58:
    case 59:
    case 61:
    case 62:
    case 63:
    case 64:
    case 66:
    case 71:
    case 72:
    case 73:
    case 74:
    case 81:
    case 82:
    case 83:
    case 84:
    case 85:
    case 86:
    case 87:
    case 88:
    case 91:
    case 92:
    case 93:
    case 95:
    case 96:
    case 99:
        return(FALSE);
    default:
        return(TRUE);
    }
  }


/*
 * interprets the users input and acts upon it
 *
 * input:  character entered by the user
 *         structure to hold query information
 *
 * output: action is taken depending upon the character passed
 */

interp(input, usrquery)
  char input;
  struct INFO usrquery[];
  {
  int row, col;

  /* obtain current xy coordinates */
  getyx(general, row, col);

  /* take action according to the user's input */
 switch(input)
   {
   case UPARROW:
   case DOWNARROW:
   case LEFTARROW:
   case RIGHTARROW:
      proc_arrow(input, row, col);
      break;
   case PAGEUP:  /* display year and IRD from the last query */
      disp_string(general, ROWTAX, MINCOL, usrquery[1].year);
      disp_string(general, ROWIRD, MINCOL, usrquery[1].ird);
      strcpy(usrquery[0].year, usrquery[1].year);
      strcpy(usrquery[0].ird, usrquery[1].ird);
      wmove(general, ROWLAST, MINCOL);
      wrefresh(general);
      break;
   case TAB:  /* Tab to the next field */
      switch(row)
        {
        case ROWTAX:
             wmove(general, ROWIRD, MINCOL);
             break;
        case ROWIRD:
             wmove(general, ROWLAST, MINCOL);
             break;
        case ROWLAST:
             if(col >= COLINI)
                wmove(general, ROWSSN, MINCOL);
             else
                wmove(general, ROWINI, COLINI);
             break;
        case ROWSSN:
             wmove(general, ROWSP, MINCOL);
             break;
        case ROWSP:
             wmove(general, ROWSEI, MINCOL);
             break;
        default:
             wmove(general, ROWTAX, MINCOL);
             break;
      }
      break;
   case BACKSPACE:
        del_char(row, col, usrquery);
        break;
   default:
     input = toupper(input);
     add_char(input, row, col, usrquery);
     break;
   }
}



/*
 * Processes an arrow key that is entered at the query screen
 *
 * Input:  user's directional key
 *         current row and column of cursor
 *
 * Output: cursor is moved in the direction indicated by the user's
 *         entry
 *
 */
```

```
proc_arrow(input, row, col)
    char input;
    int row, col;
    {
    switch(input)
      {
      case UPARROW:
         if(row > ROWTAX)  /* If not at the top of the query area */
           {
           /* Move to the previous field */
           if( (row == ROWINI) && (col >= COLINI) )
              wmove(general, ROWLAST, MINCOL);
           else if(row == ROWSSN)
              wmove(general, ROWINI, COLINI);
           else
              wmove(general, row-2, MINCOL);
           }
         wrefresh(general);  /* refresh the screen */
         break;
      case DOWNARROW:
         if(row < ROWSEI)  /* If not at the bottom of the query area */
           {
           /* Move down to the next field */
           if( (row == ROWLAST) && (col < COLINI) )
             wmove(general, ROWINI, COLINI);
           else
             wmove(general, row+2, MINCOL);
           }
         wrefresh(general);  /* refresh the screen */
         break;
      case LEFTARROW:
         if(col > MINCOL)  /* If not at the far left of the query area */
           wmove(general, row, col-1); /* Move to the left */
         wrefresh(general);
         break;
      case RIGHTARROW:
         if(col < MAXCOL) /* if not at the far right */
            wmove(general, row, col+1);  /* Move to the right */
         wrefresh(general);
         break;
      }
   }



/*
 * Process a backspace character by deleting the last character
 * in the query array
 *
 * Input:  current row and column of cursor
 *         array with user's query
 *
 * Output: last character is deleted from screen and from correct
 *         query array
 *
 */

del_char(row, col, usrquery)
    int row, col;
    struct INFO usrquery[];
    {
    switch(row)
       {
```

```
case ROWTAX:
     if((col > MINCOL) && ((col-MINCOL) < 3))     /* In entry area?   */
       {
       /* delete character from screen */
       wmove(general, row, col-1);
       waddstr(general, "-");
       wmove(general, row, col-1);
       wrefresh(general);
       usrquery[0].year[(col-1)-MINCOL] = 0;      /* delete character */
       }
     break;
case ROWIRD:
     if((col > MINCOL) && ((col-MINCOL) < 3))     /* In entry area?   */
       {
       /* delete character from screen */
       wmove(general, row, col-1);
       waddstr(general, "-");
       wmove(general, row, col-1);
       wrefresh(general);
       usrquery[0].ird[(col-1)-MINCOL] = 0;       /* delete character */
       }
     break;
case ROWLAST:
     if(col < COLINI)                            /* In last name area? */
       if(((col-MINCOL) < 8) && (col > MINCOL))     /* In entry area?  */
         {
         /* delete character from screen */
         wmove(general, row, col-1);
         waddstr(general, "-");
         wmove(general, row, col-1);
         wrefresh(general);
         usrquery[0].lastname[(col-1)-MINCOL] = 0; /* Delete Char  */
         }
     else                                        /* In initials area   */
       if(((col-COLINI) < 3) && (col > COLINI)) /* In entry area?     */
         {
         /* delete character from screen */
         wmove(general, row, col-1);
         waddstr(general, "-");
         wmove(general, row, col-1);
         wrefresh(general);
         usrquery[0].initials[(col-1)-COLINI] = 0; /* Delete char  */
         }
     break;
case ROWSSN:
     if(((col - MINCOL) < 10) && (col > MINCOL)) /* In entry area? */
       {
       /* delete character from screen */
       wmove(general, row, col-1);
       waddstr(general, "-");
       wmove(general, row, col-1);
       wrefresh(general);
       usrquery[0].ssno[(col-1)-MINCOL] = 0;      /* Delete character */
       }
     break;
case ROWSP:
     if(((col - MINCOL) < 10) && (col > MINCOL)) /* In entry area? */
       {
       /* delete character from screen */
       wmove(general, row, col-1);
       waddstr(general, "-");
       wmove(general, row, col-1);
       wrefresh(general);
       usrquery[0].sp_ssno[(col-1)-MINCOL] = 0;  /* Delete character */
       }
     break;
```

```
        case ROWSEI:
             if(((col - MINCOL) < 6) && (col > MINCOL)) /* In entry area?   */
               {
               /* delete character from screen */
               wmove(general, row, col-1);
               waddstr(general, "-");
               wmove(general, row, col-1);
               wrefresh(general);
               usrquery[0].sei[(col-1)-MINCOL] = 0;     /* Delete character */
               }
             break;
        }
    }



/*
 * Add a character to the query array and place it on the screen.  Filters
 * to accept only letters, decimal numbers, spaces, aprostropes, dashes or
 * periods.  Other characters are disregarded
 *
 * Input:  User entered character
 *         current row and column of cursor
 *         structure holding query
 *
 * Output: character is placed on the screen and in the currect query
 *         array, if it is a valid input
 *
 */

add_char(input, row, col, usrquery)
   char input;
   int row, col;
   struct INFO usrquery[];
   {
   switch(row)
      {
      case ROWTAX:
            if( (col-MINCOL) < 2)
              {
              usrquery[0].year[col-MINCOL] = input;
              waddch(general, input);
              if( (col - MINCOL) == 1)
                wmove(general, ROWIRD, MINCOL);
              }
            break;
      case ROWIRD:
            if( (col-MINCOL) < 2)
              {
              usrquery[0].ird[col-MINCOL] = input;
              waddch(general, input);
              if( (col - MINCOL) == 1)
                wmove(general, ROWLAST, MINCOL);
              }
            break;
      case ROWLAST:
            if(col < COLINI)
             {
              if( (col-MINCOL) < 7)
                {
                usrquery[0].lastname[col-MINCOL] = input;
                waddch(general, input);
                if( (col - MINCOL) == 6)
                wmove(general, ROWINI, COLINI);
            }
          }
        else
          {
          if( (col - COLINI) < 2)
            {
            usrquery[0].initials[col-COLINI] = input;
            waddch(general, input);
            if( (col - COLINI) == 1)
                wmove(general, ROWSSN, MINCOL);
            }
          }
        break;
    case ROWSSN:
        if( (col-MINCOL) < 9)
          {
          usrquery[0].ssno[col-MINCOL] = input;
          waddch(general, input);
          if( (col - MINCOL) == 8)
              wmove(general, ROWSP, MINCOL);
          }
        break;
    case ROWSP:
        if( (col - MINCOL) < 9)
          {
          usrquery[0].sp_ssno[col-MINCOL] = input;
          waddch(general, input);
          if( (col - MINCOL) == 8)
              wmove(general, ROWSEI, MINCOL);
          }
        break;
    case ROWSEI:
        if( (col - MINCOL) < 5)
          {
          usrquery[0].sei[col-MINCOL] = input;
          waddch(general, input);
          }
        break;
    }
  }




/*****************************************************************************/
/*                                                                           */
/*                    Functions Called by Select_Matches()                   */
/*                                                                           */
/*****************************************************************************/


/*
 * Assign a set of user files to hold the current data
 *
 * Input:  Multidimensional Array to hold names assigned to the user
 *         user number of the current user
 *
 * Output: Array is filled with the user files
 *
 */
```

```
assign_user_files(dbase, user)
  char dbase[20][10];
  int user;
  {
  switch(user)
    {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
    case 7:
    case 8:
       strcpy(dbase[0], "one");
       strcpy(dbase[1], "two");
       strcpy(dbase[2], "three");
       strcpy(dbase[3], "four");
       strcpy(dbase[4], "five");
       strcpy(dbase[5], "six");
       strcpy(dbase[6], "seven");
       strcpy(dbase[7], "eight");
       strcpy(dbase[8], "nine");
       strcpy(dbase[9], "ten");
       strcpy(dbase[10], "eleven");
       strcpy(dbase[11], "twelve");
       strcpy(dbase[12], "thirteen");
       strcpy(dbase[13], "fourteen");
       strcpy(dbase[14], "fifteen");
       strcpy(dbase[15], "sixteen");
       strcpy(dbase[16], "seventeen");
       strcpy(dbase[17], "eighteen");
       strcpy(dbase[18], "nineteen");
       strcpy(dbase[19], "twenty");
       break;
    default:
       disp_err("Invalid User -- Please go to an authorized Terminal");
       endwin();
       exit();
       break;
    }
  }


/*
 * Request that the files on the jukebox be downloaded to the
 * user files
 *
 * Input:  user number
 *         query information
 *         flag for type of search
 *
 * Output: messages are exchanged with the file handler, and the
 *         correct data is copied to the user files
 *
 */

req_user_data(user, queryinfo, exact)
  struct INFO queryinfo[];
  int user, exact;
   {
   $ char disknum[5];
   $ char sidenum[5];
   $ char jukenum[5];
   $ char files[10];
   $ char query[1000];
```

```
char filename[20][10];
char mountdir[15];
char querylast[15], lastlen[3];
char queryinit[7];
int count, initsearch, len;
int jnum, dnum, snum, i;

/*  Determine if searching on initials or not */
if(strlen(queryinfo[0].initials) > 0)
    initsearch = TRUE;
else
    initsearch = FALSE;

/* Determine length of last name and set up last name for query */
len = strlen(queryinfo[0].lastname);
sprintf(lastlen, "%d", len);
sprintf(queryinit, "%s%s%s", "\"", queryinfo[0].initials, "\"");
sprintf(querylast, "%s%s%s", "\"", queryinfo[0].lastname, "\"");

/* setup query */
stcopy("select filenm, disk, side, juke from i", query);
stcat(queryinfo[0].ird, query);
stcat(queryinfo[0].year, query);
if( (initsearch == FALSE) && (exact == FALSE) )
  {
  stcat(" where fname[1,", query);
  stcat(lastlen, query);
  stcat("] = ", query);
  stcat(querylast, query);
  stcat(" or lname[1,", query);
  stcat(lastlen, query);
  stcat("] = ", query);
  stcat(querylast, query);
  stcat(" or ", query);
  stcat(querylast, query);
  stcat(" between fname and lname", query);
  }
if( (initsearch == FALSE) && (exact == TRUE) )
  {
  stcat(" where ", query);
  stcat(querylast, query);
  stcat(" between fname and lname", query);
  }
if( (initsearch == TRUE) && (exact == TRUE) )
  {
  stcat(" where fname <= ", query);
  stcat(querylast, query);
  stcat(" and lname >= ", query);
  stcat(querylast, query);
  stcat(" and lname > fname", query);
  stcat(" or fname = ", query);
  stcat(querylast, query);
  stcat(" and lname = fname and ", query);
  stcat(queryinit, query);
  stcat(" between finit and linit", query);
  }
if( (initsearch == TRUE) && (exact == FALSE) )
  {
  stcat(" where fname[1,", query);
  stcat(lastlen, query);
  stcat("] = ", query);
  stcat(querylast, query);
  stcat(" and lname > fname", query);
  stcat(" or lname[1,", query);
  stcat(lastlen, query);
  stcat("] = ", query);
```

```
   stcat(querylast, query);
   stcat(" and lname > fname or fname[1,", query);
   stcat(lastlen, query);
   stcat("] = ", query);
   stcat(querylast, query);
   stcat(" and lname = fname and ", query);
   stcat(queryinit, query);
   stcat(" between finit and linit", query);
   stcat(" or ", query);
   stcat(querylast, query);
   stcat(" between fname and lname", query);
   stcat(" and lname not matches fname", query);
   }

 if(queryinfo[0].year[1] == '8')
   $database ind88;
 else
   $database ind89;
 if(sqlca.sqlcode)
  {
  disp_err("ERROR OPENING DATABASE");
  endwin();
  exit();
  }

 $prepare query_id from $query;
 if(sqlca.sqlcode)
  {
  disp_err("ERROR PREPARING THE QUERY ID");
  endwin();
  exit();
  }


 $declare rptr cursor for query_id;
 if(sqlca.sqlcode)
  {
  disp_err("ERROR DECLARING THE QUERY ID");
  endwin();
  exit();
  }


 $open rptr;
 if(sqlca.sqlcode)
  {
  disp_err("ERROR OPENING THE QUERY");
  endwin();
  exit();
  }

 count = 0;
 while(sqlca.sqlcode != SQLNOTFOUND)
   {
   $fetch next rptr into $files, $disknum, $sidenum, $jukenum;
   if(sqlca.sqlcode != SQLNOTFOUND)
     {
     if(count < 20)
       {
       stcopy(files, filename[count]);
       for(i = 0; i < 10; i++)
          if(filename[count][i] == ' ')
             {
             filename[count][i] = 0;
             break;
             }
          }
        count++;
        }
      }

   $close rptr;


   if(count < 21)
     {
     jnum = atoi(jukenum);
     dnum = atoi(disknum);
     snum = atoi(sidenum);

     clrwin(ERRORMSG);
     disp_string(errormsg, 2, 10, "Requesting Disk from jukebox");
     wrefresh(errormsg);
     req_disk(user, jnum, dnum, snum, mountdir);
     clrwin(ERRORMSG);
     disp_string(errormsg, 2, 10, "Copying User files");
     wrefresh(errormsg);
     copy_files(user, mountdir, filename, count);
     clrwin(ERRORMSG);
     disp_string(errormsg, 2, 10, "Decompressing Files");
     wrefresh(errormsg);
     ret_disk(user, jnum, dnum, snum);
     decomp_files(user, count);
     clrwin(ERRORMSG);
     disp_string(errormsg, 2, 10, "Retrieving Matches");
     wrefresh(errormsg);
     }
   return(count);
   }


/*
 * Request that the disk with the data be mounted
 *
 * Input:  Jukebox number
 *         Disk number
 *         Side of Optical disk
 *         Array to hold the directory name that the disk was mounted to
 *
 * Output: A drive is allocated, mounted, and the directory is returned
 *
 */

req_disk(user, jukenum, disknum, sidenum, mountdir)
  char mountdir[];
  int jukenum, disknum, sidenum;
  int user;
  {
  struct MSGACK msgack, *pmsgack;
  struct MSGBUF message, *pmessage;
  int queueid, ackid, res;

  pmessage = &message;
  pmsgack = &msgack;

  ackid = msgget(KEY3, 0);
  queueid = msgget(KEY1, 0);
  if( (queueid < 0) || (ackid < 0) )
    {
    disp_err("CANNOT COMMUNICATE WITH HANDLER -- PLEASE ALERT COMPUTER ROOM");
    endwin();
    exit();
```

```
     }

  pmessage->mtype = MOUNT;
  pmessage->user = user;
  pmessage->juke = jukenum;
  pmessage->disk = disknum;
  pmessage->side = sidenum;
  res = msgsnd(queueid, pmessage, MSGLEN, IPC_NOWAIT);
  if(res < 0)
     {
     disp_err("CANNOT COMMUNICATE WITH HANDLER -- PLEASE ALERT COMPUTER ROOM");
     endwin();
     exit();
     }

  msgrcv(ackid, pmsgack, ACKLEN, ACK+user, WAIT);
  strcpy(mountdir, pmsgack->mtdir);
  }


/*
 * Copies the necessary files from the jukebox to the user's files
 *
 * Input:   User Number
 *          Directory to which the drive is mounted
 *          Array containing the files to copy
 *          Count of files to copy
 *
 * Output:  Files are copied from the jukebox to the hard drive
 *
 */

copy_files(user, mountdir, filename, count)
  char mountdir[], filename[20][10];
  int count, user;
  {
  char sourcefile[50];
  char destfile[50];
  char basename[20][20];
  int i;
  char command[125];
  char direct[5];

  strcpy(basename[0], "one");
  strcpy(basename[1], "two");
  strcpy(basename[2], "three");
  strcpy(basename[3], "four");
  strcpy(basename[4], "five");
  strcpy(basename[5], "six");
  strcpy(basename[6], "seven");
  strcpy(basename[7], "eight");
  strcpy(basename[8], "nine");
  strcpy(basename[9], "ten");
  strcpy(basename[10], "eleven");
  strcpy(basename[11], "twelve");
  strcpy(basename[12], "thirteen");
  strcpy(basename[13], "fourteen");
  strcpy(basename[14], "fifteen");
  strcpy(basename[15], "sixteen");
  strcpy(basename[16], "seventeen");
  strcpy(basename[17], "eighteen");
  strcpy(basename[18], "nineteen");
  strcpy(basename[19], "twenty");

  sprintf(direct, "%d", user);
  for(i = 0; i < count; i++)
```

```
    {
    strcpy(sourcefile, mountdir);
    strcat(sourcefile, "/");
    strcat(sourcefile, filename[i]);
    strcat(sourcefile, ".dat.z");

    strcpy(destfile, "/usr/pilot/junk/");
    strcat(destfile, direct);
    strcat(destfile, "/");
    strcat(destfile, basename[i]);
    strcat(destfile, ".dat.z");

    strcpy(command, "cp ");
    strcat(command, sourcefile);
    strcat(command, " ");
    strcat(command, destfile);
    system(command);

    strcpy(sourcefile, mountdir);
    strcat(sourcefile, "/");
    strcat(sourcefile, filename[i]);
    strcat(sourcefile, ".idx.z");

    strcpy(destfile, "/usr/pilot/junk/");
    strcat(destfile, direct);
    strcat(destfile, "/");
    strcat(destfile, basename[i]);
    strcat(destfile, ".idx.z");

    strcpy(command, "cp ");
    strcat(command, sourcefile);
    strcat(command, " ");
    strcat(command, destfile);
    system(command);
    }
  }



/*
 * Decompress the Database files
 *
 * Input:  User Number
 *         Count of the files
 *
 * Output: Files are decompressed
 *
 */

decomp_files(user, count)
  int count, user;
  {
  char command[125];
  char basename[20][20];
  char name[20][20];
  char direct[5];
  char sourcefile[50];
  char destfile[50];
  int i;

  strcpy(basename[0], "one____100");
  strcpy(basename[1], "two____101");
  strcpy(basename[2], "three__102");
  strcpy(basename[3], "four___103");
  strcpy(basename[4], "five___104");
strcpy(basename[5], "six____105");
strcpy(basename[6], "seven__106");
strcpy(basename[7], "eight__107");
strcpy(basename[8], "nine___108");
strcpy(basename[9], "ten____109");
strcpy(basename[10], "eleven_110");
strcpy(basename[11], "twelve_111");
strcpy(basename[12], "thirtee112");
strcpy(basename[13], "fourtee113");
strcpy(basename[14], "fifteen114");
strcpy(basename[15], "sixteen115");
strcpy(basename[16], "sevente116");
strcpy(basename[17], "eightee117");
strcpy(basename[18], "ninetee118");
strcpy(basename[19], "twenty_119");

strcpy(name[0], "one");
strcpy(name[1], "two");
strcpy(name[2], "three");
strcpy(name[3], "four");
strcpy(name[4], "five");
strcpy(name[5], "six");
strcpy(name[6], "seven");
strcpy(name[7], "eight");
strcpy(name[8], "nine");
strcpy(name[9], "ten");
strcpy(name[10], "eleven");
strcpy(name[11], "twelve");
strcpy(name[12], "thirteen");
strcpy(name[13], "fourteen");
strcpy(name[14], "fifteen");
strcpy(name[15], "sixteen");
strcpy(name[16], "seventeen");
strcpy(name[17], "eighteen");
strcpy(name[18], "nineteen");
strcpy(name[19], "twenty");

sprintf(direct, "%d", user);
for(i = 0; i < count; i++)
  {
  strcpy(command, "unpack ");
  strcat(command, "/usr/pilot/junk/");
  strcat(command, direct);
  strcat(command, "/");
  strcat(command, name[i]);
  strcat(command, ".dat.z 2> /dev/null");
  system(command);

  strcpy(command, "unpack ");
  strcat(command, "/usr/pilot/junk/");
  strcat(command, direct);
  strcat(command, "/");
  strcat(command, name[i]);
  strcat(command, ".idx.z 2> /dev/null");
  system(command);

  strcpy(sourcefile, "/usr/pilot/junk/");
  strcat(sourcefile, direct);
  strcat(sourcefile, "/");
  strcat(sourcefile, name[i]);
  strcat(sourcefile, ".dat");

  strcpy(destfile, "/usr/pilot/user");
  strcat(destfile, direct);
  strcat(destfile, ".dbs");
  strcat(destfile, "/");
```

```
    strcat(destfile, basename[i]);
    strcat(destfile, ".dat");

    strcpy(command, "cp ");
    strcat(command, sourcefile);
    strcat(command, " ");
    strcat(command, destfile);
    system(command);

    strcpy(sourcefile, "/usr/pilot/junk/");
    strcat(sourcefile, direct);
    strcat(sourcefile, "/");
    strcat(sourcefile, name[i]);
    strcat(sourcefile, ".idx");

    strcpy(destfile, "/usr/pilot/user");
    strcat(destfile, direct);
    strcat(destfile, ".dbs");
    strcat(destfile, "/");
    strcat(destfile, basename[i]);
    strcat(destfile, ".idx");

    strcpy(command, "cp ");
    strcat(command, sourcefile);
    strcat(command, " ");
    strcat(command, destfile);
    system(command);
    }

  strcpy(command, "rm /usr/pilot/junk/");
  strcat(command, direct);
  strcat(command, "/");
  strcat(command, "*.*");
  system(command);
  }



/*
 * Return a mounted disk when finished
 *
 * Input:  Request number
 *
 * Output: A drive is returned
 *
 */

ret_disk(user, jukenum, disknum, sidenum)
  int jukenum, disknum, sidenum;
  int user;
  {
  struct MSGBUF message, *pmessage;
  int queueid, res;

  pmessage = &message;
  queueid = msgget(KEY1, 0);
  if(queueid < 0)
    {
    disp_err("CANNOT COMMUNICATE WITH HANDLER -- PLEASE ALERT COMPUTER ROOM");
    endwin();
    exit();
    }

  pmessage->mtype = UNMOUNT;
  pmessage->user = user;
  pmessage->juke = jukenum;
  pmessage->disk = disknum;
  pmessage->side = sidenum;
  res = msgsnd(queueid, pmessage, MSGLEN, IPC_NOWAIT);
  if(res < 0)
    {
    disp_err("CANNOT COMMUNICATE WITH HANDLER -- PLEASE ALERT COMPUTER ROOM");
    endwin();
    exit();
    }
  }


/*
 * Check to see if the last name is to be searched as an exact
 * match, or as a prefix
 *
 * Input:  last name string
 *
 * Output: exact search (TRUE) or prefix search (FALSE)
 */

int check_search_type(last)
  char last[];
  {
  int exact = FALSE;
  int j = 0;

  /* check to see if the user has specified a search for the exact */
  /* of the last name, by placing a period at the end of the string */
  while(last[j] != 0)
     {
     if(last[j] == '.')
       {
       exact = TRUE;
       last[j] = 0;
       j--;
       }
     j++;
     }
   return(exact);
   }




/****************************************************************************/
/*                                                                          */
/*                    Functions Called by Display_Matches()                 */
/*                                                                          */
/****************************************************************************/



/*
 * Display the header information
 *
 * Input:  Structure holding the query information
 *
 * Output: Header is displayed for the user
 *
 */

view_header(usrquery)
  struct INFO usrquery[];
 *         array containing the browse information on the matches
 *
 */

restore_line(curmatch, querymatches)
 int curmatch;
 struct QUERY querymatches[];
 {
 int row, col, i, j;
 char buffer[25];

 /* clear the current row */
 getyx(general, row, col);
 clearline(general, row);

 /* display the rightmost part of the string */
 disp_string(general, row, 2, querymatches[curmatch].lastname);
 disp_string(general, row, 10, querymatches[curmatch].initials);
 space_ssn(buffer, querymatches[curmatch].ssno);
 disp_string(general, row, 15, buffer);
 disp_string(general, row, 28, querymatches[curmatch].sei);
 space_ssn(buffer, querymatches[curmatch].sp_ssno);
 disp_string(general, row, 36, buffer);
 disp_string(general, row, 50, querymatches[curmatch].sp_sei);

 /* Display the last name */
 wmove(general, row, 57);
 if( strlen(querymatches[curmatch].fullname) > 20)
   {
   for(j = 0; j < 20; j++)
      buffer[j] = querymatches[curmatch].fullname[j];
   buffer[20] = 0;
   waddstr(general, buffer);
   }
 else
   waddstr(general, querymatches[curmatch].fullname);

 wmove(general, row, 2);
 wrefresh(general);
 }




/*
 * Move the line to the left to view the remainder of the full name
 * field
 *
 * Input:  current match number
 *         array containing the browse information on the matches
 *
 */

move_line(curmatch, querymatches)
 int curmatch;
 struct QUERY querymatches[];
 {
 int row, col, i, j;
 char viewssn[12];

 /* clear the current row */
 getyx(general, row, col);
 clearline(general, row);

 /* display the rightmost part of the string */
 disp_string(general, row, 4, querymatches[curmatch].sei);
 space_ssn(viewssn, querymatches[curmatch].sp_ssno);
 disp_string(general, row, 10, viewssn);
 disp_string(general, row, 22, querymatches[curmatch].sp_sei);
 disp_string(general, row, 28, querymatches[curmatch].fullname);
 wrefresh(general);
 }




/*
 *
 * Display Selected Record in Detail
 *
 * Input:  current match number
 *         arrays holding the current set of matches
 *
 * Output: window with detailed information
 *
 *
 */

disp_rec(curmatch, querymatches, full_record, pminfo)
 struct QUERY querymatches[];
 struct REC full_record[];
 struct MANAGEINFO *pminfo;
 int curmatch;
 {
 int j, i;
 char input;
 char viewssn[12];

 /* Bring up the detail window */
 show_panel(pdetail);
 clrwin(DETAILWIN);

 /* Display name */
 disp_string(detail, 2, 2, "Name   : ");
 waddstr(detail, querymatches[curmatch].fullname);

 /* Display Address */
 disp_string(detail, 3, 2, "Address: ");
 waddstr(detail, full_record[curmatch].st_add);
 disp_string(detail, 4, 12, full_record[curmatch].city_st);
 waddstr(detail, "  ");
 waddstr(detail, full_record[curmatch].zip);

 /* Display Column Headings */
 disp_string(detail, 6, 4, "Name");
 disp_string(detail, 6, 14, "SSN");
 disp_string(detail, 6, 27, "NESEB");
 disp_string(detail, 6, 33, "NESEF");
 disp_string(detail, 6, 39, "WAGES");
 disp_string(detail, 6, 45, "SEI");
 disp_string(detail, 6, 52, "YEARQU");
 disp_string(detail, 6, 60, "ST");
 disp_string(detail, 6, 64, "DLN/EIN IND CD");
 disp_string(detail, 7, 2, "--");
 for(j = 0; j < 37; j++)
  waddstr(detail, "--");

 /* Display First row */
 disp_string(detail, 8, 2, querymatches[curmatch].lastname);
 disp_string(detail, 8, 10, querymatches[curmatch].initials);
 space_ssn(viewssn, querymatches[curmatch].ssno);
 disp_string(detail, 8, 14, viewssn);
disp_string(detail, 8, 27, full_record[curmatch].neseb);
disp_string(detail, 8, 33, full_record[curmatch].nesef);
disp_string(detail, 8, 39, full_record[curmatch].wages);
disp_string(detail, 8, 45, querymatches[curmatch].sei);
disp_string(detail, 8, 52, querymatches[curmatch].year);
waddstr(detail, full_record[curmatch].month);
waddstr(detail, full_record[curmatch].option_cd);
waddstr(detail, full_record[curmatch].se_quar);
disp_string(detail, 8, 60, querymatches[curmatch].ird);
disp_string(detail, 8, 64, full_record[curmatch].dln);


/* Display second Row */
disp_string(detail, 9, 5, full_record[curmatch].iwp);
disp_string(detail, 9, 8, full_record[curmatch].err_code);
space_ssn(viewssn, querymatches[curmatch].sp_ssno);
disp_string(detail, 9, 14, viewssn);
disp_string(detail, 9, 27, full_record[curmatch].sp_neseb);
disp_string(detail, 9, 33, full_record[curmatch].sp_nesef);
disp_string(detail, 9, 39, full_record[curmatch].sp_wages);
disp_string(detail, 9, 45, querymatches[curmatch].sp_sei);
disp_string(detail, 9, 52, full_record[curmatch].date);
disp_string(detail, 9, 64, full_record[curmatch].ein);
disp_string(detail, 9, 74, full_record[curmatch].ind_cd);

disp_string(detail, 14, 18, "ENTER=PRINT RECORD    ESC=BROWSE SCREEN");

/* Wait for user to hit ESC, indicating done */
input = wgetch(detail);
while(input != ESC)
   {
   if(input == ENTER)
      {
      inc_print_count(querymatches[curmatch].year, pminfo);
      print_rec(curmatch, full_record, querymatches);
      }
   input = wgetch(detail);
   }

/* Hide the detail box */
hide_panel(pdetail);
}



/*
 *
 * Print Selected Record
 *
 * Input:  current match number
 *         arrays holding the current set of matches
 *
 * Output: detailed information is sent to printer
 *
 *
 */

print_rec(curmatch, full_record, querymatches)
 struct REC full_record[];
 struct QUERY querymatches[];
 int curmatch;
 {
 /* Turn on the local printer (all output now goes to printer          */
 system("tput mc5");

 /* send the record to the printer                                      */
 printf("Name  :  %s\n", querymatches[curmatch].fullname);
 printf("Address: %s\n", full_record[curmatch].st_add);
 printf("         %s  %s\n\n",
        full_record[curmatch].city_st, full_record[curmatch].zip);

 printf("Name       SSN        NESEB NESEF WAGES SEI   YEARQU ST DLN/EIN IND CD\n");
 printf("-------------------------------------------------------------------------\n");
 printf("%-8s%-3s%-10s%-6s%-6s%-6s%-6s%-2s%-2s%-3s%s%s%-3s%-14s\n",
        querymatches[curmatch].lastname, querymatches[curmatch].initials,
        querymatches[curmatch].ssno, full_record[curmatch].neseb,
        full_record[curmatch].nesef, full_record[curmatch].wages,
        querymatches[curmatch].sei, querymatches[curmatch].year,
        full_record[curmatch].month, full_record[curmatch].se_quar,
        full_record[curmatch].option_cd,
        querymatches[curmatch].ird, full_record[curmatch].dln);

 printf("   %-3s%-3s%11s%6s %-6s%-6s%-6s%-10s%-10s%-5s\n\f",
        full_record[curmatch].iwp, full_record[curmatch].err_code,
        querymatches[curmatch].sp_ssno, full_record[curmatch].sp_neseb,
        full_record[curmatch].sp_nesef, full_record[curmatch].sp_wages,
        querymatches[curmatch].sp_sei, full_record[curmatch].date,
        full_record[curmatch].ein, full_record[curmatch].ind_cd);

 /* return the output to the screen                              */
 system("tput mc4");
 }



/*
 * restore screen to the browse view
 *
 * Input:  number of matches in memory
 *         current match number
 *         users query information
 *         arrays holding match information
 */

restore_scr(nummatch, curmatch, usrquery, querymatches, full_record)
  int nummatch, curmatch;
  struct INFO usrquery[];
  struct REC full_record[];
  struct QUERY querymatches[];
  {
  int pgmatch, rownum;

  /* Display the header information */
  view_header(usrquery);

  /* Display current page of matches */
  disp_screen(querymatches, nummatch);

  /* Move cursor to correct line on screen */
  rownum = (curmatch * 2) + 6;
  wmove(general, rownum, 2);
  }



/*********************************************************************/
/*                                                                   */
/*                       General Functions                           */
/*                                                                      */
/************************************************************************/



/*
 * Insert space in an social security number between the segments
 * of numbers (xxx xx xxxx), and store the new array in a temporary
 * buffer
 *
 * input:  buffer to place spaced ssn in
 *         ssno string to space
 *
 * Output: temp buffer is filled with spaced ssno
 *
 */

space_ssn(temp, ssno)
     char temp[];
     char ssno[];
     {
     int j;

     for(j = 0; j < 3; j++)
       temp[j] = ssno[j];
     temp[3] = ' ';
     for(j = 3; j < 5; j++)
       temp[j+1] = ssno[j];
     temp[6] = ' ';
     for(j = 5; j < 9; j++)
       temp[j+2] = ssno[j];
     temp[11] = 0;
     }



/*
 * Move to a specified location on the screen and output a character
 * or character string
 *
 * input:  window in which to display the string
 *         row and column at which to start the string
 *         string to display
 *
 * output: string is placed on the screen
 */

disp_string(win, row, col, str)
   int row, col;
   WINDOW *win;
   char str[];
   {
   wmove(win, row, col);
   waddstr(win, str);
   }




/*
 * Display an Error Message
 *
 * Input:  Message to display in the error box
 *
 * Output: The message is displayed
 *
 */

disp_err(msg)
   char msg[];
   {
   char input;

   /* Bring up the error panel and place the message in it                */
   show_panel(perrormsg);
   clrwin(ERRORMSG);
   disp_string(errormsg, 2, 10, msg);
   wrefresh(errormsg);

   /* wait until the user indicates that they saw the message             */
   input = wgetch(errormsg);
   while(input != ESC)
      input = wgetch(errormsg);

   /* hide error message and return to query entry                        */
   hide_panel(perrormsg);
   }



/*
 * Display a Wait Message
 *
 * Output: The wait message is displayed
 *
 */

disp_wait()
   {
   /* Bring up the error panel and place the message in it                */
   show_panel(perrormsg);
   clrwin(ERRORMSG);
   disp_string(errormsg, 2, 10, "PLEASE WAIT -- RETRIEVING DATA");
   wrefresh(errormsg);
   }


/*
 * Remove a Wait Message
 *
 * Output: The wait message is removed from the screen
 *
 */

rem_wait()
   {
   /* hide error message and return to query entry                        */
   hide_panel(perrormsg);
   }



/*
 * clears a window to end of line at the specified row
 *
 * Input:  window in which to clear the row
 *         row to clear
 *
 * Output: The specified row in the specified window is cleared
 */
```

```
clearline(win, row)
   WINDOW *win;
   int row;
   {
   int i;

   for(i = 0; i < 80; i++)
      disp_string(win, row, i, " ");
   box(win, 0, 0);
   }




/*
 * Display matches currently in memory
 *
 * input:  array containing browse information on matches
 *         number of matches in memory
 *
 * output: a screen of matches is displayed
 *
 */

disp_screen(querymatches, num)
   struct QUERY querymatches[];
   int num;
   {
   int i, row, col;
   char buffer[21];

   /* clear the screen of old matches                                   */
   for(i = 6; i < 21; i++)
     clearline(general, i);

   /* move to the top of the screen, just below the header             */
   wmove(general, 4,2);
   for(i = 0; i < num; i++)
     {
     /* go to next line                                                */
     getyx(general, row, col);
     row = row+2;

     /* add browse fields                                              */
     disp_string(general, row, 2, querymatches[i].lastname);
     disp_string(general, row, 10, querymatches[i].initials);
     space_ssn(buffer, querymatches[i].ssno);
     disp_string(general, row, 15, buffer);
     disp_string(general, row, 28, querymatches[i].sei);
     space_ssn(buffer, querymatches[i].sp_ssno);
     disp_string(general, row, 36, buffer);
     disp_string(general, row, 50, querymatches[i].sp_sei);

     /* display the first 20 characters of the full name              */
     wmove(general, row, 57);
     if( strlen(querymatches[i].fullname) > 20)
        {
        strncpy(buffer, querymatches[i].fullname, 20);
        buffer[20] = 0;
        waddstr(general, buffer);
        }
     else
        waddstr(general, querymatches[i].fullname);
     }
   /* Add user directional prompts                                    */
   disp_string(general, 23, 4,
   "ESC=QUERY SCR   ENTER=DETAIL SCR   PGUP/PGDN=SCROLL   END/HOME=SHIFT REC");

   /* Move to the first row and display the matches                   */
   wmove(general, 6, 2);
   wrefresh(general);
   }




/*
 * The next set of matches is loaded to memory, if it exists
 *
 * Input:  arrays to hold the matches
 *
 * Output: the next set of matches is stored in memory
 *
 */

int load_matches(querymatches, full_record)
    struct QUERY querymatches[];
    struct REC full_record[];
    {
    int i, j;

     /* Get a match from the file                                    */
     for(i = 0; i < MATCHSCR; i++)
       {
       $fetch next recptr into $retrvar.lastname, $retrvar.initials,
         $retrvar.ssno, $retrvar.neseb, $retrvar.nesef, $retrvar.wages,
         $retrvar.sei, $retrvar.year, $retrvar.month, $retrvar.se_quar,
         $retrvar.option_cd, $retrvar.ird, $retrvar.dln, $retrvar.full_name,
         $retrvar.iwp, $retrvar.err_code, $retrvar.sp_ssno, $retrvar.sp_neseb,
         $retrvar.sp_nesef, $retrvar.sp_wages, $retrvar.sp_sei,
         $retrvar.date_rcvd, $retrvar.ein, $retrvar.ind_cd, $retrvar.st_add,
         $retrvar.city_st, $retrvar.zip;

       /* If no matches exist, exit the loop                          */
       if(sqlca.sqlcode == SQLNOTFOUND)
          break;

       /* Copy the match from the host variable to the array          */
       stcopy(retrvar.lastname, querymatches[i].lastname);
       stcopy(retrvar.initials, querymatches[i].initials);
       stcopy(retrvar.ssno, querymatches[i].ssno);
       stcopy(retrvar.neseb, full_record[i].neseb);
       stcopy(retrvar.nesef, full_record[i].nesef);
       stcopy(retrvar.wages, full_record[i].wages);
       stcopy(retrvar.sei, querymatches[i].sei);
       stcopy(retrvar.year, querymatches[i].year);
       stcopy(retrvar.month, full_record[i].month);
       stcopy(retrvar.se_quar, full_record[i].se_quar);
       stcopy(retrvar.option_cd, full_record[i].option_cd);
       stcopy(retrvar.ird, querymatches[i].ird);
       stcopy(retrvar.dln, full_record[i].dln);
       stcopy(retrvar.full_name, querymatches[i].fullname);
       stcopy(retrvar.iwp, full_record[i].iwp);
       stcopy(retrvar.err_code, full_record[i].err_code);
       stcopy(retrvar.sp_ssno, querymatches[i].sp_ssno);
       stcopy(retrvar.sp_neseb, full_record[i].sp_neseb);
       stcopy(retrvar.sp_nesef, full_record[i].sp_nesef);
       stcopy(retrvar.sp_wages, full_record[i].sp_wages);
       stcopy(retrvar.sp_sei, querymatches[i].sp_sei);
```

```
      stcopy(retrvar.date_rcvd, full_record[i].date);
      stcopy(retrvar.ein, full_record[i].ein);
      stcopy(retrvar.ind_cd, full_record[i].ind_cd);
      stcopy(retrvar.st_add, full_record[i].st_add);
      stcopy(retrvar.city_st, full_record[i].city_st);
      stcopy(retrvar.zip, full_record[i].zip);
    }

  /* return the number of matches found                                    */
  return(i);
  }




/*
 * The previous set of matches is loaded to memory, if it exists
 *
 * Input:  arrays to hold the matches
 *
 * Output: the previous set of matches is stored in memory
 *
 */

int load_prev_matches(querymatches, full_record)
    struct QUERY querymatches[];
    struct REC full_record[];
    {
    int i, j;

     /* Get one match from the data file                                    */
     for(i = 1; i <= MATCHSCR; i++)
       {
       $fetch previous recptr into $retrvar.lastname, $retrvar.initials,
         $retrvar.ssno, $retrvar.neseb, $retrvar.nesef, $retrvar.wages,
         $retrvar.sei, $retrvar.year, $retrvar.month, $retrvar.se_quar,
         $retrvar.option_cd, $retrvar.ird, $retrvar.dln, $retrvar.full_name,
         $retrvar.iwp, $retrvar.err_code, $retrvar.sp_ssno, $retrvar.sp_neseb,
         $retrvar.sp_nesef, $retrvar.sp_wages, $retrvar.sp_sei,
         $retrvar.date_rcvd, $retrvar.ein, $retrvar.ind_cd, $retrvar.st_add,
         $retrvar.city_st, $retrvar.zip;

       /* If no matches exits, exit the loop                                */
       if(sqlca.sqlcode == SQLNOTFOUND)
          break;

       /* Copy the match from the host variable to the arrays               */
       stcopy(retrvar.lastname, querymatches[MATCHSCR - i].lastname);
       stcopy(retrvar.initials, querymatches[MATCHSCR - i].initials);
       stcopy(retrvar.ssno, querymatches[MATCHSCR - i].ssno);
       stcopy(retrvar.neseb, full_record[MATCHSCR - i].neseb);
       stcopy(retrvar.nesef, full_record[MATCHSCR - i].nesef);
       stcopy(retrvar.wages, full_record[MATCHSCR - i].wages);
       stcopy(retrvar.sei, querymatches[MATCHSCR - i].sei);
       stcopy(retrvar.year, querymatches[MATCHSCR - i].year);
       stcopy(retrvar.month, full_record[MATCHSCR - i].month);
       stcopy(retrvar.se_quar, full_record[MATCHSCR - i].se_quar);
       stcopy(retrvar.option_cd, full_record[MATCHSCR - i].option_cd);
       stcopy(retrvar.ird, querymatches[MATCHSCR - i].ird);
       stcopy(retrvar.dln, full_record[MATCHSCR - i].dln);
       stcopy(retrvar.full_name, querymatches[MATCHSCR - i].fullname);
       stcopy(retrvar.iwp, full_record[MATCHSCR - i].iwp);
       stcopy(retrvar.err_code, full_record[MATCHSCR - i].err_code);
       stcopy(retrvar.sp_ssno, querymatches[MATCHSCR - i].sp_ssno);
       stcopy(retrvar.sp_neseb, full_record[MATCHSCR - i].sp_neseb);
       stcopy(retrvar.sp_nesef, full_record[MATCHSCR - i].sp_nesef);
      stcopy(retrvar.sp_wages, full_record[MATCHSCR - i].sp_wages);
      stcopy(retrvar.sp_sei, querymatches[MATCHSCR - i].sp_sei);
      stcopy(retrvar.date_rcvd, full_record[MATCHSCR - i].date);
      stcopy(retrvar.ein, full_record[MATCHSCR - i].ein);
      stcopy(retrvar.ind_cd, full_record[MATCHSCR - i].ind_cd);
      stcopy(retrvar.st_add, full_record[MATCHSCR - i].st_add);
      stcopy(retrvar.city_st, full_record[MATCHSCR - i].city_st);
      stcopy(retrvar.zip, full_record[MATCHSCR - i].zip);
    }
  return(i - 1);
  }




/*********************************************************************/
/*                                                                   */
/*                       Windowing Functions                         */
/*                                                                   */
/*********************************************************************/

/*
 * Create a Window
 *
 * Input:  length of window
 *         width of window
 *         start point of window
 *
 * Output: window is created
 */

WINDOW *create(length, width, y_start, x_start)
  int length, width, y_start, x_start;
  {
  WINDOW *win;

  /* create the window */
  win = newwin(length, width, y_start, x_start);
  return(win);
  }



/*
 *
 * setup the windows to be used - create them and set their parameters
 *
 * input:  None (window and panel pointers are global)
 *
 * output: Windows are created and set to correct settings
 *
 */


setup_windows()
  {
  WINDOW *create();

  /* Create the background, detail, and error message windows */
  detail = create(16, 80, 5, 0);
  general = create(25, 80, 0, 0);
  errormsg = create(5, 80, 7, 0);
```

```
/* Generate a panel pointer for the windows and hide the */
/* detail and error message windows */
pgeneral = new_panel(general);
pdetail = new_panel(detail);
perrormsg = new_panel(errormsg);
hide_panel(pdetail);
hide_panel(perrormsg);

/* Set up the windows so that the characters are not echoed   */
/* onto the screen as they are typed, and the function keys   */
/* are returned as a single character, not an escape sequence */
noecho();
keypad(general, TRUE);
keypad(detail, TRUE);
keypad(errormsg, TRUE);

/* set the input to non buffering (does not wait for carriage  */
/* return to be available to the program)                      */
cbreak();

/* Do not flush the tty driver buffer when an interrupt key is  */
/* hit by the user.  The first argument (window) is ignored, so */
/* this is set up for all windows, not just "general"           */
intrflush(general, FALSE);

/* Make the cursor more visible (large flashing rectangle)      */
curs_set(2);

/* These functions should be added when debugging so that a     */
/* time out message is not received when ESC is hit.  However   */
/* they will conflict with receiving a plain ESC when running   */
/* so should not be used on a normal basis                      */
/*  notimeout(general, TRUE);
notimeout(errormsg, TRUE);
notimeout(detail, TRUE); */
}



/*
 * Clear the window
 *
 * Input: window to clear
 *
 * Output: window is cleared
 *
 */

clrwin(win)
  int win;
  {
  int i, j;

  /* if window is the background window */
  if(win == GENERALWIN)
    {
    chg_text(GENERALWIN);
    for(j = 0; j < 25; j++)
      for(i = 0; i < 80; i++)
        disp_string(general, j, i, " ");
    box(general, 0, 0);
    wrefresh(general);
    }
  /* if window is the detail window */
  else if(win == DETAILWIN)
    {
    chg_text(DETAILWIN);
    for(j = 0; j < 16; j++)
      for(i = 0; i < 80; i++)
        disp_string(detail, j, i, " ");
    box(detail, 0, 0);
    wrefresh(detail);
    }


  /* if window is the error message window */
  else if(win == ERRORMSG)
    {
    chg_text(ERRORMSG);
    for(j = 0; j < 5; j++)
      for(i = 0; i < 80; i++)
        disp_string(errormsg, j, i, " ");
    box(errormsg, 0, 0);
    wrefresh(errormsg);
    }
  }




/*
 * Change the color of the text in the current window
 *
 * input:  current window
 *
 */

chg_text(win)
 int win;
 {
 if(win == GENERALWIN)
    wattrset(general, A_NORMAL);
 else if(win == DETAILWIN)
    wattrset(detail, A_BOLD);
 else if(win == ERRORMSG)
    wattrset(errormsg, A_BOLD);
 }
```

```
/*
 * Program to create a database file from the indexing
 * information created to locate IRD files on an optical disk
 *
 * index.ec
 * by Natalie Willman
 *
 * April 2, 1991
 *
 */
/* Define statements */
#define TRUE  1
#define FALSE 0

/* Include Files  */
$include sqlca;
#include <stdio.h>
#include <string.h>

struct INDEXINFO
   {
   char first[8];
   char last[8];
   char file[10];
   char disk[3];
   char side[2];
   char juke[2];
   };


/*
 * Main Program
 *
 */

main()
  {
  struct INDEXINFO record, *precord;
  char filenm[30], tablename[15], ird[3], filelong[30];
  FILE *split, *fopen();
  char ch;


  /* Set up pointers */
  precord = &record;

  printf("Enter the IRD:  \n\n");
  scanf("%s%*c", ird);

  printf("Enter the Disk # the data is stored on:  \n\n");
  scanf("%s%*c", precord->disk);

  printf("Enter the Side of the disk:  \n\n");
  scanf("%s%*c", precord->side);

  printf("Enter the Jukebox the in which the disk will reside:  \n\n");
  scanf("%s%*c", precord->juke);

  /* Open the database */
  $ database eas;
  if(sqlca.sqlcode)
    printf("ERROR\n");


  $ create table i0188
```

```
      (
      fname  char(7),
      lname  char(7),
      filenm char(9),
      disk   char(2),
      side   char(1),
      juke   char(1)
      );

   $ create index i0188 on i0188 (fname, lname);
   strcpy(tablename, "i0188");

   strcpy(filenm, "startend.");
   strcat(filenm, ird);

   split = fopen(filenm, "r");
   ch = fscanf(split,"%s%s%s%*c",precord->first,precord->last,filelong);
   strcpy(precord->file, &filelong[8]);
   while(ch != EOF)
    {
    printf("Indexing file %s ...\n\n", precord->file);

    /* read in and add records to index table */
    add_rec(tablename, precord);

    ch = fscanf(split,"%s%s%s%*c",precord->first,precord->last,filelong);
    strcpy(precord->file, &filelong[8]);
    }
   }




 /*
  * Read in the records and convert
  * the data to .dbs format
  *
  */

 add_rec(tablename, precord)
   struct INDEXINFO *precord;
   char tablename[];
   {
   $char firstnm[8];
   $char lastnm[8];
   $char flnm[10];
   $char diskno[3];
   $char sideno[3];
   $char jukebx[3];

   /* Copy to a host variable - bug in Informix */
   stcopy(precord->first, firstnm);
   stcopy(precord->last, lastnm);
   stcopy(precord->file, flnm);
   stcopy(precord->disk, diskno);
   stcopy(precord->side, sideno);
   stcopy(precord->juke, jukebx);

   /* Insert record into database */
   $ INSERT INTO i0188 (fname, lname, filenm, disk, side, juke)
          VALUES ($firstnm, $lastnm, $flnm, $diskno, $sideno, $jukebx);

   if(sqlca.sqlcode)
     {
     printf("ERROR occured! - Number %d\n", sqlca.sqlcode);
  exit();
  }
}
```

```
/*
 * Program to kill message queues created by device handler
 *
 * killq.c
 * by Natalie Willman
 *
 * April 2, 1991
 *
 */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define KEY1 1234L
#define KEY2 5678L
#define KEY3 2345L

struct MSGBUF
  {
  long mtype;
  int user;
  int juke;
  int disk;
  int side;
  }

main()
 {
 int id, id2, id3;
 int res;

 id = msgget(KEY1, 0);
 id2 = msgget(KEY2, 0);
 id3 = msgget(KEY3, 0);

 res = msgctl(id, IPC_RMID, (struct MSGBUF *) 0);
 if(res < 0)
   {
   printf("could not remove queue 1\n");
   }

 res = msgctl(id2, IPC_RMID, (struct MSGBUF *) 0);
 if(res < 0)
   {
   printf("could not remove queue 2\n");
   }

 res = msgctl(id3, IPC_RMID, (struct MSGBUF *) 0);
 if(res < 0)
   {
   printf("could not remove queue 3\n");
   }
 }
```

```
/*
 *
 * Print_mi.c
 *
 * by Natalie Willman
 * June 13, 1991
 *
 * This program will retrieve management information files
 * created by the pilot user interface and generate
 * statistics of performance
 *
 */

#include <stdio.h>     /* standard input/output header file              */
#include <string.h>    /* String manipulation functions                  */


struct MANAGEINFO
  {
  long int logtime;
  long int logcount;

  long int qt88;
  long int qt89;

  long int qc88;
  long int qc89;

  long int dc88;
  long int dc89;

  long int pc88;
  long int pc89;
  } input;


struct MI
  {
  long int totlog;
  long int totallogs[5];

  long int totacctime;

  float avgacc;

  long int totqc;
  long int qc88;
  long int qc89;

  long int totqt;
  long int qt88;
  long int qt89;

  float totavgqt;
  float avgqt88;
  float avgqt89;

  long int totdc;
  long int dc88;
  long int dc89;

  long int totpc;
  long int pc88;
  long int pc89;
  } output;
```

```
/*
 * Main Program
 *
 */

main()
  {
  FILE *fopen(), *mi, *curfil;
  char filename[20];
  int i;

  for(i = 1; i < 5; i++)
    {
    /* open each of the station mi files */
    sprintf(filename, "%s%d", "mi_info.", i);
    curfil = fopen(filename, "r");
    if(curfil == NULL)
      {
      printf("File %s does not exist\n", filename);
      printf("Not able to generate Management Information\n");
      exit();
      }
    else
      {
      /* Read in statistics gathered by the user interface */
      fscanf(curfil, "%d%*c", &input.logtime);
      fscanf(curfil, "%d%*c", &input.logcount);
      fscanf(curfil, "%d%*c", &input.qt88);
      fscanf(curfil, "%d%*c", &input.qt89);
      fscanf(curfil, "%d%*c", &input.qc88);
      fscanf(curfil, "%d%*c", &input.qc89);
      fscanf(curfil, "%d%*c", &input.dc88);
      fscanf(curfil, "%d%*c", &input.dc89);
      fscanf(curfil, "%d%*c", &input.pc88);
      fscanf(curfil, "%d%*c", &input.pc89);
      fclose(curfil);

      /* Calculate total logins and login count by station */
      output.totlog = input.logcount + output.totlog;
      output.totallogs[i] =  input.logcount;

      /* Calculate total login time, and total query count */
      /* and query count for each year (88 and 89)         */
      output.totacctime = output.totacctime + input.logtime;
      output.totqc = output.totqc + input.qc88 + input.qc89;
      output.qc88 = input.qc88 + output.qc88;
      output.qc89 = input.qc89 + output.qc89;

      /* Calculate total query time, and query time for each year */
      output.totqt = output.totqt + input.qt88 + input.qt89;
      output.qt88 = output.qt88 + input.qt88;
      output.qt89 = output.qt89 + input.qt89;

      /* Calculate total detail count, and detail counts for each year */
      output.totdc = output.totdc + input.dc88 + input.dc89;
      output.dc88 = output.dc88 + input.dc88;
      output.dc89 = output.dc89 + input.dc89;

      /* Calculate total printout count, and printout counts for each year */
      output.totpc = output.totpc + input.pc88 + input.pc89;
      output.pc88 = output.pc88 + input.pc88;
      output.pc89 = output.pc89 + input.pc89;
      }
    }
```

```
/* Calculate averages of login time, total query time, and query */
/* time for each year */
output.avgacc = (float) output.totacctime/ (float) output.totlog;
output.totavgqt = (float) output.totqt/ (float) output.totqc;
output.avgqt88 = (float) output.qt88/ (float) output.qc88;
output.avgqt89 = (float) output.qt89/ (float) output.qc89;

/* Open output file and print statistics to it */
mi = fopen("mi_stats", "w");

fprintf(mi, "\n\n\n\n");
fprintf(mi, "\t\t\tManagement Information Printout\n\n\n");

fprintf(mi, "\t\tTotal Accesses to the System       = %10d\n", output.totlog);
for(i = 1; i < 5; i++)
   fprintf(mi, "\t\tTotal Accesses from Station %d      = %10d\n",i,output.totallogs[
fprintf(mi, "\n\n");
fprintf(mi, "\t\tTotal Access Time in minutes       = %10d\n", output.totacctime);
fprintf(mi, "\t\tAverage Access Time in minutes     = %10.3f\n\n\n", output.avgacc);

fprintf(mi, "\t\tTotal Queries                      = %10d\n", output.totqc);
fprintf(mi, "\t\tTotal Query Time in minutes        = %10d\n", output.totqt);
fprintf(mi, "\t\tAverage Query Time in minutes      = %10.3f\n\n\n", output.totavgqt)

fprintf(mi, "\t\tTotal 1988 queries                 = %10d\n", output.qc88);
fprintf(mi, "\t\tTotal 1988 query time in minutes   = %10d\n", output.qt88);
fprintf(mi, "\t\tAverage 1988 query time in minutes = %10.3f\n\n\n", output.avgqt88);

fprintf(mi, "\t\tTotal 1989 queries                 = %10d\n", output.qc89);
fprintf(mi, "\t\tTotal 1989 query time in minutes   = %10d\n", output.qt89);
fprintf(mi, "\t\tAverage 1988 query time in minutes = %10.3f\n\n\n", output.avgqt89);

fprintf(mi, "\t\tTotal detail records               = %10d\n", output.totdc);
fprintf(mi, "\t\tTotal 1988 detail records          = %10d\n", output.dc88);
fprintf(mi, "\t\tTotal 1989 detail records          = %10d\n\n\n", output.dc89);

fprintf(mi, "\t\tTotal printed records              = %10d\n", output.totpc);
fprintf(mi, "\t\tTotal 1988 printed records         = %10d\n", output.pc88);
fprintf(mi, "\t\tTotal 1989 printed records         = %10d\n\n\n", output.pc89);
fclose(mi);
}
```

```
/*
 *
 * sep.c
 *
 * by Natalie Willman
 *
 * This program dividies an EASEAR file into smaller files of
 * 300 records.  The records are logically seperated into files
 * by IRD, with a file only containing one IRD.  The program
 * also creates an indexing table that contains the starting
 * last name and initials and the ending last name and initials
 * of each file of records.
 *
 */

/* Include files */
#include <stdio.h>
#include <string.h>

/* Define statements */
#define TRUE  1
#define FALSE 0


/*
 *
 *  MAIN PROGRAM
 *
 */

main()
  {
  int i, j, count, filenum, irdint, searchint, previrdint;
  char filename[30], searchird[3];
  char filenm[20], output[15], filestr[10];
  char record[235], errfile[20];
  char ch, ird[3], last[10], init[4], previnit[4];
  char prevlast[10], resp, namefile[20];
  FILE *dat, *temp, *err, *name, *fopen();
  int rec, reccount, end, filect;
  int result, totalrec;



  printf("Enter the IRD for which to search:  \n\n");
  scanf("%s%*c", searchird);
  printf("\n");

  strcpy(resfile, "ird.");
  strcat(resfile, searchird);


  /* open the name file and print blank spaces for the initial last name */
  sprintf(namefile, "%s%s", "startend.", searchird);
  name = fopen(namefile, "w");
  fprintf(name, "              ");
  fclose(name);

  /* set up once only variables */
  reccount = 0;
  filect = 1;
  prevlast[0] = 0;
  previnit[0] = 0;


  do
```

```
{
printf("Make sure that the source disk is mounted as read-only\n");
printf("in Drive 2 and that the destination disk is mounted\n");
printf("as read/write in drive 1\n\n");

/* Retrieve Information on the files to analyze */
printf("Enter the Total number of files to search:  \n\n");
scanf("%d%*c", &count);
printf("\n\n");
printf("Enter the file number with which to start:  \n\n");
scanf("%d%*c", &filenum);
printf("\n\n");
sprintf(filename, "%s%d", "/drive2/", filenum);

/* Analyze each of the files */
end = filenum + count;
for(i = filenum; i < end; i++)
  {
  printf("Analyzing File %s ...\n", filename);

  /* Open the file to read */
  dat = fopen(filename, "r");

  /* Set the counter variables to 0 */
  rec = 0;
  totalrec = 0;
  previrdint = 0;

  ch = fgetc(dat);

  /* Read in each of the records in the file                       */
  do
    {
    /* setup first character */
    record[0] = ch;
    last[0] = ch;

    /* Read in the record - 234 characters long */
    for(j = 1; j < 234; j++)
      {
      record[j] = fgetc(dat);
      /* Test for error condition */
      if(record[j] == EOF)
         {
         sprintf(errfile, "%s%s", "err.", searchird);
         err = fopen(errfile, "a");
         fprintf(err,"Error on record %d, file %s\n",totalrec+1,filename);
         fclose(err);
         }
      ch = record[j];

      /* store the lastname, initials, and ird to seperate variables */
      if(j == 1)
         last[1] = ch;
      if(j == 2)
         last[2] = ch;
      if(j == 3)
         last[3] = ch;
      if(j == 4)
         last[4] = ch;
      if(j == 5)
         last[5] = ch;
      if(j == 6)
         {
         last[6] = ch;
         last[7] = 0;
      }

   if(j == 7)
      init[0] = ch;
   if(j == 8)
      {
      init[1] = ch;
      init[2] = 0;
      }

   if(j == 44)
      ird[0] = ch;
   if(j == 45)
      {
      ird[1] = ch;
      ird[2] = 0;
      }
   }

/* Add EOL character and increment the record count */
record[234] = 0;
totalrec++;

/* check to see that one of the IRD's is not EOF characters,   */
/* and that we are at an IRD later than or equal to our search */
/* ird                                                         */
irdint = atoi(ird);
searchint = atoi(searchird);
searchint--;
if( (ird[0] != EOF)  && (irdint > searchint) )
  {
  /* compare the current ird with the ird for which we are searching */
  result = strcmp(ird, searchird);
  /* if they are the same, increment record count.  If the record    */
  /* count is greater than 300 start a new file of records           */
  if(result == 0)
      {
      if(reccount == 300)
         {
         filect++;
         name = fopen(namefile, "a");
         fprintf(name, "%s %s   %s\n", prevlast, previnit, filenm);
         fprintf(name, "%s %s   ", last, init);
         fclose(name);
         reccount = 0;
         }
      reccount++;

      /* Open the output file and write the current record */
      sprintf(filestr, "%d", filect);
      sprintf(filenm, "/drive1/");
      strcat(filenm, ird);
      strcat(filenm, filestr);

      temp = fopen(filenm, "a");
      if(record[0] != EOF)
         fprintf(temp, "%s", record);
      fclose(temp);

      /* copy the current last name to the previous last name */
      strcpy(prevlast, last);
      strcpy(previnit, init);
      }
  else /* If the current ird is greater than the search ird, exit */
      {
      printf("Exiting on file %s, ird %s\n\n", filename, ird);
```

```
                name = fopen(namefile, "a");
                fprintf(name, "%s %s   %s\n", prevlast, previnit, filenm);
                fclose(name);
                fclose(dat);
                exit();
                }
            }
        /* If the IRD is less than our search IRD, skip over the record */
        else
            {
            if(irdint != previrdint)
               printf("skipping ird %d\n\n", irdint);
            }

        /* set the previous IRD integer variable equal to this IRD integer */
        previrdint = irdint;

        /* weed out EOB characters */
        if(rec == 135)
             {
             ch = fgetc(dat);
             rec = 0;
             }
        else
            rec++;

        /* Get next character and check it for EOF */
        ch = fgetc(dat);
        }
      while(ch != EOF);

      /* close the data file */
      fclose(dat);


      /* Create the next file name */
      sprintf(filename, "%s%d", "/drive2/", i+1);
      }

    /* ird may be split over two tapes, so may wish to switch */
    printf("Do you wish to switch tape disks? (y/n)\n\n");
    resp = getchar();
    getchar();

    /* open the name file and complete the last entry and start */
    /* a new one                                                */
    filect++;
    name = fopen(namefile, "a");
    fprintf(name, "%s %s   %s\n", last, init, filenm);
    fprintf(name, "%s %s   ", last, init);
    fclose(name);
    reccount = 0;
    }
  while( (resp == 'y') || (resp == 'Y') );
  }
```

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4654

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
AUGUST 1991

4. TITLE AND SUBTITLE
Development of an Optical Disk System for the Automated Retrieval of EASEAR Records

5. AUTHOR(S)
Natalie Willman

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)
U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED
Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)
Social Security Administration
6401 Security Boulevard
Baltimore, MD 21235

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

The Social Security Administration (SSA) maintains records of the yearly wages earned by every person in the United States. Each year, approximately 2.5 gigabytes of data are collected on self-employed wage earners, and 47 gigabytes of data are collected on other wage earners. The records are currently stored on over 110,000 rolls of microfilm. It takes over 400 scouts to retrieve information needed by the professional staff. Often, the needed roll of film is missing from the file due to being currently in use, misfiled, or misplaced. In addition, the information obtained from the microfilm is not always the most current, and decisions are sometimes based on obsolete data. Advances in peripheral mass storage technology during the 1980s (e.g. magneto-optic recording) now allow for alternate approaches to data storage and retrieval. It is believed that an automated retrieval system would provide a more accurate, timely, and cost effective means of retrieving information.

The pilot application detailed in this document is intended to demonstrate the feasibility of a full scale automated records retrieval system, to study the impact of an automated system on the users, and to gain insight into the factors which would impact the design of a full scale system.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

automated retrieval system; magneto-optic media; optical disk jukebox; optical disk media; pilot system

13. AVAILABILITY

X	UNLIMITED
	FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
	ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
x	ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
147

15. PRICE
A07

ELECTRONIC FORM