K/DSRD--80

DE89 014831

K/DSRD-80

APPROACHING DISTRIBUTED DATABASE APPLICATIONS USING A PROGRAMMABLE TERMINAL EMULATOR

J. A. Clinard and J. J. Robinson
Engineering Technology Division of
Oak Ridge National Laboratory

J. T. Phillips, Jr.
Data Systems Engineering Organization of
Data Systems Research and Development

and
G. L. Johnson
A. G. Technical Associates

June 1989

Research sponsored by
Pacific Missile Testing Center
Point Mugu, California 90342
Under Interagency Agreement DOE 1714-1714-A1

Oak Ridge Gaseous Diffusion Plant
Oak Ridge, Tennessee 37831-7129
operated by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
Under Contract No. DE-AC05-84OR21400

CONTENTS

ABSTRACT

Two separate approaches were used to develop a prototype for entering data into a remote host computer in an automated manner. In the first approach, revisions were implemented in the IBM PC's* terminal emulator VTEK 4.2**. These revisions allowed pre-written script files to be processed to the host-based operating system and applications software as if the script file information had been entered on the keyboard. The script processing capability was implemented taking advantage of existing user-defined key capability and the DEC VT100*** asynchronous terminal emulation of VTEK communications software.

At present the script command and data files must be manually created at the PC using an editor or word processor. The script processing capability works with any host-based operating system or application software that interacts with a DEC VT100 terminal. An example is provided where VTEK script processing is used to automatically interact with a VAX***-based Database Management System (DBMS), INGRES,**** appending PC-resident data records to an INGRES table, creating a default report, aborting to VMS,*** and disposing of the report ... all without touching a key.

An appendix is provided which discusses the second approach of developing a prototype VT100 emulator specifically designed for data entry to a remote host computer system. This software loads data automatically into a Vax Datatrieve database. It provides an alternative method of prototype development.

The challenges for future development are identified and discussed. The use of the programmable terminal emulator for data control in the case of distributed database applications is also discussed.

---

*IBM PC is a trademark of IBM Corporation.

**VTEK is a trademark of Scientific Endeavors Corporation.

***DEC, VAX, VMS, and VT100 are trademarks of Digital Equipment Corporation.

****INGRES is a trademark of Relational Technology, Inc.

1. INTRODUCTION

A common computing need of organizations/projects is that of processing data that is distributed across heterogeneous computing environments.[1] Solutions are being sought which would allow different vendors' hardware and software to work together harmoniously and transparently, without reprogramming or loss of investment in existing databases and/or file structures. This current project at the Pacific Missile Test Center (PMTC) was undertaken for the Information Resources Management Office (IRMO) in order to investigate two possible approaches to solving these problems.

The obstacles to overcoming this common computing need are often tied to lack of mature standards in computing technology, competition for market share among hardware/platform and software vendors, the highly developmental and complex nature of computing technology, the U.S. government's ADP* procurement regulations, politics existing within organizations, and presently inherent in many organizations to evolve systems and database designs once specified and developed. At present there is no single vendor product or collection of products that allows global solutions to such distributed database problems. For example, some limited successes have been recently reported.[2,3]

There are many developments which offer this promise of building global solutions, though it may be 2 to 5 years before their impact will be widely felt. Such developments include the following:

- Advancements in commercial distributed Relational Database Management System (RDBMS) software (e.g., INGRES and ORACLE**) that enable applications to access data from various vendors' databases and file management systems; distributed data managers that allow access to multiple databases simultaneously (even if on separate computers); and products that allow transparent access to remote computer systems connected through networking,

- Advances in Fourth Generation Languages (4GLs) such as improved query optimizers,

- Acceptance of Structured Query Language (SQL) as a data definition and manipulation standard (at the base of most 4GLs),

- Development of analysis and design support software such as Computer-Aided Software Engineering (CASE) tools,

- Development and refinement of configuration management software,

---

*All acronyms are defined in the Glossary.

**ORACLE is a trademark of Oracle Corporation.

- Integration of DBMS, CASE, and configuration management software into consistent single-product combinations,

- Improvements in data dictionaries and the development of international standards for dictionary systems,

- Development and utilization of advanced communication protocol and networking standards; examples include Ethernet, TCP/IP, GOSIP, DECnet*, SNA,** and NFS,***

- Development of artificial intelligence (expert system) links to DBMSs such as in query optimization, natural language query tools, reality checking (Quality Assurance) to avoid erroneous data entry, and intelligent interfaces to overcome incompatibilities in data forms/formats eliminating the need to recode data, among others,

- Increasing computing speeds and decreasing cost for all classes of platforms, from microcomputers to mainframes,

- Development of advanced microcomputers that allow multitask operating systems such as OS/2** or UNIX****, and workstations allowing multiple DOS***** sessions under UNIX,

- The broader acceptance of UNIX as an OS standard, and

- Additional successful developments in computing standards possibly due to vendor cooperation (OSF and X/Open) and/or to government insistence (CALS and OSI).

There are several additional developments which are likely to drive more DBMS applications to a distributed environment. Among these are the following:

- Implementation of available networking-communications technology such as the widespread use of broadband fiber optics for connectivity,

- The influx of microcomputer database technology,

- The further development of economical mass storage for microcomputers, such as optical disks,

---

*DECnet is a trademark of Digital Equipment Corporation.

**SNA and OS/2 are trademarks of IBM Corporation.

***NFS is a trademark of Sun Microsystems, Inc.

****UNIX is a trademark of AT&T Corporation.

*****DOS is a trademark of Microsoft, Inc.

- Proliferation of inexpensive "UNIX workstations" from vendors such as SUN, DEC, and IBM, among many others,

- Successful developments in computing technology standards, particularly the POSIX and X Windows standards, and

- Proliferation of Local Area Networks (LANs), particularly those composed of micro/minicomputer UNIX workstations, and the further development of networking technology to produce general acceptance/utilization of Wide Area Networks (WANs).

Generic solutions to distributed database problems may or may not evolve in the near term; no individual or organization can forecast the database future with much confidence. Meanwhile certain specific distributed database needs are being addressed by vendors, systems developers, and systems integrators who are succeeding, to various degrees, in grounding the developing technology in real-world requirements and constraints. One example is the current PMTC IRMO task to support development and demonstration of a virtual-data-entry terminal emulator which is a building block to a broader implementation of distributed database technology.

The reason such a primitive tool, as the one developed in our first approach, is important is its immediate capability to penetrate the heterogeneous computing environment allowing entry into "closed" DBMSs without vendor involvement other than the "universal" compliance with three very basic concepts or standards: (1) interactive terminal-to-host database transaction processing, (2) VT100 terminal service, and (3) asynchronous communication/networking support. The current efforts demonstrate the concept of controlling heterogeneous database environments using a "foreign" coordinating node and form the beginnings of a possible expandable distributed database solution.

The product of this effort is a microprocessor-based terminal emulator computer program that contains a powerful script language allowing "virtual" interactive terminal-to-host sessions. The sessions are virtual because the script conducts them interactively as though there were a terminal operator interacting with the application program, yet no terminal operator is needed to read the incoming screens or to strike the appropriate keyboard responses to achieve transaction processing. To accomplish the virtual session a script command file is prepared on the microprocessor that resembles a series of instructions of the form "wait for a certain host-supplied prompt, then supply the prearranged response." The command file must be constructed to match an exact sequence of desired interactions with the host-based application program. The prearranged responses may be supplied in either of two ways, as syntactically appropriate: first, they may be in-line-coded in the command file; or second, they may pre-exist as records in a microprocessor data file and simply be pointed to (read and supplied) by the script.

A second approach to the PMTC IRMO task was to develop a prototype VT100 emulator designed specifically for virtual data entry. In its current state, it has the ability to automatically load data into a VAX Datatrieve database but could be expanded to include additional computers and DBMS's. It is discussed in Appendix C.

The remainder of this report is divided into five sections and five appendices. Section 2 will examine the various forms that distributed databases might have and suggest an architecture which would be desirable for allowing successful distributed DBMS applications. Section 3 fully describes the script processing capabilities of the terminal emulator computer program discussed

above. Section 4 discusses limitations of the current approach and suggests improvements and extensions. Section 5 discusses other future developments that could eventually produce very powerful distributed database coordinations. Section 6 contains the references. In Appendix A, the terminal emulator software VTEK used in this study is fully described. An example script-controlled data entry session is presented in Appendix B along with other example script command files for various terminal-to-host sessions. Appendix C discusses a prototype VT100 emulator specifically designed for data entry. Appendices D and E contain the complete source code listings. Finally, a glossary of terms is provided for reference.

2. DISTRIBUTED DATABASE ARCHITECTURES

Distributed database technology is a comparatively recent development within the overall database field. The term distributed database has no common definition, but, typically, is a database that is not stored in its entirety at a single physical location, but rather is spread across a network of computers that are geographically dispersed and connected via communication links.[4]

There are at least two distributed architectures. The first may be called central processing where dispersed users ship their local database to a central node/computer for processing. Under this architecture all centralized data is available for retrieval by any user, and the combined data give a more informative report than one available on the local node only. Each local database is always up-to-date; the central node is periodically updated. The second may be called a partitioned database with zero overlap where data tables and files are scattered in various locations/nodes connected by a network. Data are linked using appropriate data keys. Applications for the second type of architecture can be run on any node; furthermore, applications may call for information from any of the locations, without having to write explicit instructions that describe data residence.

The current efforts of this PMTC IRMO project are aimed at helping automate an architecture like the first described above. The "shipping" process is managed by a script running on a PC which moves data from the PC to the central computer by virtual data entry. The architecture which has the most desirability and which appears to be evolving in the commercial sector is more like the second described above. The decentralization process is being pushed along by many technological developments such as those described in Section 1: Introduction.

More desirable attributes of a distributed database or DBMS include the following:

1. It allows users to connect dispersed data into a single unified information source.
2. It transparently ties together data across dispersed systems enabling users to cross-reference data dynamically.
3. It eliminates the need to modify existing applications running on different platforms (Central Processing Units or CPUs) and extends use of all system resources.
4. It provides full read and update support along with replication and partitioning.
5. It isolates the user and the database application from communication activities.
6. It allows strategic balancing of process workloads across multiple systems.
7. It allows database segmentation and storage on multiple systems.
8. It allows data with high availability requirements to be replicated in mirror-image copies at multiple locations and ensures that replicas are synchronized to produce data integrity.
9. It allows for backup-critical functions even if all but one of the database copies becomes inoperable.
10. It allows monitoring of activity profiles and revamping of storage strategies as activity suggests.

This list, while possibly of great importance and desirability, describes a situation that cannot be achieved presently for heterogeneous computing environments. At present, it is

theoretically possible to achieve such a situation but only if homogeneous software and hardware are involved in the distributed database system. The requirements for such a system possibly could be established through a formal life-cycle management approach that would result in a single-vendor distributed RDBMS software procurement. All distributed CPUs would also have to be networked through common communications software such as TCP/IP or SNA. All CPUs would require multitasking Operating Systems (OSs) that supported the RDBMS and the network communications.

Such solutions for the RDBMS include INGRES and ORACLE, for examples, each with a SQL star-distributed architecture as the enabling technology. DEC/VAX computers are a possible hardware choice.[5] One admitted drawback is the proliferation of processes that is necessary to coordinate the various nodal database activities resulting in the requirement of fast multiprocessing CPUs with large memories. If need were establishable and funds available, it should be possible to purchase a distributed RDBMS solution from such commercial vendors.

3. VTEK'S SCRIPT PROCESSING CAPABILITIES

A script processing capability was added to the PC terminal emulator VTEK. A series of script commands can be prepared on the PC which amount to a program for the PC to follow as if it were emulating a terminal to a host-based application. The script essentially eliminates the need for a terminal operator and can be used to achieve virtual data entry to the host. Data on the PC can be entered into the host database in this fashion.

The script files are PC resident and are either command files or data files. The command files automate the interactive terminal-to-host process as though all information and data were keyboard entered through a DEC VT100 terminal. At present the script files must be created with the PC's word processor or editor. This preparation is quite straightforward with the possible exception of the necessity to insert control characters in the script data file.

It should be clearly noted that VTEK-beta (in its current development status) can be programmed to participate in any well-defined interactive session. Script command files can be written for any session where the prompts/questions/error-conditions from the host's OS or applications software are a priori known; then legitimate responses can be included in the script. Error processing, though theoretically possible, remains an exercise for follow-on activities.

To aid in the preparation of a script file, advantage can be taken of the VTEK **Alt-w** command which captures all VT100 mode characters from the host into a designated PC dump file. Upon disconnection from a host, the file can be replayed using the **Ctrl-F3** command and can be echoed to the printer for further study. Using this method an actual interactive session can be captured, studied, and then manually modeled into an equivalent script.

3.1 SCRIPT COMMAND SYNTAX

The precise script command syntax for the current VTEK-beta is given below. There are but 18 script commands necessary for the complete script language. In the following definitions (and in the key macros), the "^" is the caret character, ASCII 94. "STRING" represents a string of characters of length less than 100 characters. The lower 128 ASCII (7-bit) characters are acceptable with control based on escape sequences not currently accommodated. The "##" represents a string of decimal integers, one per # sign. "FILENAME" is a data file obeying DOS naming conventions. The definitions are given below without great detail; many are further illustrated in the examples of Appendix B.

^(A-Z)	=	send a control character where uppercase A-Z represents the desired character
^!COMMENT^!	=	embed comment in script; skip to end of comment
^###	=	delay of ### hundredths of a second; this most nonspecific type of delay may be necessary to avoid buffer overflow such as during script data file send operations
^=##	=	wait for at least ## characters to be received from the host (on-line only)
^b	=	beep (such as for an error message)
^d##STRING^d	=	delay until pause, matching STRING in last ## chars (online only)

^e^label	=	extract data from data file and send to host till **Ctrl-A** or **Ctrl-B** is encountered. Send data to host. If **Ctrl-B**, close data file and jump to label (online only)
^fFILENAME^f	=	open data file (online only) (CRs in file are "^n")
^g(a-z)	=	goto label (26 labels allowed)
^k%	=	send VT100 keypad key (online only) % is given by: 0-9 . , - e p1 p2 p3 p4 u d l r - for Enter, PF1-PF4 keys, and the up, down, left, and right arrows, respectively
^l(a-z)	=	label
^mSTRING^TIME^SUCCESS_LABEL^m	=	match the string. Parse the input stream for TIME seconds. If the string is matched, flush the input buffer and go to SUCCESS_LABEL. If timeout or failure, go to the next step
^n	=	send a carriage return
^p....^p	=	send enclosed characters to the PC's printer
^q	=	quit and end script macro
^s##STRING^label	=	match STRING with last ## chars received (at a halt). If match, goto label (a-z). Else parse next character.
^v....^v	=	execute VTEK command (or key macro) till a second ^v
^w%	=	wait for a character(%) from the host (online only)

only keys valid in a ^v line:

^aF#	=	Alt-F# key
^cF#	=	Ctrl-F# key
^sF#	=	Shift-F# key
^upa	=	up arrow
^dna	=	down arrow
^lfta	=	left arrow
^m#	=	use VTEK communications selection # (= 1,2 or 3)
^n	=	send carriage return to VTEK
^rgta	=	right arrow
^t###	=	wait ### minutes before proceeding
^a(a-z)	=	alt key (see partial list below)
^oSTRING^o	=	output a string to the terminal (i.e., a user prompt)
^u##STRING^label##STRING^label.....^u	=	accept user input, compare ## chars. If matched, go to label (a-z)
^w	=	toggle CtrlPrtScr printer echo

Since the script processor can be used to execute most of the pre-programmed VTEK commands (the ALT keys), a partial list of those commands of likely utility for scripting is

reviewed. Very brief descriptions are provided here; reference to the VTEK User's Manual[6] or the help utility of VTEK will supply more detailed information. The VTEK commands of likely utility are as follows:

Alt-b send short BREAK, such as used by some computer systems to interrupt a host operation or stop the host from sending,

Alt-d perform DOS utilities, such as COPY, DEL, RENAME, etc., and run other PC programs without exiting from VTEK,

Alt-i lock keyboard while in use, such as might be desired to protect you from wrecking a job with inadvertent key pushes and/or to prevent unauthorized access to the PC during a prolonged script,

Alt-q send long BREAK which when used with properly configured modems causes both the local and remote data sets to disconnect,

Alt-u upload ASCII file to host, which might be useful to send a stream of ASCII characters into a host program such as an editor program,

Alt-w output to PC disk file, which might be useful for capturing parts/all of a script session for later study, particularly in testing/development phases,

Alt-x exit, close all files, exit from VTEK, turn off the serial port, and return to DOS, and

Alt-z Kermit and XMODEM protocols, which will usually be necessary to give access to file transfer protocols.

Ctrl-PrtSc printer echo, which will cause all ASCII traffic to be echoed to the PC printer until **Ctrl-PrtSc** is entered a second time.

These **Alt**-key commands are abbreviated for placement in the script file between "^v" (caret-v) characters. For example, **Alt-u** should appear as "**^v^au^v**".

3.2 SCRIPT DATA FILES

The script data file, while not required, may prove quite useful for multiple record entry; it is likewise PC resident. It can be of any length containing as many records as desired. The data file contains ASCII characters/keystrokes preformatted with **Ctrl-A** as end-of-record and **Ctrl-B** as end-of-file. Many available PC editors allow insertion of control characters; any one may be used to construct appropriate data files for scripting.

The best method to convey details of the data file is simply to illustrate with an example. Below is the simple script data file employed in the first example of Appendix B.

```
John Doe^A          1st record of "John", "tab", "Doe" plus Ctrl-A
Betty Smith^A       2nd record ending in Ctrl-A
Alan Bridges^A      3rd record ending with Ctrl-A
Lloyd Bridges^A     4th record ending with Ctrl-A
James Jones^B       5th and last record ending with Ctrl-B
```

In this example, five names (one per record) are separated by **Ctrl-A**. **Ctrl-B** follows the last name signifying end-of-file to the script processor. The likely method to employ such multiple-record PC files is through a script "LOOP", using the goto command, ^g(a-z), with record extraction and entry to the host through the extract-and-send command, ^e^lab. It should be

clear that the data file contains actual control characters, not implied characters. This PC script data file has the name "NAMES.DAT" as is seen in the first example of Appendix B. It is opened for reading with the command, ^fNAMES.DAT^f.

4. IMPROVEMENTS TO VTEK'S SCRIPT PROCESSING

Discussion in this section is limited to possible extensions and/or improvements to the script processing. Possible uses for the script capability range from simple to complex. The examples provided in Appendix B, though only moderately complex, illustrate much of the power of the script. The dominant script-controlled tasks coming to mind when one considers the distributed database problem are transfer of data among host computers using the PC as intermediary, and periodic updating of host-resident data tables from more current information maintained on the PC or in LANs (virtual data entry). Both are illustrated in the Appendix B examples.

Two desired major improvements to the VTEK script as it stands are in the areas of error handling and interfacing. Scripting will currently fail upon occurrence of any error for which an appropriate response has not been programmed. Explicit processing of errors is possible without VTEK modification. For example, one could write a script that handled error conditions from Kermit because these are well known and because legitimate responses are known/understood and can be programmed for all conditions. The Kermit example relies on deductive logic only. It would be possible through the extension of VTEK's capabilities to include inference measures (inductive logic) such as in the case of an expert system. This is considered a challenging exercise given the limitation of current PC hardware and the DOS operating system. (It is recommended that such a development be delayed until the PC operating system OS/2 is more mature.)

At present VTEK script data file format is rigid as described in Sect. 3.2. To allow PC application file formats such as from LOTUS* or dBASE** to be used, either a translation program that runs as a preprocessor to VTEK must be developed or expanded script capabilities for understanding additional formats must be developed.

It is not clear how to drastically improve the interface for preparation of the script command file; perhaps preparation through a "keystrokes remembered" capability as in SYMPHONY* could be added to VTEK. However, it appears likely that some need for manual preparation will be necessary to accomplish any complex interactive session. Because it is anticipated that scripts will have their greatest utility for repetitive data entry tasks, once the command file is prepared and debugged, it will require no maintenance until the entry task itself is altered.

In the introduction an explanation was offered for the choices of asynchronous communications and VT100 emulation for the current exercise. However, there are many other choices available that could be implemented using the PC. Script capabilities are a distinct portion of the terminal emulation software which must be added in all cases regardless of the communication protocol and/or type of terminal being emulated.

The asynchronous protocol was the obvious starting point because it is the only method to accomplish PC communications using the standard COM ports. (VTEK allows all available asynchronous flexibility by providing port selection, speed and parity setting, flow control, full

---

*LOTUS and SYMPHONY are trademarks of Lotus Development Corporation.

**dBASE is a trademark of Ashton-Tate.

duplex and half duplex, etc.) While asynchronous communication is the only protocol currently addressed by VTEK, the PC, with hardware extensions (cards), can be made to communicate using synchronous nodes such as SNA/SDLC, for example. And certainly the PC, again with added hardware, can function in an Ethernet LAN running TCP/IP, if desired. VTEK could be modified to include software drivers for the various protocols of communication. Concerning terminal emulation, the VT100 emulation (ANSI compatible) is a very broadly employed standard which is an acceptable terminal definition for most host applications, including the host OS itself. Most commercial software (applications, run-time libraries of the OS, utilities, screen management systems software such as DEC/ACMS, etc.) automatically handles the VT100's screen operations such as cursor movement and scrolling regions at a very high software layer. But, VT100 is not the only terminal type with broad applications. Another equally popular terminal, particularly for DBMS applications, is the IBM 3270-type terminal. Though it is certainly possible to modify VTEK to add 3270 emulation, it might be wiser to start with an existing successful emulator and then add script capabilities on top of the emulation, if desired. Lastly, it should be noted that the 3270 terminal is almost always host-supported in a synchronous communication mode.

5. FUTURE DISTRIBUTED DATABASE SOLUTIONS

As mentioned in the introduction, distributed RDBMSs are evolving with no mature system identifiable at present. The future success of expanding distributed RDBMSs to heterogeneous environments will probably hinge on various vendors' selection to introduce links to open systems[7] through compliance in OS and applications software with X Windows and SQL standards as an example workable approach. The vendors seem to be selecting the OSI communications standards (TCP/IP, then GOSIP) without much debate; thus, the networking selection is likely not a roadblock. Also, network file services such as NFS from SUN Microsystems are being broadly accepted to allow remote directory and file access on a network of workstations. Hopefully, consortiums such as X/Open and OSF will lead to the open systems that encourage joint accomplishments without severely impeding the speed of the necessary developments. The U.S. commercial sector offers the best chance for successful evolution of a general distributed RDBMS solution. Time will tell.

A further explanation concerning the importance of X Windows and SQL is offered. X Windows, often called simply "X", was developed by MIT Project Athena with the participation of DEC and other workstation vendors. X was developed with portability in mind, and is both hardware-independent and network-transparent. If applications programs such as RDBMSs abided by the X Windows standard, then a windowed environment would be possible that is independent of the operating system, independent of hardware constraints, and transparent to the networks, thus allowing the application to be used both locally and remotely. This is the essence of the star architecture of distributed RDBMSs. If the various vendors of database software abided by X, then a giant step toward the heterogeneous distributed solution would result. Coordinating nodes running under one vendor's software would be able to establish back-end processes on another vendor's software.

The importance of SQL is even more apparent. Processes written in SQL potentially run on all vendors' RDBMSs, and thus SQL is the universal language of the distributed system. The present differences in the versions of SQL of the various vendors offer a problem. It is possible that differences will be resolved due to vendor cooperation or through strict adherence to a mature SQL standard sometime in the future. Another solution would be the emergence of a natural language query tool using Artificial Intelligence (AI) to provide transparent access to heterogeneous SQLs.

One downside of the heterogeneous multi-vendor solution is the likely inability to interact across the distributed system's nodes at the dictionary layers of the various RDBMSs, which will limit the effectiveness of applications design tools such as CASE. It seems clear that very broad software compliances, those reaching the dictionary and design layers, will come to the software market much later than standards such as X, SQL, and OSI.

On the positive side, X is fast becoming the de facto window-management standard, particularly for UNIX workstations. X is also gaining support on a growing number of operating systems. For example, efforts are under way to provide X support on VMS and MS-DOS.

X Windows would be useful in expanding the current terminal emulation activities to produce a "communications platform." If the PC were replaced by a UNIX workstation in a TCP/IP network, for example, then it would be possible to multitask communications using different windows of the workstation as virtual terminals to mainframes or to other workstations

of the network. Scripts could "orchestrate" multiple simultaneous virtual data entry sessions and/or pipe database files to appropriate destinations in the distributed system. Such a mode for distributed database control would still require manual preparation of scripts and data files and would interact across the DBMSs only at high-level applications layers and/or at communications layers.

6. REFERENCES

1. E. H. Sibley, ed., ACM Computing Surveys: Special Issue: Data-Base Management Systems, 8(1), (March 1976).

2. B. Y. Forman and J. Fegreus, "Populating INGRES' Universe," Digital Review 4(13), 29-30 (June 29, 1987).

3. N. Margolis, "INGRES Database Extends Reach into RMS, dBase," Digital Review 4(21), 1-6, (November 9, 1987).

4. C. J. Date, An Introduction to Database Systems, Volume 1, Addison-Wesley Publishing Co., Reding, Massachusetts, ISBN 0-201-14471-9 (1982).

5. K. L. Kannan, Evaluation of Relational Database Products for the VAX, ORNL/TM-9696 (November 1985).

6. Anon., VTEK 4.2 - DEC VT102/VT100/VT52 and Tektronix 4105/4010/4014 Terminal Emulation, Scientific Endeavors Corp., Kingston, TN (September 1988).

7. A. S. Tanenbaum and R. V. Renesse, "Distributed Operating Systems," ACM Computing Surveys, 17(4), 419-70 (December 1985).

APPENDIX A:

EXISTING FEATURES OF VTEK

VTEK was selected as the software vehicle for this exercise because of its existing wide range of features and because a site license for the commercially-vended product can be obtained for a relatively modest fee which allows the licensing organization access to the C-based source code and rights to generate virtually unlimited copies of the executables. VTEK source was modified in the current exercise under an existing site license to produce the preliminary script processing capability. The resulting "beta" version is available for demonstration and study purposes. The remainder of Appendix A describes features of VTEK, many of which are not important to the current exercise.

VTEK allows the IBM PC or clone (such as Zenith Z-248) to emulate DEC VT100, VT102, or VT52 terminals as well as Tektronix 4105, 4010, or 4014 terminals. Within the hardware limitations of the PC, all interactive features of these terminals have been supported.

VTEK is especially suited for high resolution graphics found on many modern graphics cards. For example, the 640x350 resolution of the IBM Enhanced Graphics Adapter is efficiently supported. In addition, pictures are rerasterized to yield the highest resolution available on selected printers and plotters.

The memory required by VTEK depends upon setup options and particular hardware. For most systems, 256 Kb of memory is needed. Saving the graphics screen can use up to 128 Kb of extra memory. Color printer dumps also require an additional 256 Kb of memory. Since memory is dynamically allocated and de-allocated as it is required, the amount of memory in use will vary as a function of time. Two floppy disk drives can be used, but a hard disk is better.

In Tektronix 4010/4014 emulation mode, VTEK will correctly plot the usual 1024x780 pixel resolution and the super high 4096x3133 pixel Tektronix formats.

Tektronix 4105 emulation allows full color plotting by VTEK on high resolution 16 color graphics boards. Color output is available on pen plotters, ink-jet printers, and some dot matrix printers. A VT100 text window on the graphics screen is supported.

Twenty keys (**Alt-F1** through **Alt-F10** and **Shift-F1** through **Shift-F10**), may be user-defined. They may be programmed to send long strings, such as logon sequences and command strings, to the computer. These keys may be redefined at any time, even when connected to the host computer. User-defined keys are callable from within the script command file, and script command files are callable from user-defined keys.

VTEK offers password protection if desired. If this feature is activated, a password will be required to run VTEK. Also, it is possible to lock the keyboard, preventing unwanted key pushes or terminal access. The key definition file is encoded so that confidential access codes are protected. VTEK allows files to be downloaded from the host to the PC, and uploaded from the PC to the host. There are three methods of doing this. If the host computer supports the XMODEM or Kermit protocols, VTEK can be used to send and receive ASCII and binary files with error checking. Otherwise, ASCII files can be transferred without error checking if there is an editor on the host computer that will accept a character stream with carriage returns at the end of each line.

Both full and half duplex communication protocols are supported by VTEK. The COM1 or COM2 ports can be used.

For speed, VTEK uses only integer arithmetic in its terminal emulation modes. However, in zoom mode VTEK does use floating point arithmetic. VTEK will automatically detect the presence of a numeric coprocessor and use it.

VTEK has implemented all of the numeric keypad and PF keys of the VT100. The new enhanced keyboard is supported using the keys on the numeric keypad together with the **Home**, **End**, **PgUp**, and **PgDn** keys for **PF1-PF4**. On regular keyboards, the **F1-F10** and **Alt-1-Alt-8** keys have been used. The correct mapping is displayed on the screen by entering **Ctrl-F1**. A second help screen (**Ctrl-F2**) can be edited to match a particular application program or host editor.

The VT102 printer control commands have been implemented. They allow the host to control the PC-based printer directly. The host can turn printer echo on and off, print a screen, send a file to the printer without echoing to the screen (saves time), and send escape sequences to the printer.

APPENDIX B:

EXAMPLE VTEK EMULATOR EXERCISES

Five example exercises are provided. The first is extensive and illustrates virtual data entry from a PC-resident script data file. The remaining four example exercises are simpler script command sequences that illustrate the variety of activities possible with the script language.

In the example files provided, the CR character has been used to separate the commands and/or the records for the sake of clarity and readability. However, the CR is not required by the script processor and may be omitted if desired.

In all examples the script execution is begun using VTEK's Alt-K - option 4 selection.

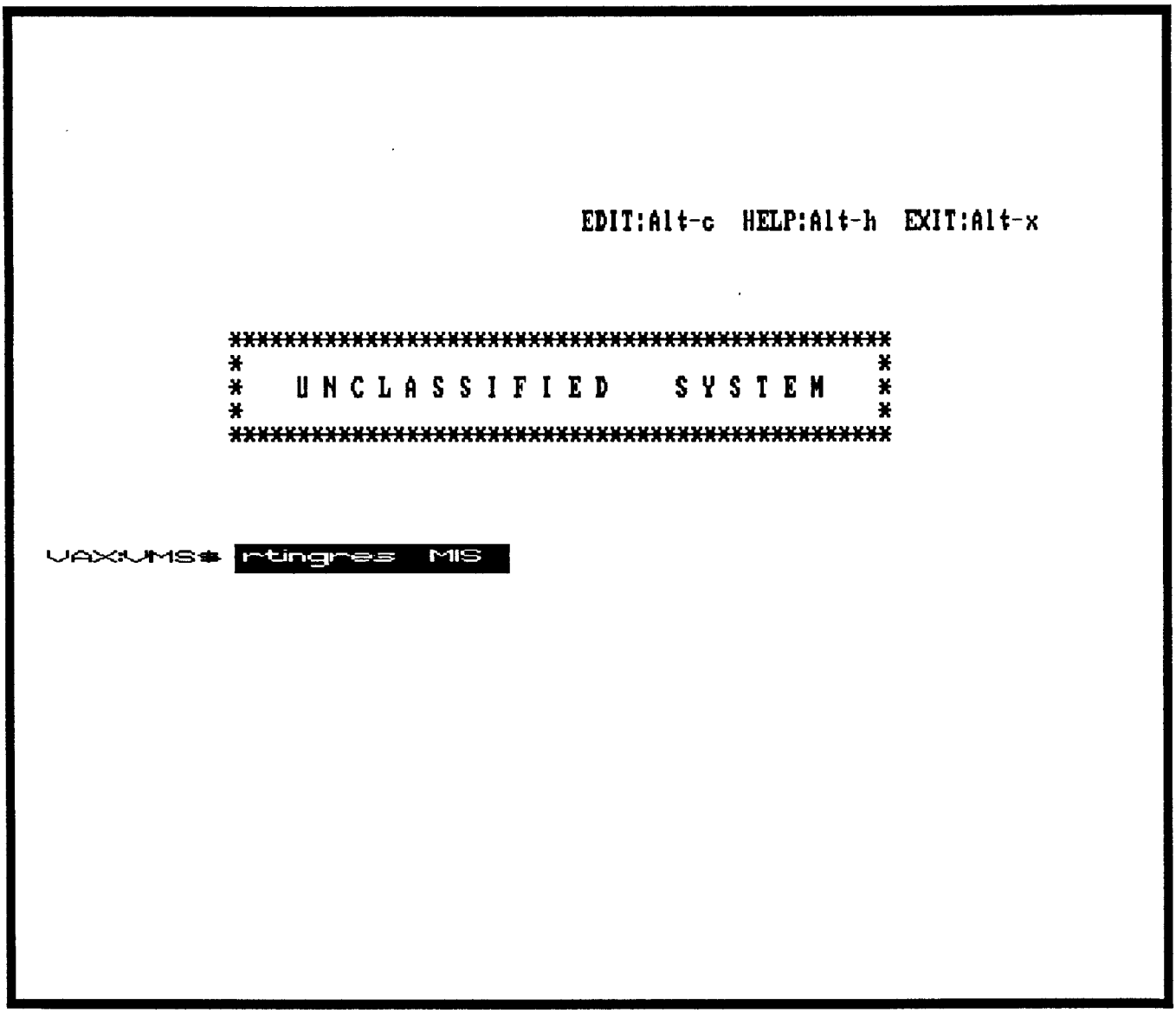
Example 1: Virtual Data Entry

The following example script command file (referred to as ITST.SCR) has the function of automatically interacting with a VAX-based INGRES database called "MIS". It is designed to illustrate automation of a process that requires interactive responses and does not show the most efficient method for updating databases. In some cases, DBMS's (INGRES, in this example) have the ability to process data in batch mode which can reduce on-line processing time and would, therefore, be more efficient than an interactive method.

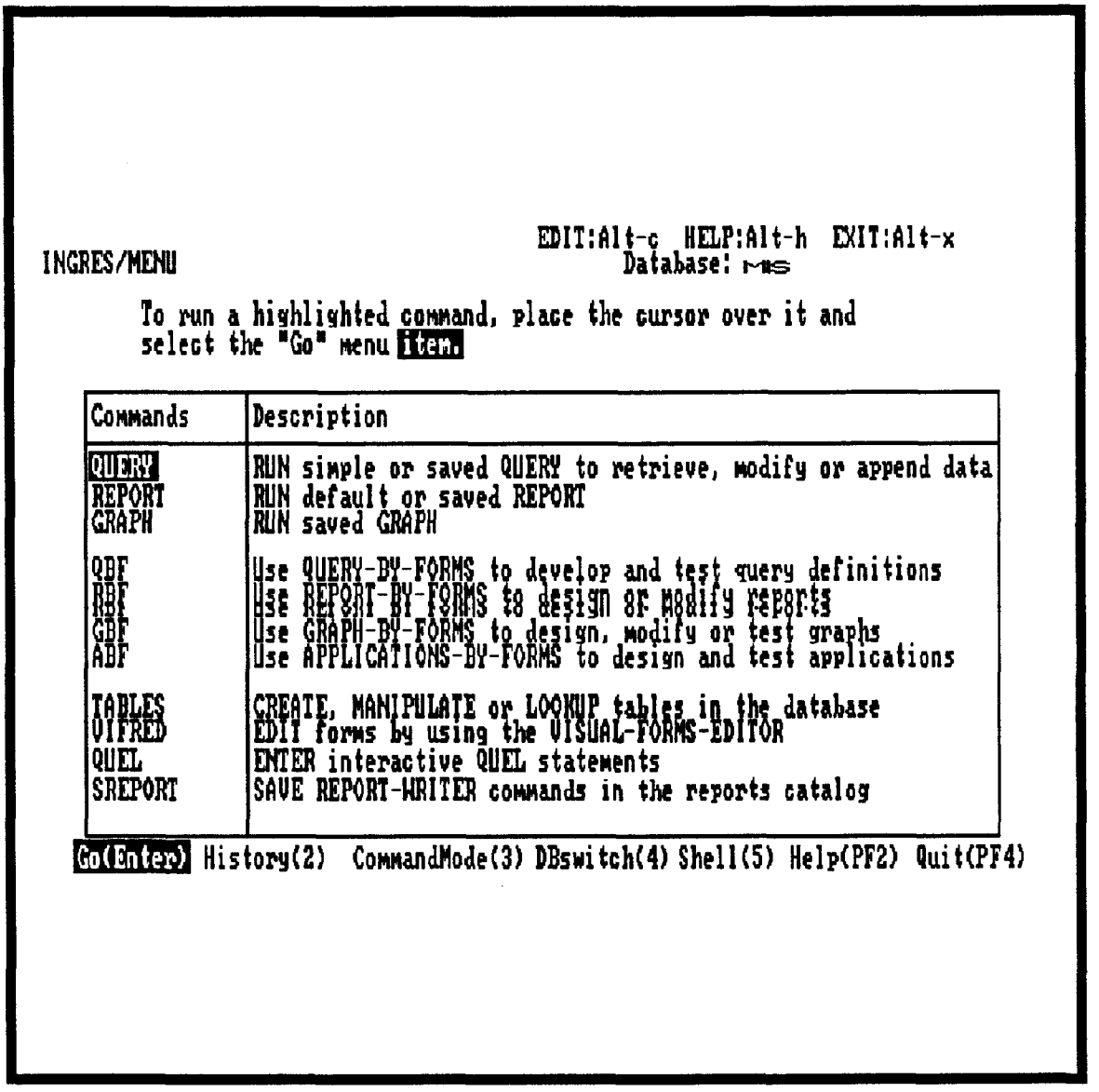
ITST.SCR Script Command File

Script	Figure	Explanation
rtingres MIS^n	B.1	Supply "rtingres MIS" plus CR to VMS starting INGRES. The INGRES command menu appears.
^d10tem.^d^ke	B.2	Delay until in last 10 chars "tem." is found then supply keypad "Enter" which selects QUERY.
^d12def name^d	B.3	Delay until in last 12 chars "def name" is found. At this point the QUERY menu is on the screen.
NAMESTAB^ke	B.3	Supply "NAMESTAB", (name of the table) and keypad "Enter". The QBF-Execution Phase submenu of Fig. B.4 to appear.
^d40Execution Phase^d	B.4	Delay until in last 40 chars "Execution Phase" is found.

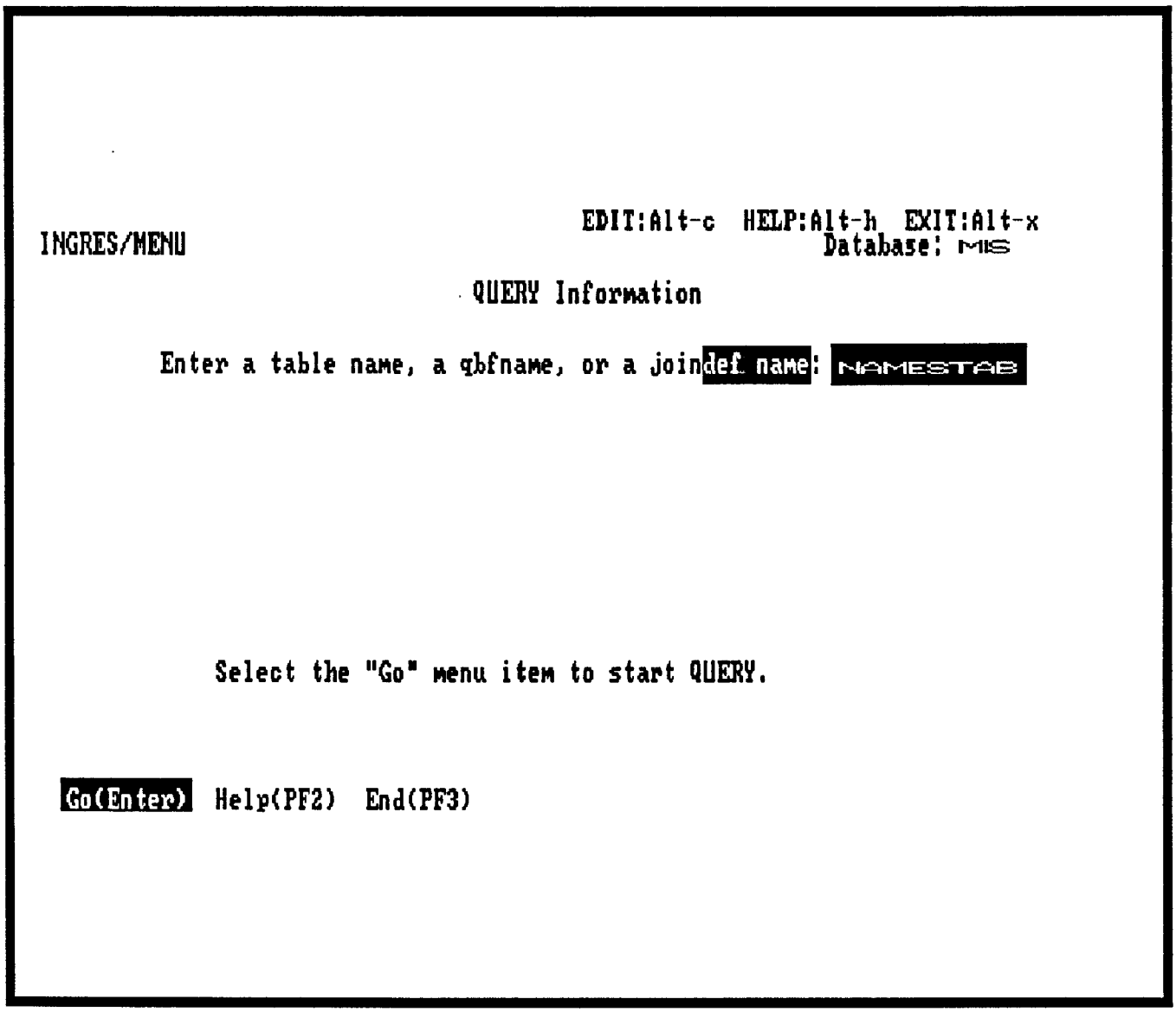
^d10:^d	B.4	Delay for ":" prompt at end of QBF submenu.
^k1	B.4	Supply keypad "1" designating the Append function.
^fnames.dat^f	-	Open data file "names.dat" on the PC for upload.
^d30last:^d	B.5	Delay until in last 30 chars "last:" is found. This is the prompt for a record to be appended to the "NAMESTAB" target table.
^la	-	Label "a" for goto.
^e^b	B.6	Extract data and send to host; at end of data, jump to "b".
^ke^100	B.6	Supply keypad "Enter". Delay 1 full sec (though not necessary).
^ga	-	Goto label "a" and get more data.
^lb	-	Label "b" for goto.
^ke^050	B.7	Supply keypad "Enter" to complete last record. Delay 1/2 sec.
^kp3^d10:^d	B.7	Supply keypad "PF3" to append rows. Then delay for ":" of Fig. B.8.
^kp4^d10tem.^d	B.8	Supply keypad "PF4" to exit QBF Menu. Then delay for "tem." from the INGRES command menu as in Fig. B.9.
^kd	B.9	Move cursor with keypad "downarrow" to REPORT from command menu.
^ke	B.9	Supply keypad "Enter".
^d10REPORT^d	B.10	Delay until in last 10 chars "REPORT" is found in the REPORT Information submenu.
NAMESTAB table	B.11	Supply "NAMESTAB", "tab", then "table" to select table-option report. Supply "tab", "tab", and "tab" to default other report options.
rpt.dat^ke	B.11	Supply "rpt.dat" followed with keypad "Enter" to specify the report name.
^d20ready^d^n	B.12	Delay for "ready" then supply CR to complete the INGRES REPORT process.
^kp4	B.13	Supply keypad "PF4" to exit INGRES.
type rpt.dat^n	B.14	Supply "type rpt.dat" and CR to VMS causing the default report to be typed to screen.
^q	-	Quit and end script macro.



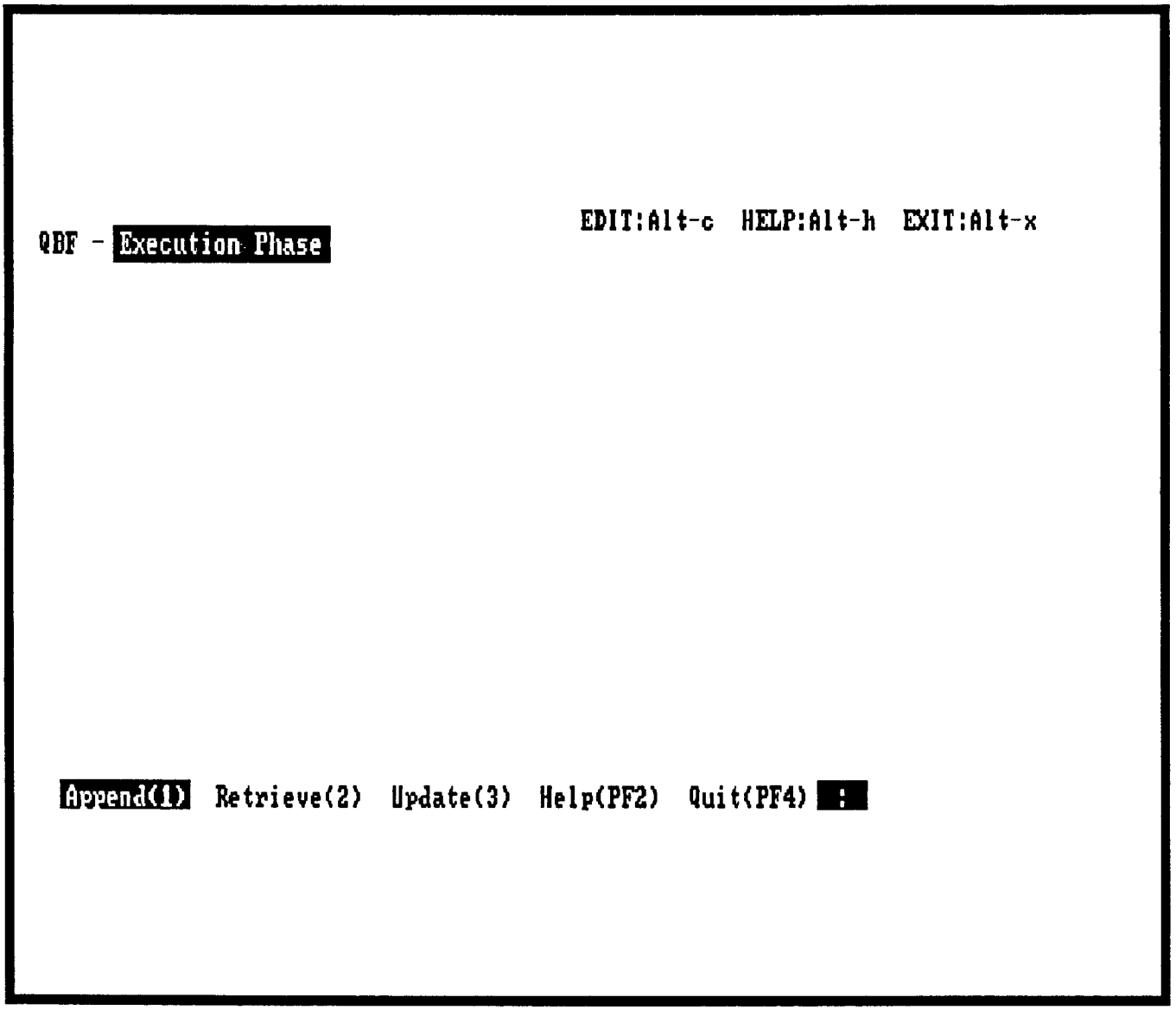
Initial OS prompt and command to begin the INGRES interactive session.
(INGRES)



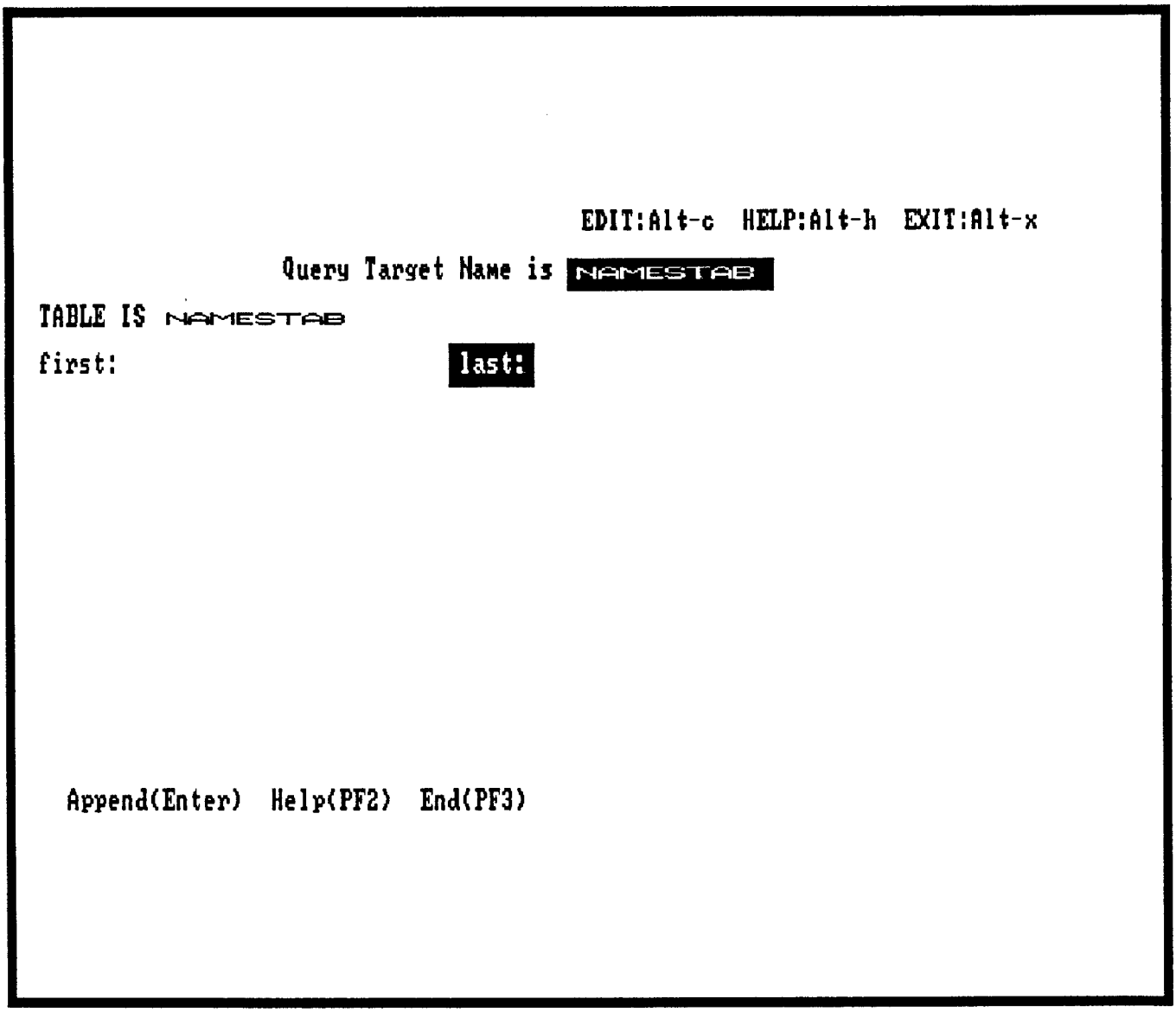
INGRES screen and commands to begin QUERY for appending data. (INGRES)



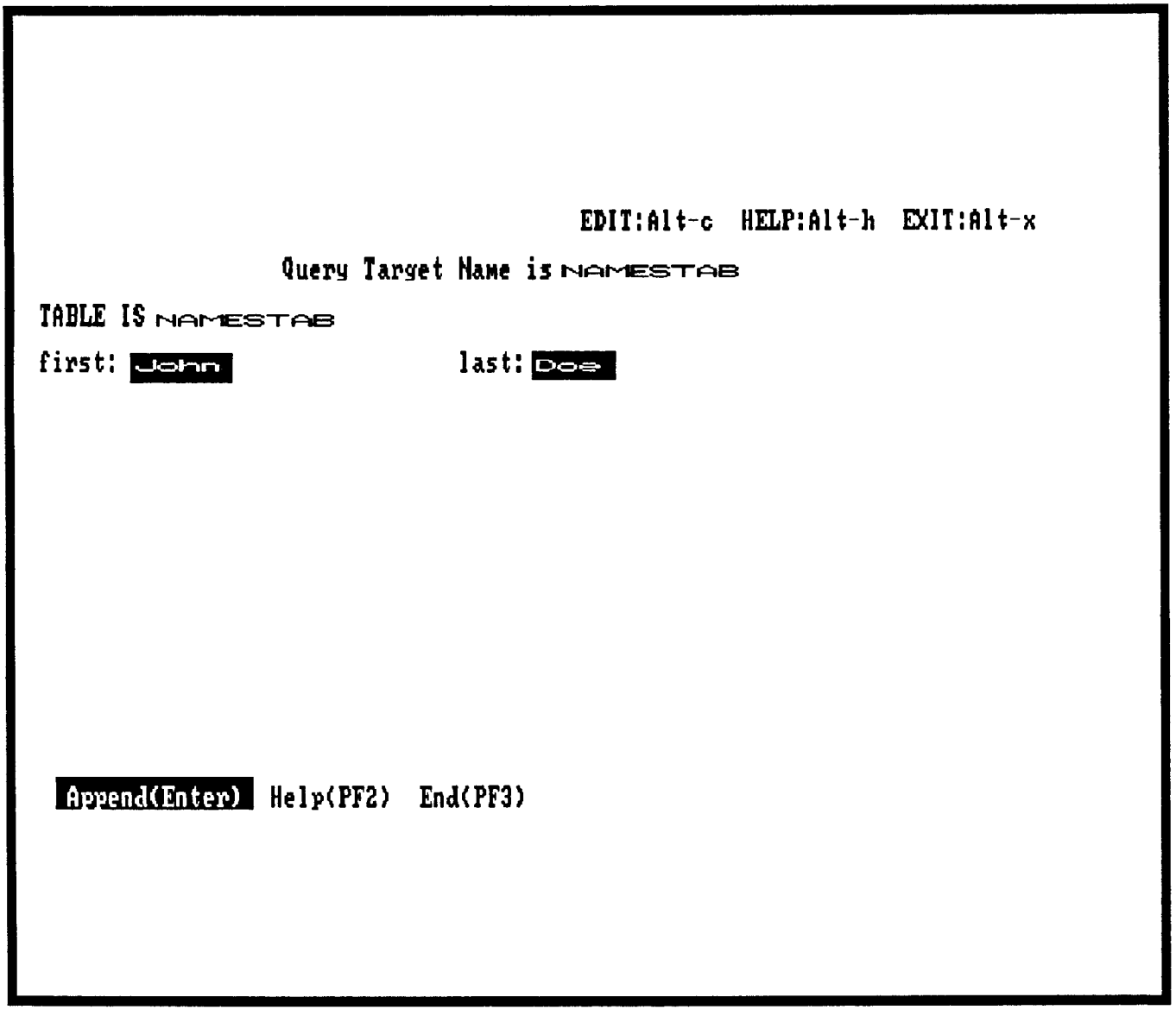
The QUERY submenu and commands to designate the table NAMESTAB.
(INGRES)



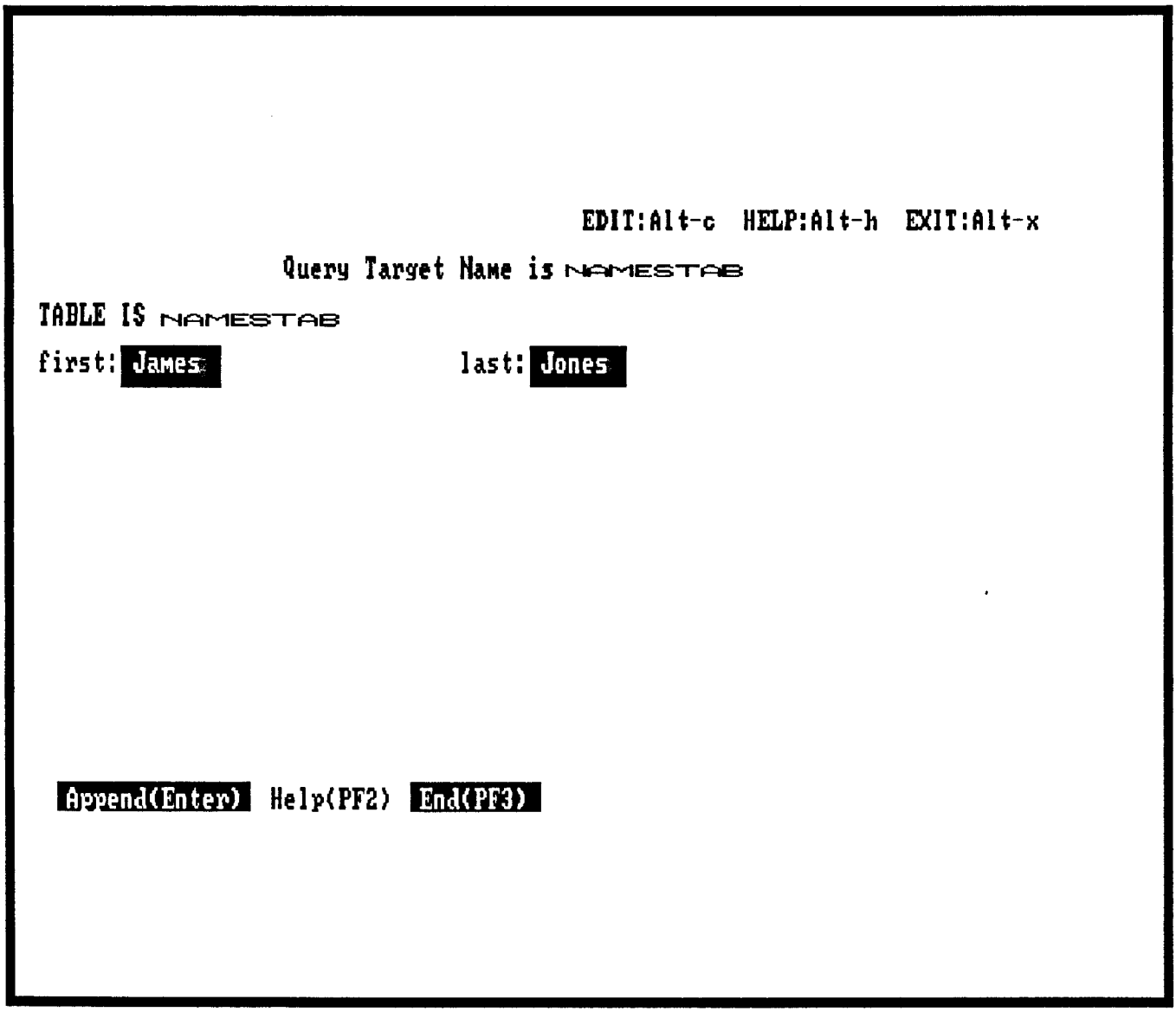
The INGRES QBF commands to append data.
(INGRES)



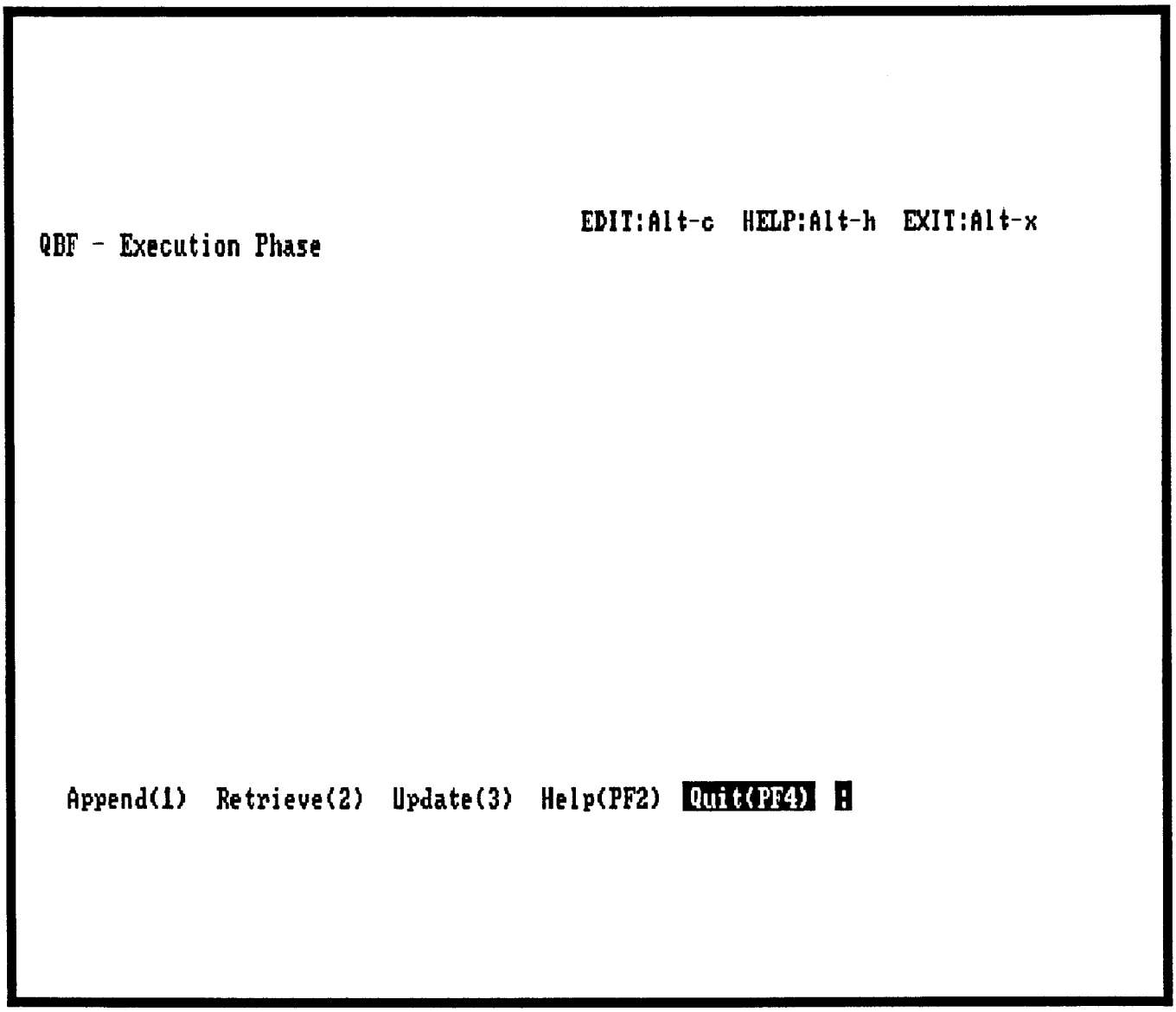
Designation of the query target as NAMESTAB.
(INGRES)



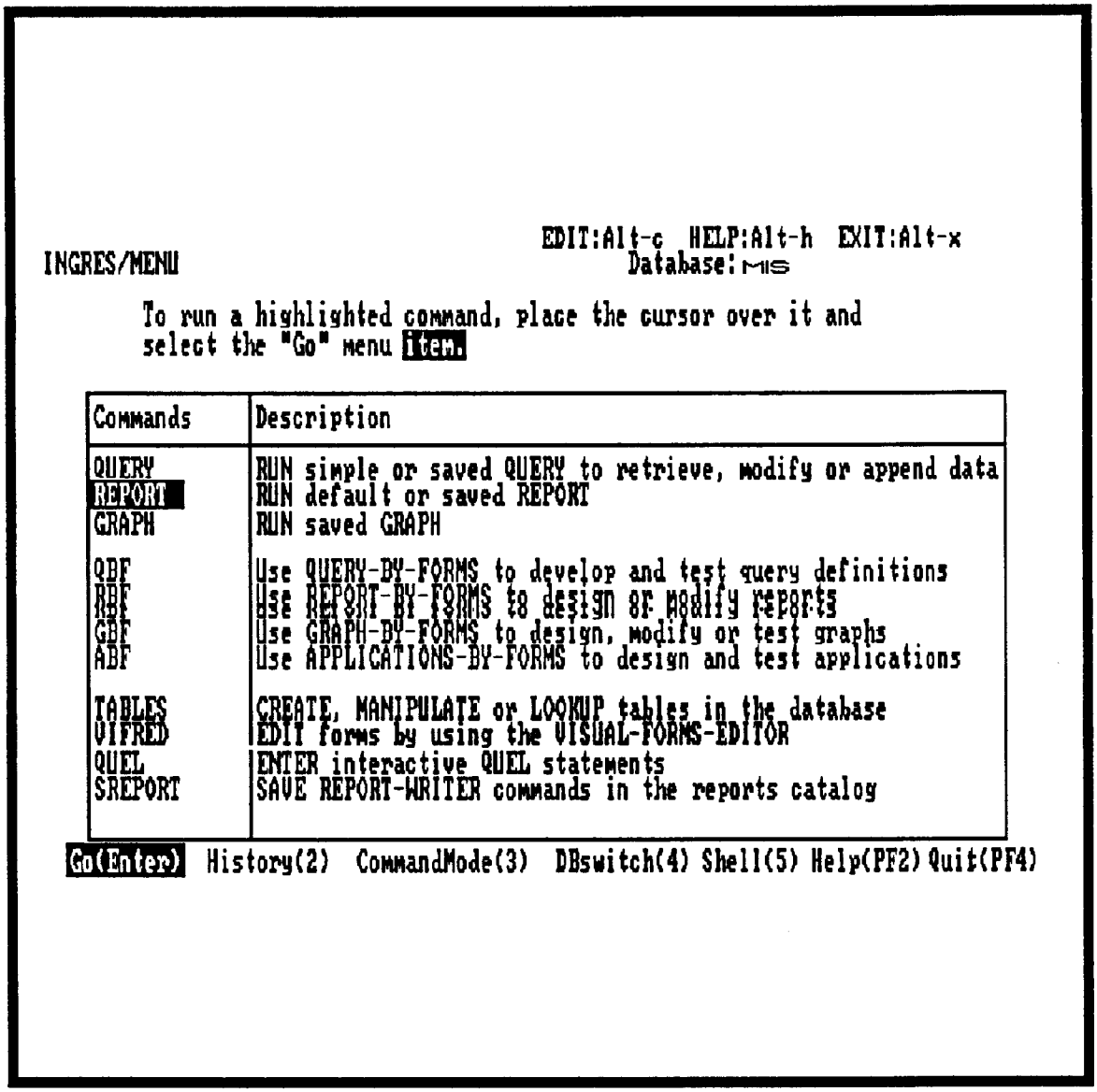
Name number one appended to NAMESTAB from the PC's data file, NAMES.DAT. (INGRES)



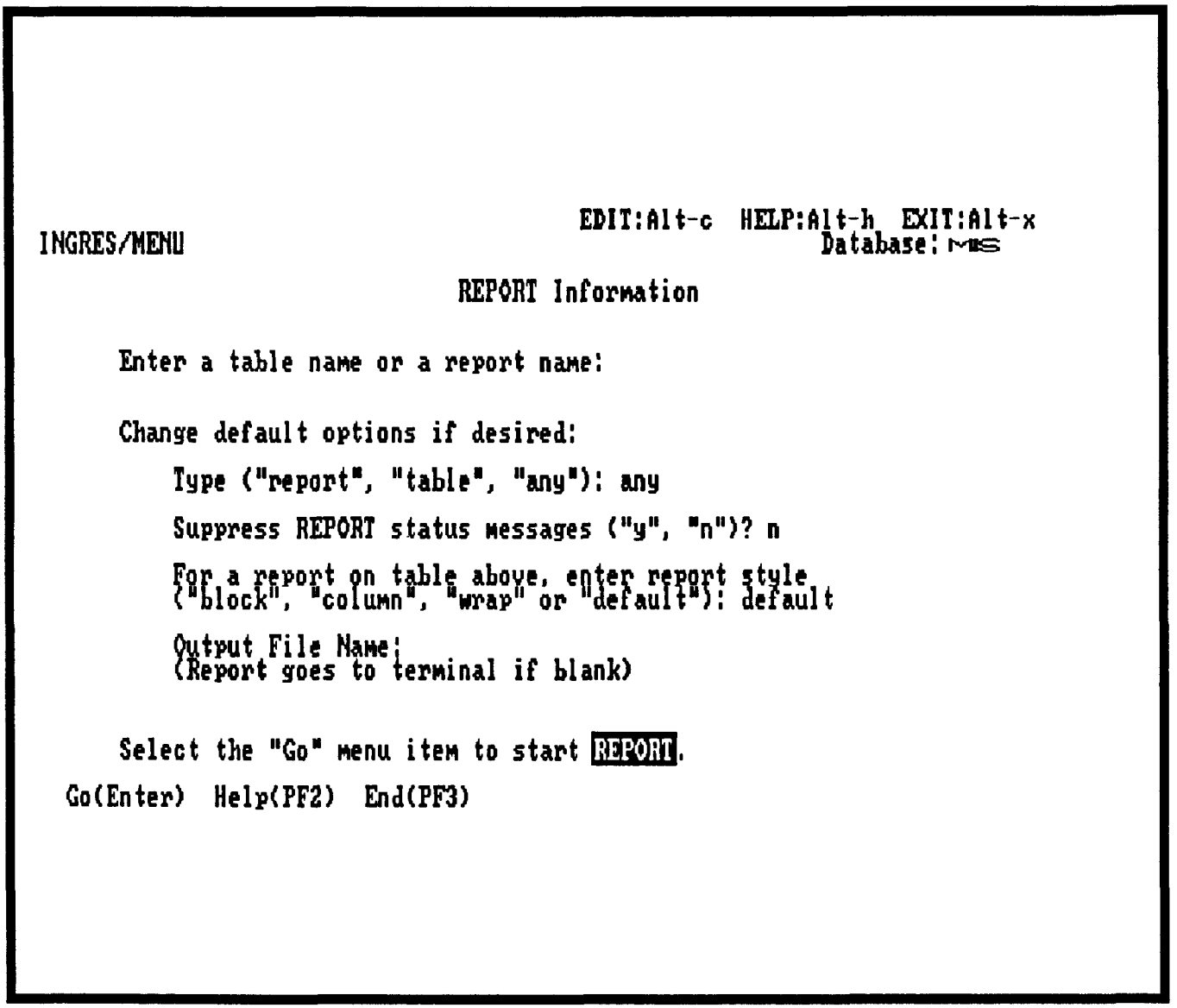
The last name appended to NAMESTAB.
(INGRES)



Quitting the INGRES QBF execution phase.
(INGRES)



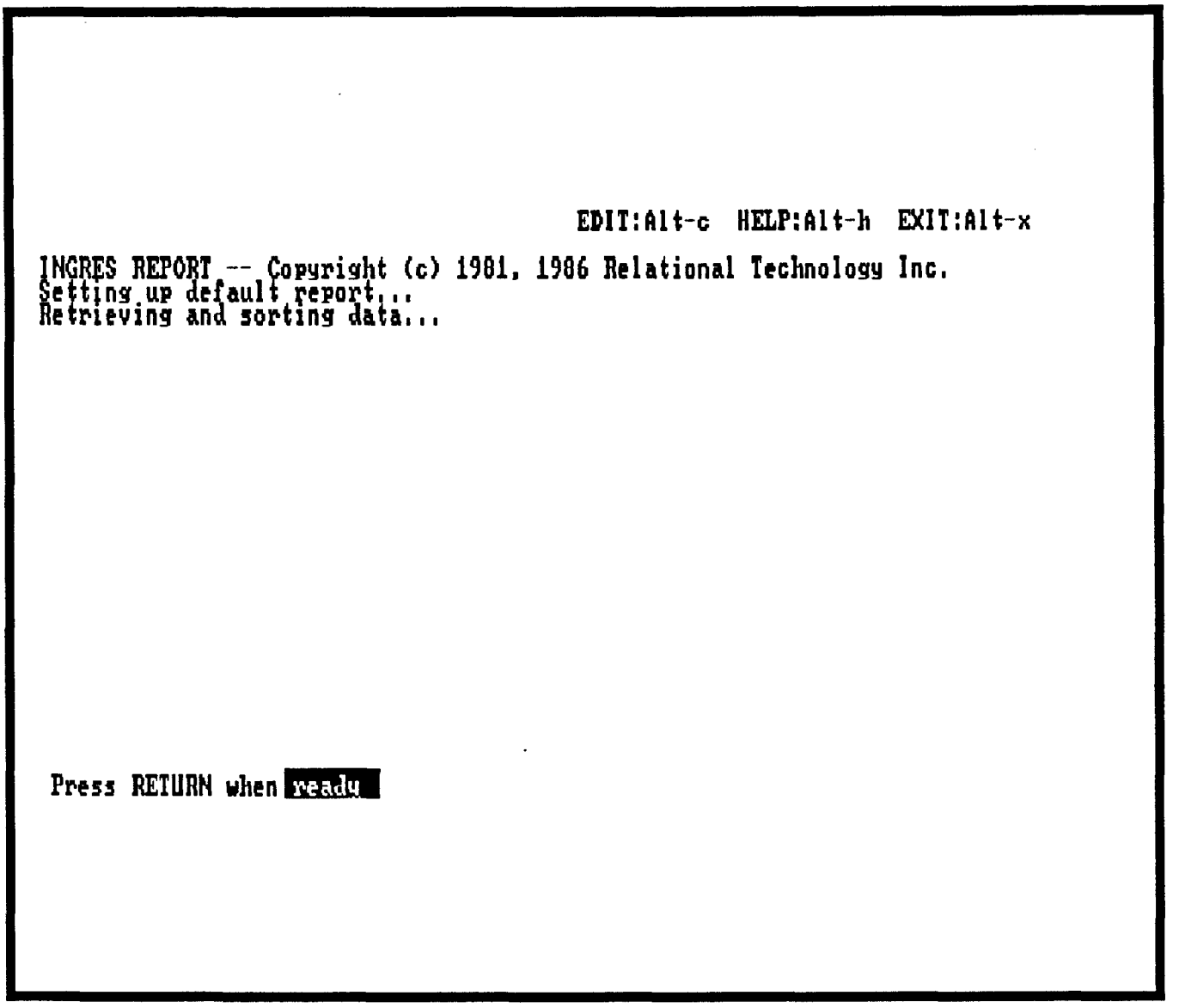
INGRES screen and commands to generate a report.
(INGRES)



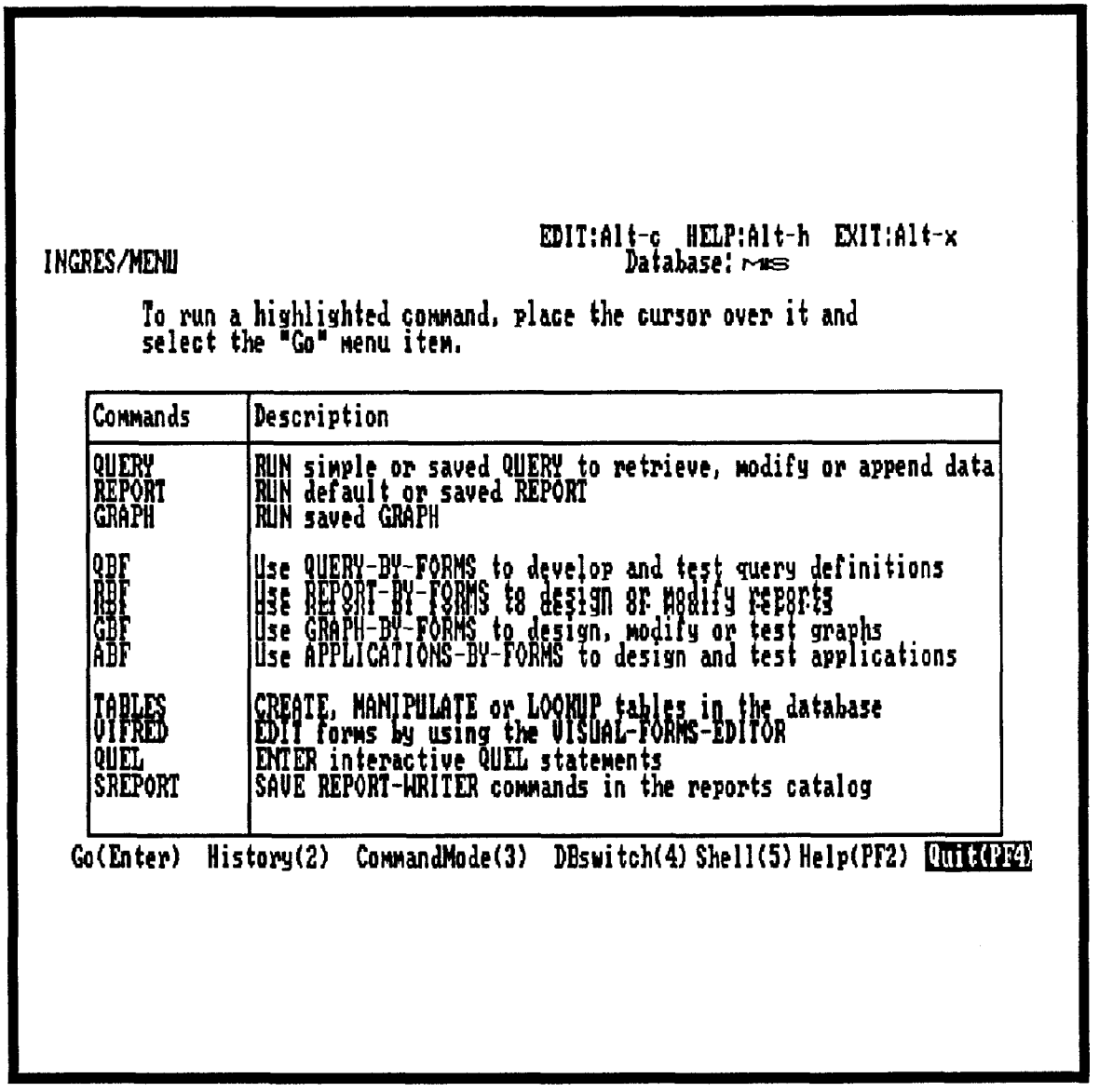
INGRES submenu for report information.
(INGRES)

```
                                        EDIT:Alt-c  HELP:Alt-h  EXIT:Alt-x
INGRES/MENU                                              Database: MIS

                          REPORT Information

     Enter a table name or a report name: NAMESTAB

     Change default options if desired:

         Type ("report", "table", "any"): table

         Suppress REPORT status messages ("y", "n")? n

         For a report on table above, enter report style
         ("block", "column", "wrap" or "default"): default

         Output File Name: rpt.dat
         (Report goes to terminal if blank)

     Select the "Go" menu item to start REPORT.

 Go(Enter)  Help(PF2)  End(PF3)
```

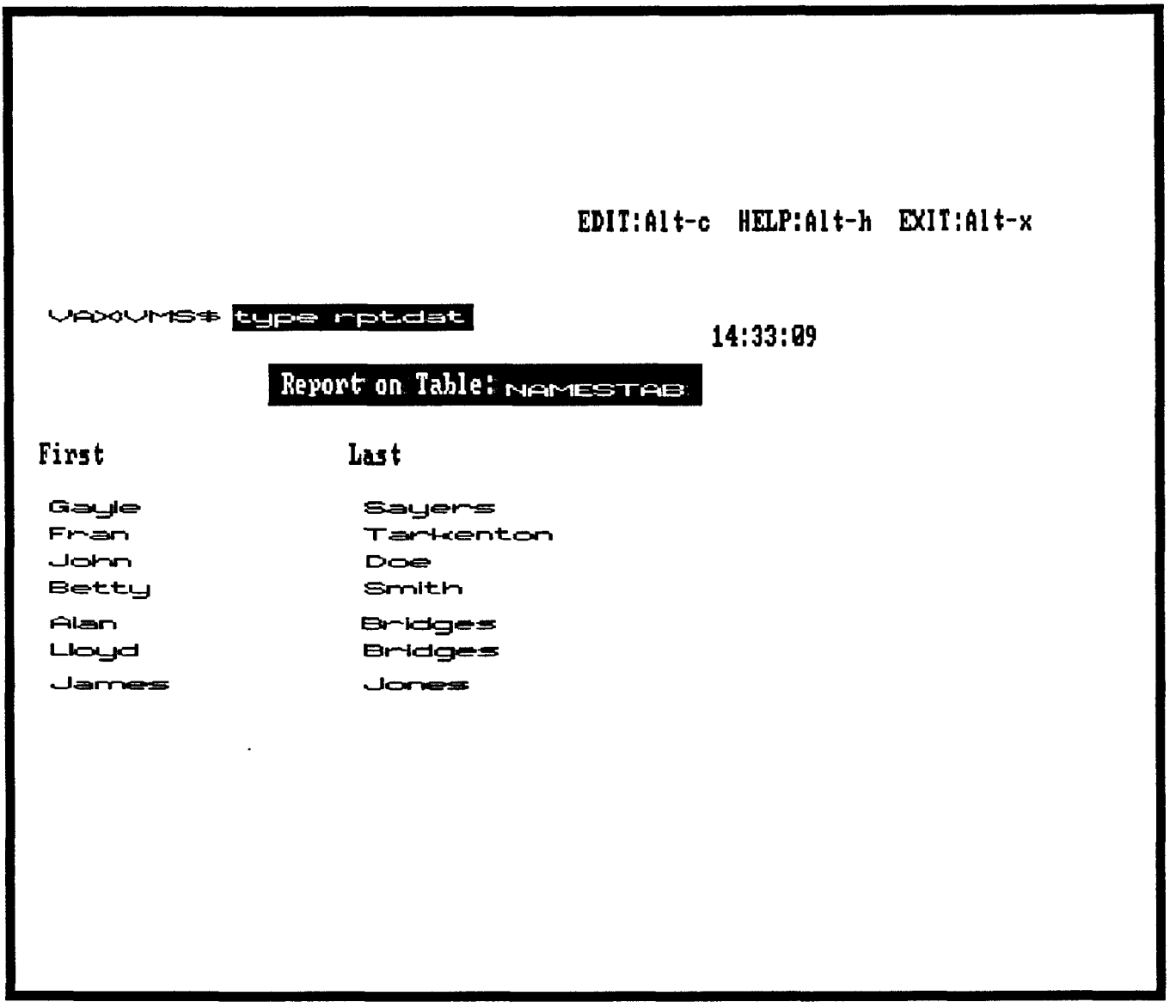
Answer to INGRES prompts for report characteristics designation.
(INGRES)



Completion of the INGRES report.
(INGRES)



The INGRES exit to OS.
(INGRES)



Typing the INGRES report to PC screen at the OS level of the VAX.
(INGRES)

Example 2: Demon Dialer

The script DD.SCR uses the Hayes command set to dial a number repeatedly until successful connection has been established.

```
^! DD.SCR - Script to repeatedly dial a number until a successful connection  ^!
^!          has been established.                                             ^!
^v
^!          Select the second set of communications parameters from the Setup ^!
^m2^!       Menu.                                                             ^!
^v
^la^!       Label A                                                           ^!
^!          If you do not have a touch tone phone, replace ATDT with ATDP     ^!
^100ATDTyour_phone_number^n^100^! Using the Hayes command set, dial the phone.^!
^=07^s07BUSY^a^! If a busy signal is encountered, wait 1 second and try again.^!
^q
```

Example 3: Kermit Download of a Host File

```
^! KMITDN.SCR - Script to download ascii files to PC.  Substitute          ^!
^!              appropriate filename for *.dat                            ^!
kermit^n^w2^!           Invoke Kermit on the host; wait for kermit prompt  ^!
send *.dat^n^!          Send all files with the extension DAT from the host^!
^v^az12n^n^n^v^!        Tell VTEK's Kermit to receive ascii file(s) to     ^!
^!                      default directory. Enter CR after completed xfer.  ^!
^Z^!                    Exit host Kermit                                   ^!
^q^!                    End of script                                      ^!
```

Example 4: Kermit Upload of a PC File

```
^! KMITUP.SCR - Script to upload ascii files to host.  Substitute          ^!
^!              appropriate filename for *.dat                            ^!
kermit^n^w2^!           Invoke Kermit on the host; wait for kermit prompt. ^!
receive^n^!             Tell host Kermit to receive file(s).               ^!
^v^az11n*.dat^n^n^v^!   Tell VTEK's Kermit to send all files ending in DAT ^!
^!                      from default directory.  File type is ascii.       ^!
^!                      Enter CR after completed transfer.                 ^!
^Z^!                    Exit host Kermit.                                  ^!
^q^!                    End of script.
```

Example 5: Interaction with VMS MAIL

```
^! GETMAIL.SCR -  Script to execute VMS MAIL and print any new messages on    ^!
^!                a local PC printer                                          ^!
MAIL^n^d10MAIL>^d^!           Invoke MAIL program                             ^!
^s25new^a^!                   Search for message indicating new mail          ^!
exit^n^q^!                    If not found, exit mail and quit script         ^!
^la^!                         Label a.  Starts new mail selection process.    ^!
select newmail^n^w>^!         Choose the newmail folder.                      ^!
extract/all outfile.tmp^n^!   Extract all new messages and place in a         ^!
^!                            temporary file.                                 ^!
^w>move/all/noconfirm MAIL^n^! Move all the extracted messages to folder MAIL.^!
^w>exit^n^!                   Wait for the prompt then exit MAIL.             ^!
^w$type outfile.tmp^n^!       Wait for the VMS $ prompt and type temp. file.  ^!
^d04.tmp^d^v^w^v^!            Turn on printer echo.                           ^!
^w$^!                         Wait for VMS prompt                             ^!
^v^w^v^!                      Turn off printer echo.                          ^!
del outfile.tmp;*^n^!         Delete temporary file containing mail messages.^!
^q^!                          Quit script.                                    ^!
```

APPENDIX C:

VDE: A VIRTUAL DATA ENTRY TERMINAL EMULATOR

The purpose of this component of the VDE project was to create a terminal emulator that, running on a PC, can enter data into a host-resident program as if it were a person sitting at a terminal keyboard entering the data. To this end, a program was written for an original (non-VTEK) program to implement VDE. The approach taken with the VDE program itself was to first write a generic VT100 emulator and then add features necessary for the virtual data entry ability. In this way one would avoid code and features, including possible side effects (or " bugs") that could accompany the use of an already existent terminal emulator that was not originally written with the virtual data entry in mind. A parallel benefit, to the project, is intimate familiarity with the resulting code.

Thus, a general DEC VT100 terminal emulator was written that has about 3/4ths of the functionality of a real VT100. It would require little work to bring VDE up to 90% VT100 functionality, ignoring features that are basically difficult or irrelevant on a PC. However, a closer emulation of the VT100 could be unneeded for its basic goal of performing virtual data entry. In its present state VDE works with such DEC VAX software as EDT, DEBUG, and DTR (Datatrieve) - which is probably enough. Response to additional escape sequences can be easily added in any case. It should also be noted that in its present state the program has a lot of temporary fixes and "make-do's" that would be taken care of in a final product.

VDE must get characters streaming from the host at nearly a thousand characters per second. To deal with this rush of data an interrupt service routine(ISR), written in assembly language, is activated for each incoming character. The ISR 'ascint' puts these characters in a buffer that it shares with 'vde.c' - see lines 13 and 16 of VDE. This is a circular buffer and its large size of 4096 characters and the messy code gyrations between lines 127 and 147 are to make sure it doesn't lose any data. Incidently the ISR 'ascint' is enabled by the subroutine 'ascenb' - see line 89 and 283 through 323. At line 121 the subroutine 'vtputch' is called to put the fetched character on the screen. Between lines 595 and 655 'vtputch' checks the character extensively and sets flags, pointers and calls subroutines as appropriate. Especially important is the shifting back and forth between the VT100's regular character set 'G0' and its graphic character set 'G1', and the all-important processing of escape sequences. The escape sequence processing in 'vtputch' can be viewed as entering various 'escape states' according to the second character in the escape sequence. At line 628 the character finally leaves 'vtputch' only to be further checked and processed by 'myputchar' (lines 451 - 519).

We have looked at the processing of the incoming character stream, we shall now examine how characters get from the keyboard to the host. An important aside at this point is to note that the program consists of three independently running "sub-tasks" that reside within a "do-forever" loop between lines 94 through 150. From 96 through 105 is the keyboard "sub-task", lines 107 through 110 (also see 101 - 103) do the virtual data entry "task call", and lines 112 through 149 process incoming host data. The keyboard "sub-task" is activated at line 96 when a key is hit. At line 98 'getvt100' is invoked to return a 'VT100' character or escape sequence. From lines 348 through 397 we see that keyboard processing is pretty straight-forward compared

to incoming data processing. The most involved and important is mapping certain special PC keys to VT100 keyboard escape sequences between lines 364 and 389 - thus implementing the VT100's function keys and cursor keys.

But all of the above VT100 functionality is only a necessary foundation for the real job - which is the virtual data entry. At this point in time 'VDE' is at the "proof of principle" stage - though it has a very good start. At present the code between lines 1094 and 1276 implements a 'hard-coded' virtual data entry that actually enters real data into a pre-existent VAX Datatrieve application that was written in 1985, over two years ago. This Datatrieve application was NOT modified in any way for this test and when originally written was never envisioned to be used in this manner. The fact that for now the virtual data entry is hard-coded should not be seen as detracting from the program - for the logic in 'do_virt' as well as its supporting subroutines ('send', 'waitfor', 'getdata', etc.) is exactly that which could be done in a good, usable first version of the real VDE. First, note that 'do_virt' must allow the other major "sub-tasks" - keyboard, and host data, to co-process with it. Also, and this is very important, 'do_virt' co-processes within itself. This means it is not locked into a rigid response sequence to the incoming host data stream - i.e. it could respond properly with the right data even if some host prompts came out of sequence or repeated or didn't come at all. To this end a "pseudo instruction pointer" (variable 'pip' in code) is used to control the flexible execution of 'do_virt'.

As can be seen from all of the above, this "proof of principle" stage VDE is an excellent basis for an actual functioning VDE to use with STAFS or other host (or PC) resident systems. The VT100 emulation is essentially complete for our purposes and the general approach to the virtual data entry (as well as some code) is done. The next step is to define a flexible easy-to-use english-like script language that a general computer-literate user could use to govern his virtual data entry requirements. VDE would read and 'compile' this VDE Script Language (VSL) into its own internal data structures and then govern its own behavior and response to the host appropriately. Sophisticated pattern matching and error responses could be specified by the user via a few simple english-like VSL commands. These commands would be like: "SEND", "WAITFOR", "TIMEOUT", "ON ERROR", "OPEN FILE", "MATCH PATTERN", etc. A similar script language, complete with conditional branching ("IF ... THEN") and special data types/objects was defined and realized by Johnson, 1984, for a large FORTRAN program that he did that simulated the 'hot-line' rolling of aluminum. The VDE VSL and its execution would be in some ways similar. As new VDE requirements and customer needs became apparent they could be added to and accommodated by this structure.

In conclusion, the present "proof-of-principle" VDE is ready to be made into a functioning product and performed the task at hand to provide an optional platform for future system development.

APPENDIX D:

VDE C LANGUAGE SOURCE

```
 1: /* vde.c = virtual data entry terminal emulator */
 2: #include <stdio.h>
 3: #include <dos.h>
 4: #include <signal.h>
 5: #include <stdlib.h>
 6: #include <ctype.h>
 7: #include <dtypes.h>
 8: #include <rs232.h>
 9: #include <keycodes.h>
10: #define  VIDEO_IO 0x10
11: #define  RS232_IO 0x14
12: #define  KEYBD_IO 0x16
13: #define  RSBUFSIZ 4096
14: #define  xon    17
15: #define  xoff   19
16: unsigned char rsbuffer[RSBUFSIZ];
17: int count;
18: int head;
19: union REGS inregs, outregs;
20: struct SREGS segregs;
21: FILE *virtfile = NULL;      /* handle for virtual entry file     */
22: int virtmode  = NULL;      /* flag for virtual entry mode       */
23: int pip = 0;               /* `pseudo instruction pointer' for do_virt  */
24: int look_flag = NULL;  /* flag to turn `lookee' to prn on & off  */
25: int mymy_flag = NULL;      /* flag to turn `lookmy' to prn on & off    */
26: int cs_save, ip_save;              /* save old rs232 ISR cs:ip here      */
27: int save_row, save_col;    /* save row and column here when DECSC        */
28: int esc_state = NULL;              /* set when have received ascii escape    */
29: int esc_lbrack = NULL;     /* set when recvd `[' as 2nd char in esc seq */
30: int esc_lparen = NULL;     /* set when recvd `(' as 2nd char in esc seq */
31: int esc_rparen = NULL;     /* set when recvd `)' as 2nd char in esc seq */
32: int esc_pound  = NULL;     /* set when recvd `#' as 2nd char in esc seq */
33: int esc_pstate = NULL;     /* set when recvd `P' as 2nd char in esc seq */
34:
35: char esc_seq[32];          /* save escape sequence here for `release'  */
36:
37: unsigned char *save_ch_set;/* save char set pointer here in DECSC  */
38: unsigned char ukset[128], usset[128], spset[128], *g0_set, *g1_set,          *ch_set;
39: unsigned char special[32] =
40:     {  32,   4, 178, 231, 232, 234, 233, 248,          /* remap ...*/
41:       241, 237, 251, 217, 191, 218, 192, 197,          /* vt100 g1.*/
42:       196, 196, 196,  95,  95, 195, 180, 193,          /* set to PC*/
43:       194, 179, 243, 242, 227, 215, 156, 205        };  /* ...chars */
44:
45: int vt_top, vt_bot, pc_top, pc_bot, vt_org; /* beg & ending rows, cols       */
46:
47: BYTE decckm   = NULL;             /* flag for set cursor key mode       */
48: BYTE decanm   = ~NULL;            /* flag for set ANSI/VT52 mode = ANSI*/
49: BYTE deccolm  = NULL;             /* flag for set column mode = 132    */
50: BYTE decsclm  = NULL;             /* flag for set scroll mode to smooth*/
51: BYTE decscnm  = NULL;             /* flag for set screen to reverse vid*/
52: BYTE decom    = NULL;             /* flag for set origin mode           */
53: BYTE decawm   = NULL;             /* flag for set auto wrap mode        */
54: BYTE decarm   = ~NULL;            /* flag for set auto repeat mode     */
55:
56: BYTE attr   = 0x0f;               /* display attribute - init bright white    */
57: BYTE save_attr;            /* save attribute here when DECSC    */
58:
59: char curupseq[4], curdnseq[4], currtseq[4], curltseq[4];  /* cursor esc              seqs*/
60:
61: char hostline[129];               /* accumulate host output line here */
62: int  host_cnt = 0;         /* count of # of characters in host line    */
63:
64: main()
```

```
{
    int   result, i=0, j, ascint(), outcom(), restore();
    int   xoff_flag=0, xon_flag=0, metric;
    unsigned char  ch, kh1;
    char statbuff[80], tempbuff[80], keybdseq[4];

    initpgm();                          /* initialize program parameters     */

    /* initialize the rs232 port - 9600 baud, 8 data bits, 1 stop, no                parity */
    inregs.h.ah = 0;
    inregs.h.al = 0xe3;
    inregs.x.dx = 0;
    result = int86(RS232_IO, &inregs, &outregs);
    if(outregs.x.cflag)
        {
        printf("Can not init serial port - error %d\n", result);
        exit(1);
        }

    segread(&segregs);
    count = segregs.ds;
    dsinit();
    count = 0;
    head =0;
    ascenb(ascint);
    onexit(restore);                        /* when exit restore old ISR */
    signal(SIGINT, SIG_IGN);
    system("cls");
    /* main loop of program - see if KB char, if yes pump out; else check ser.*/
    for(;;)
        {
        if(biokbhit() == NULL)
            {
      ch = getvt100(keybdseq);
            if( ch == 1)exit();
       if( ch == 2)system("command");
       if( ch == 4)virtmode = NULL;   /* ctrl-D means disable virt. mode*/
       if( ch == 5)virtmode = ~NULL;  /* ctrl-E means enable virt. mode */
       if( ch == 5)openvirt();         /* open virtual file */
       if( ch != 2 && ch != 5 )sendhost( keybdseq );
       }

   if(virtmode != NULL)
      {
      do_virt();
      }

        if(head != count)
            {
            kh1 = rsbuffer[head++];
            head = head & RSBUFSIZ - 1 ;
            if(count >= head)
                metric = count - head;
            else
                metric = head + RSBUFSIZ - count;
            if(kh1 == xoff) kh1 = 233;
       vtputch(kh1);
       hostline [ host_cnt ] = kh1;
       hostline [ ++host_cnt ] = NULL;
       if(host_cnt > 127) host_cnt = 0;
            }

        if(
            ((count > head && (head+RSBUFSIZ - count) < RSBUFSIZ/4) ||
             (count < head && (head - count) < RSBUFSIZ/4 ))        &&
```

```
130:            (xoff_flag == 0 )
131:          )
132:            {
133:            outcom( xoff);
134:            xoff_flag = 1;
135:            xon_flag = 0;
136:            }
137:        if(
138:            ((count > head && (head+RSBUFSIZ - count) > RSBUFSIZ/2) ||
139:             (count < head && (head - count) > RSBUFSIZ/2 )         ||
140:             (count == head ))                                      &&
141:            (xon_flag == 0 )
142:          )
143:            {
144:            outcom( xon);
145:            xon_flag = 1;
146:            xoff_flag = 0;
147:            }
148:
149:        }
150: }
151:
152: /* initpgm = func to initialize program parameters  */
153: int initpgm()
154: {
155:     makesets();                /* create char sets */
156:     pc_top = 0;
157:     pc_bot = 23;
158:     vt_top = 1;
159:     vt_bot = 24;
160:     ckm_set();                          /* cursor key mode(DECCKM) set  */
161:
162: }
163:
164: int ckm_set()                           /* func for cursor key mode(DECCKM) set */
165: {
166:     strcpy( curupseq, "\x1b[A" );
167:     strcpy( curdnseq, "\x1b[B" );
168:     strcpy( currtseq, "\x1b[C" );
169:     strcpy( curltseq, "\x1b[D" );
170:
171: }
172:
173: int ckm_off()                  /* func for cursor key mode(DECCKM) reset*/
174: {
175:     strcpy( curupseq, "\x1bOA" );
176:     strcpy( curdnseq, "\x1bOB" );
177:     strcpy( currtseq, "\x1bOC" );
178:     strcpy( curltseq, "\x1bOD" );
179:
180: }
181:
182: int anm_set()                           /* func for ANSI/VT52 mode set = ANSI   */
183: {
184: }
185:
186: int anm_off()                          /* func for ANSI/VT52 mode set = ANSI   */
187: {
188: }
189:
190: int colm_set()                         /* func for set column mode to = 132    */
191: {
192: }
193:
194: int colm_off()                         /* func for ANSI/VT52 mode set = ANSI   */
```

```
195: {
196: }
197:
198: int sclm_set()                          /* func for set scroll mode to smooth   */
199: {
200: }
201:
202: int sclm_off()                          /* func for reset scroll mode to jump   */
203: {
204: }
205:
206: int scnm_set()                          /* func to set screen mode reverse video*/
207: {
208:     int i;
209:     char far *video;
210:
211:     attr=( ( attr & 0x70 ) >> 4 ) | /*old forgrnd -->bakgrnd*/
212:    ( ( attr & 0x07 ) << 4 ) | /*old bakgrnd -->forgrnd*/
213:    ( attr & 0x80 ) |           /* keep blink bit        */
214:    ( attr & 0x08 ) ;           /* keep hi-intensity bit*/
215:
216:     video = 0xb8000000;
217:     for(i=0;i<1920;i++) video[2*i+1] = attr;
218:
219: }
220:
221: int scnm_off()                          /* func for reset screen mode regular   */
222: {
223:     scnm_set();                /* reverse of reverse = regular */
224: }
225:
226: int om_set()                            /* func for set origin mode (DECOM)     */
227: {
228: }
229:
230: int om_off()                            /* func for reset origin mode (DECOM)   */
231: {
232: }
233:
234: int awm_set()                           /* func for set autowrap mode (DECOM)  */
235: {
236: }
237:
238: int awm_off()                  /* func for reset autowrap mode (DECOM) */
239: {
240: }
241:
242: int arm_set()                  /* func for set autowrap mode (DECOM)   */
243: {
244: }
245:
246: int arm_off()                  /* func for reset autowrap mode (DECOM) */
247: {
248: }
249:
250: /* makesets = func to create vt100 character sets   */
251: int makesets()
252: {
253:     int i;
254:
255:     for(i=0; i<128; i++)    /* assign standard ascii values to chars  */
256:    {
257:    ukset[i] = i;
258:    usset[i] = i;
259:    spset[i] = i;
```

```
260:    }
261:      ukset[35] = 156;        /* fix up U.K. set with pound sign - currency  */
262:      for(i=95; i<127; i++) spset[i] = special[i - 95];       /*special & line                fix */
263:      g0_set = usset;            /* make g0 set to be U.S. set   */
264:      g1_set = spset;            /* make g1 set to be special & line drawing */
265:      ch_set = g0_set;           /* pick U.S. set as standard    */
266:
267: }
268:
269: /* outcom = func to write char to rs232 port */
270: outcom ( ch )
271: int ch;
272: {
273:      int j;
274:
275:      for(j=0;j<10000;j++)
276:          {
277:          if( ( inp(COMM_STAT) & 0x60 ) != 0 ) break;
278:          }
279:
280:      outp(COMM_DATA, ch);
281: }
282:
283: /* ascenb = func to enable my rs232 interrupt service routine */
284: ascenb( serial_isr )
285: int (*serial_isr)();
286: {
287:      unsigned char pmask, imask;
288:      char far * longptr;
289:      int result=0;
290:
291:      /* save old rs232 communications ISR address */
292:      inregs.h.ah = 0x35;
293:      inregs.h.al = COM_INT;
294:      result=intdosx(&inregs, &outregs, &segregs);
295:      if(outregs.x.cflag)
296:          {
297:    printf("Can't save old serial isr - error %d \n", result);
298:          exit(1);
299:    }
300:      cs_save = segregs.es;
301:      ip_save = outregs.x.bx;
302:
303:      /* set up new rs232 ISR = `ascint'      */
304:      longptr = (char far *) serial_isr;
305:      inregs.h.ah = 0x25;
306:      inregs.h.al = COM_INT;
307:      inregs.x.dx = FP_OFF(longptr);
308:      segread(&segregs);
309:      segregs.ds  = segregs.cs;
310:      result=intdosx(&inregs, &outregs, &segregs);
311:      if(outregs.x.cflag)
312:          {
313:          printf("Can't set up serial isr - error %d \n", result);
314:          exit(1);
315:          }
316:      outp(COMM_MCR, 0x0b);
317:      outp(COMM_IER, 1);
318:      pmask = inp(PIC_MASK);
319:      imask = INT_MASK;
320:      imask = ~imask;
321:      pmask = pmask & imask;
322:      outp(PIC_MASK, pmask);
323: }
324:
```

```
/* restore = func to old rs232 interrupt service routine, disable          interrupts */
restore()
{
    unsigned char pmask, imask;
    int result=0;

    inregs.h.ah = 0x25;
    inregs.h.al = COM_INT;
    inregs.x.dx = ip_save;
    segregs.ds     = cs_save;
    result=intdosx(&inregs, &outregs, &segregs);
    if(outregs.x.cflag)
        {
  printf("Can't restore serial isr - error %d \n", result);
        }
    outp(COMM_MCR, 0x0b);
    outp(COMM_IER, 0);
    pmask = inp(PIC_MASK);
    imask = INT_MASK;
    pmask = pmask | imask;          /* dis-able this interrupt */
    outp(PIC_MASK, pmask);
}

/* getvt100 = function to read PC keybd and return vt100 escape seq */
int getvt100( escapeseq )
char *escapeseq;
{
    int key, getkey();
    char c;

    key = getkey(&c);                       /* get PC key press */
    escapeseq[0] = c;                      /* stick it in string    */
    escapeseq[1] = '\0';            /* terminate string      */
    if( key < 128 ) return(key);    /* reg. ascii - return   */

    /* if come here was extended ascii - function key,etc    */

    switch(key)
   {
   case UPARROW  : strcpy( escapeseq, curupseq );break;      /* cursor up*/
   case DNARROW  : strcpy( escapeseq, curdnseq );break;      /* cur. down*/
   case RTARROW  : strcpy( escapeseq, currtseq );break;      /* cur right*/
   case LTARROW  : strcpy( escapeseq, curltseq );break;      /* cur. left*/

   case SFT_0    : strcpy( escapeseq, "\x1bOp" );break;      /* keypad 0 */
   case SFT_1    : strcpy( escapeseq, "\x1bOq" );break;      /* keypad 1 */
   case SFT_2    : strcpy( escapeseq, "\x1bOr" );break;      /* keypad 2 */
   case SFT_3    : strcpy( escapeseq, "\x1bOs" );break;      /* keypad 3 */
   case SFT_4    : strcpy( escapeseq, "\x1bOt" );break;      /* keypad 4 */
   case SFT_5    : strcpy( escapeseq, "\x1bOu" );break;      /* keypad 5 */
   case SFT_6    : strcpy( escapeseq, "\x1bOv" );break;      /* keypad 6 */
   case SFT_7    : strcpy( escapeseq, "\x1bOw" );break;      /* keypad 7 */
   case SFT_8    : strcpy( escapeseq, "\x1bOx" );break;      /* keypad 8 */
   case SFT_9    : strcpy( escapeseq, "\x1bOy" );break;      /* keypad 9 */

   case SFT_MINUS: strcpy( escapeseq, "\x1bOm" );break;      /* keypad - */
   case F5       : strcpy( escapeseq, "\x1bOl" );break;      /* keypad , */
   case SFT_DEL  : strcpy( escapeseq, "\x1bOn" );break;      /* keypad 9 */
   case SFT_PLUS : strcpy( escapeseq, "\x1bOM" );break;      /* kpd enter*/
   case F1       : strcpy( escapeseq, "\x1bOP" );break;      /* PF1      */
   case F2       : strcpy( escapeseq, "\x1bOQ" );break;      /* PF2      */
   case F3       : strcpy( escapeseq, "\x1bOR" );break;      /* PF3      */
   case F4       : strcpy( escapeseq, "\x1bOS" );break;      /* PF4      */
   case F9       : strcpy( escapeseq, "\0\0" );look_flag=~look_flag;break;
   case F8       : strcpy( escapeseq, "\0\0" );mymy_flag=~mymy_flag;break;
```

```
390:
391:    default: strcpy( escapeseq, "\0\0");/* return nulls for ignored keys*/
392:    }
393:
394:    key = escapeseq[0];
395:    return(key);
396:
397: }
398:
399: /* getkey   = function to read keys from keybd */
400: int getkey(c)
401: char *c;
402: {
403:     union REGS inregs, outregs;
404:     int   result;
405:     unsigned char  khar, scan;
406:
407:     inregs.h.ah = 0;
408:     result = int86(KEYBD_IO, &inregs, &outregs);
409:     khar = outregs.h.al;
410:     scan = outregs.h.ah;
411:     *c = ( (char) khar & 0x7f );
412:
413:     /* 1st     case - khar == 0 implies is "extended ascii" char (like                fun keys)*/
414:     if(khar == 0)    return( scan << 8 );
415:
416: /* 2nd     case - khar != 0 and scan > 0x46 implies shifted numeric          keypad   */
417:     if(scan > 0x46)  return( (scan << 8) + khar);
418:
419:     /* default        case - is standard ascii character - return it   */
420:     return( khar );
421:
422: }
423:
424: /* sendhost = func to send chars (inc. escape seqs) to host */
425: int sendhost( string )
426: char *string;
427: {
428:     int i=0, j;
429:
430:     while( *string != NULL && i < 80 )
431:    {
432:    for(j=0;j<100;j++);
433:    outp(COMM_DATA, *string++);
434:    i++;
435:    }
436: }
437:
438: /* send_dos = func to send chars (inc. escape seqs) to ansi.sys & dos */
439: int send_dos( string )
440: char *string;
441: {
442:     int i=0, j;
443:
444:     while( *string != NULL && i < 80 )
445:    {
446:    putchar( *string++);
447:    i++;
448:    }
449: }
450:
451: /* myputchar = func to `putchar' without auto cr/lf */
452: int myputchar(ch)
453: unsigned char ch;
454: {
```

```
455:     int old_row, old_col, new_row, new_col;
456:
457:     biogetcurpos( &old_row, &old_col );
458:
459:     if( ch == 0 || ch == 127 ) return;
460:
461:     if( ch == 5 )
462:    {
463:    sendhost("ANSWERBACK MESSAGE");
464:    return;
465:    }
466:
467:     if( ch == 7 )
468:    {
469:    printf("\a");
470:    return;
471:    }
472:
473:     if( ch == 8 )
474:    {
475:    printf("\b");
476:    return;
477:    }
478:
479:     if( ch == 9 )
480:    {
481:    new_col = ( ( old_col >> 3 ) + 1 ) << 3;
482:    biosetcurpos( old_row, new_col );
483:    return;
484:    }
485:
486:     if( ch == 10 || ch == 11 || ch ==12 )
487:    {
488:    new_row = old_row+1;
489:
490:    if(new_row > pc_bot)
491:        {
492:        new_row = pc_bot;
493:        bioscrolldn(1, pc_top, 0, pc_bot, 79, attr);    /* do scroll    */
494:        }
495:    biosetcurpos( new_row, old_col );
496:    return;
497:    }
498:
499:     if( ch == 13 )
500:    {
501:    biosetcurpos( old_row, 0 );
502:    return;
503:    }
504:
505:     if( ch == 24 || ch == 26 )
506:    {
507:    ch = 178;
508:    esc_state = NULL;
509:    }
510:
511:     /*putchar(ch);*/
512:     /* put char to video via bios call - so no auto cr/lf/scroll    */
513:     lookmy(ch);
514:     bioputch( 0, 1, ch, attr );
515:     if( old_col == 79 ) return;
516:
517:     biosetcurpos( old_row, old_col+1);
518:
519: }
```

```
520:
521: int biogetcurpos( row, col )       /* use bios call to get cursor posn */
522: int *row, *col;
523: {
524:     inregs.h.ah = 3;
525:     int86(VIDEO_IO, &inregs, &outregs);
526:     *row = outregs.h.dh;
527:     *col = outregs.h.dl;
528:
529: }
530:
531: int biosetcurpos( row, col )       /* use bios call to set cursor posn */
532: int row, col;
533: {
534:    inregs.h.ah = 2;
535:    inregs.h.bh = 0;
536:    inregs.h.dh = row;
537:    inregs.h.dl = col;
538:    int86(VIDEO_IO, &inregs, &outregs);
539:
540: }
541:
542: int bioscrolldn( nlines, beg_row, beg_col, end_row, end_col, attr) /*scrl            dn*/
543: int nlines, beg_row, beg_col, end_row, end_col, attr;
544: {
545:       inregs.h.ah = 6;                    /* bios video scroll down   */
546:       inregs.h.al = nlines;               /* number of lines to scroll*/
547:       inregs.h.ch = beg_row;              /* beginning row for scroll */
548:       inregs.h.cl = beg_col;              /* beginning column          */
549:       inregs.h.dh = end_row;              /* ending row for scroll    */
550:       inregs.h.dl = end_col;              /* ending column for scroll */
551:       inregs.h.bh = attr;          /* attribute for new line   */
552:       int86(VIDEO_IO, &inregs, &outregs); /* do it - bios video call  */
553:
554: }
555:
556: int bioscrollup( nlines, beg_row, beg_col, end_row, end_col, attr) /*scrl            up*/
557: int nlines, beg_row, beg_col, end_row, end_col, attr;
558: {
559:       inregs.h.ah = 7;                    /* bios video scroll `up'   */
560:       inregs.h.al = nlines;               /* number of lines to scroll*/
561:       inregs.h.ch = beg_row;              /* beginning row for scroll */
562:       inregs.h.cl = beg_col;              /* beginning column          */
563:       inregs.h.dh = end_row;              /* ending row for scroll    */
564:       inregs.h.dl = end_col;              /* ending column for scroll */
565:       inregs.h.bh = attr;          /* attribute for new line   */
566:       int86(VIDEO_IO, &inregs, &outregs); /* do it - bios video call  */
567:
568:       biosetcurpos( beg_row, beg_col );
569: }
570:
571: int bioputch( page, ntimes, khar, attr)     /* bios put character &          attribute*/
572: int page, ntimes, khar, attr;
573: {
574:     lookmy(khar);
575:     inregs.h.ah = 9;
576:     inregs.h.bh = page;
577:     inregs.x.cx = ntimes;
578:     inregs.h.al = khar;
579:     inregs.h.bl = attr;
580:     int86(VIDEO_IO, &inregs, &outregs);
581:
582: }
583:
584: int biogetch( page, khar, attr)        /* bios get character &              attribute*/
```

```
585: int page, *khar, *attr;
586: {
587:     inregs.h.ah = 8;
588:     inregs.h.bh = page;
589:     int86(VIDEO_IO, &inregs, &outregs);
590:     *khar = outregs.h.al;
591:     *attr = outregs.h.ah;
592:
593: }
594:
595: /* vtputch = function to put out a vt100 char */
596: int vtputch(khar)
597: unsigned char khar;
598: {
599:     unsigned char ch, so = 14, si = 15, esc = 27;
600:     char string[2];
601:
602:     ch = khar;
603:     lookee(ch);
604:     if( !esc_state )                    /* if not in escape state do...*/
605:   {
606:
607:   if( ch == esc )           /* char = ESCape */
608:       {
609:       esc_state = ~NULL;
610:       strcpy( esc_seq, "\x1b");
611:       return;
612:       }
613:
614:   if( ch == so )            /* char = SO <ctrl-N> turn on G1 char set */
615:       {
616:       ch_set = g1_set;
617:       return;
618:       }
619:
620:   if( ch == si )            /* char = SI <ctrl-O> turn off G1 char set */
621:       {
622:       ch_set = g0_set;
623:       return;
624:       }
625:
626:   ch = ch_set[ch]; /* map char into proper char set    */
627:
628:   myputchar(ch);            /* and put it out via own function  */
629:   return;
630:   }
631:
632:     /* if come here we are in an escape state - i.e. processing esc          sequence*/
633:     string[0] = ch;                 /* make char into a null terminated string*/
634:     string[1] = '\0';
635:     strcat(esc_seq, string);        /* ...continue escape sequence    */
636:
637:     if(esc_lbrack)  { do_lbrack(ch); return; }      /* go into left          bracket proc.*/
638:     if(esc_lparen)  { do_lparen(ch); return; }      /* go into left paren.           proc.*/
639:     if(esc_rparen)  { do_rparen(ch); return; }      /* go into right paren           proc*/
640:     if(esc_pound )  { do_pound(ch);  return; }      /* go into # sign                processing*/
641:     if(esc_pstate)  { do_pstate(ch); return; }      /* go into 'P' esc               proc.    */
642:
643:     /* if come here not currently in a specific escape seq state    */
644:     if( ch == '[' ) { esc_lbrack = ~NULL; return;}  /* goto lbrack state               */
645:     if( ch == '(' ) { esc_lparen = ~NULL; return;}  /* goto lparen state               */
646:     if( ch == ')' ) { esc_rparen = ~NULL; return;}  /* goto rparen state               */
647:     if( ch == '#' ) { esc_pound  = ~NULL; return;}  /* goto pound  state               */
648:     if( ch == 'P' ) { esc_pstate = ~NULL; return;}  /* goto pstate state               */
649:
```

```
650:     /* if have gotten to here we are dealing with single char esc seq */
651:
652:     do_esc_one(ch);
653:     return;
654:
655: }
656:
657: int looklook(ch_in)
658: unsigned char ch_in;
659: {
660:     static int i;
661:     unsigned char ch;
662:
663:     ch = ch_in;
664:     if ( ch == 27) ch=172;
665:     if ( ch < 32 )
666:    {
667:    ch=ch+224;
668:    }
669:      if ( i<0 ) i=0;
670:
671:     if(i++ > 70 )
672:    {
673:    i = 0;
674:    fprintf(stdprn, "\n" );
675:    }
676:      fprintf(stdprn, "%c", ch );
677: }
678:
679: int lookee(ch_in)
680: unsigned char ch_in;
681: {
682:     if(look_flag) looklook(ch_in);
683: }
684:
685: int lookmy(ch_in)
686: unsigned char ch_in;
687: {
688:     if(mymy_flag) looklook(ch_in);
689: }
690:
691: int do_lbrack(ch)              /* func to deal with `[' escape sequence*/
692: char ch;
693: {
694: /* for now will release escape state and let ANSI.SYS deal with it  */
695:     if( !isalpha(ch) ) return;      /* accum esc sequence til get alpha              char... */
696:     esc_state = NULL;               /* ...then remove escape state     */
697:     esc_lbrack = NULL;
698:
699:     if( ch == `J' ) { do_j_seq(); return; } /* esc seq ending with `J'              */
700:     if( ch == `K' ) { do_k_seq(); return; } /* esc seq ending with `K'              */
701:     if( ch == `m' ) { do_m_seq(); return; } /* esc seq ending with `m'              */
702:     if( ch == `r' ) { do_r_seq(); return; } /* esc seq ending with `r'              */
703:     if( ch == `h' ) { do_h_seq(); return; } /* esc seq ending with `h'              */
704:     if( ch == `l' ) { do_l_seq(); return; } /* esc seq ending with `l'              */
705:
706:     send_dos( esc_seq );   /* then send it to ansi.sys for interpret.  */
707:     esc_seq[0] = `\0';              /* ...and null escape string                */
708:
709: }
710: int do_pound(ch)                   /* func to deal with `#' escape seq*/
711: char ch;
712: {
713:     /* ignore for now */
714:     esc_state = NULL;
```

```
715:     esc_pound = NULL;
716:     esc_seq[0] = `\0';              /* ...and null escape string                */
717:
718: }
719:
720: int do_pstate(ch)                   /* func to deal with `P' escape seq*/
721: char ch;
722: {
723:     /* ignore for now */
724:     esc_state = NULL;
725:     esc_pstate = NULL;
726:     esc_seq[0] = `\0';              /* ...and null escape string                */
727:
728: }
729:
730: int do_esc_one(ch)                  /* func to deal with `ESC X' esc seq*/
731: char ch;
732: {
733:     esc_state = NULL;
734:     esc_seq[0] = `\0';              /* ...and null escape string                */
735:
736:     if(esc_seq[1] == `M') {revindex(); return;} /* reverse index up a                line */
737:     if(esc_seq[1] == `D') {index();    return;} /* index down a line                 */
738:     if(esc_seq[1] == `E') {nextline(); return;} /* go to start of next               line*/
739:     if(esc_seq[1] == `7') {savecurs(); return;} /* save cursor-DEC          private */
740:     if(esc_seq[1] == `8') {restcurs(); return;} /* restore cursor-DEC               priv.*/
741:
742: }
743:
744: int revindex()                          /* do reverse index */
745: {
746:     int row, col;
747:
748:     biogetcurpos( &row, &col);
749:     if( row <= pc_top )
750:    {
751:    bioscrollup(1, pc_top, 0, pc_bot, 79, attr); /* do scroll    */
752:    return;
753:    }
754:      biosetcurpos(--row, col);
755:
756: }
757:
758: int index()                           /* do index */
759: {
760:     int row, col;
761:
762:     biogetcurpos( &row, &col);
763:     if( row >= pc_bot )
764:    {
765:    bioscrolldn(1, pc_top, 0, pc_bot, 79, attr); /* do scroll    */
766:    return;
767:    }
768:      biosetcurpos(++row, col);
769:
770: }
771:
772: int nextline()                          /* do NEL */
773: {
774:     int row, col;
775:
776:     biogetcurpos( &row, &col);
777:     col = 0;
778:     if( row >= pc_bot )
779:    {
```

```
780:    bioscrolldn(1, pc_top, 0, pc_bot, 79, attr); /* do scroll    */
781:    biosetcurpos(row, col);
782:    return;
783:    }
784:      biosetcurpos(++row, col);
785:
786: }
787:
788: int savecurs()                      /* save cursor - DEC private DECSC  */
789: {
790:      biogetcurpos( &save_row, &save_col);
791:      save_attr = attr;
792:      save_ch_set = ch_set;
793:
794: }
795:
796: int restcurs()                      /* restore cursor - DEC private DECSC  */
797: {
798:      biosetcurpos( save_row, save_col);
799:      attr = save_attr;
800:      ch_set = save_ch_set;
801:
802: }
803:
804: int do_lparen(ch)          /* func to deal with `(' escape sequence*/
805: char ch;
806: /* left parenthesis escape sequences are a sub-set of Select Character              Set  */
807: /* (SCS) cmds. All lparen seqs are 3 bytes long - we are on last             character  */
808: {
809:      esc_state = NULL;                   /* release the escape state */
810:      esc_lparen = NULL;
811:      esc_seq[0] = `\0';             /* ...and null escape string                */
812:
813:      /* "ESC ( A" designates the UNITED KINGDOM char set as g0  */
814:      if( ch == `A' ) { g0_set = ukset; return;}
815:
816:      /* "ESC ( B" designates the UNITED STATES char set as g0        */
817:      if( ch == `B')  { g0_set = usset; return;}
818:
819:      /* "ESC ( 0" designates the special char set as g0                      */
820:      if( ch == `0')  { g0_set = spset; return;}
821:
822:      /* "ESC ) 1" designates the `alternate ROM standard' char set as g0    */
823:      if( ch == `1')  { g0_set = usset; return;}
824:
825:      /* "ESC ) 2" designates the `alternate ROM special'  char set as g0 */
826:      if( ch == `2')  { g0_set = spset; return;}
827:
828: }
829:
830: int do_rparen(ch)              /* func to deal with `)' escape sequence*/
831: char ch;
832: /* right parenthesis escape sequences are a sub-set of Select Character             Set  */
833: /* (SCS) cmds. All lparen seqs are 3 bytes long - we are on last             character  */
834: {
835:      esc_state = NULL;                   /* release the escape state */
836:      esc_rparen = NULL;
837:      esc_seq[0] = `\0';             /* ...and null escape string                */
838:
839:      /* "ESC ( A" designates the UNITED KINGDOM char set as g1  */
840:      if( ch == `A' ) { g1_set = ukset; return;}
841:
842:      /* "ESC ( B" designates the UNITED STATES char set as g1        */
843:      if( ch == `B')  { g1_set = usset; return;}
844:
```

```
845:     /* "ESC ( 0" designates the special char set as g1                    */
846:     if( ch == `0')  { g1_set = spset; return;}
847:
848:     /* "ESC ) 1" designates the `alternate ROM standard' char set as g1   */
849:     if( ch == `1')  { g1_set = usset; return;}
850:
851:     /* "ESC ) 2" designates the `alternate ROM special'  char set as g1   */
852:     if( ch == `2')  { g1_set = spset; return;}
853:
854: }
855:
856: int do_r_seq()          /* func to perform scroll escape sequence ( `r' ) */
857: {
858:     char    *seq;
859:
860:     seq = esc_seq + 2;                      /* point past "<ESC>["        */
861:     esc_seq [0] = `\0';             /* null out old string           */
862:
863:     vt_top = strtol( seq, &seq, 10);        /* get top vt100 row for scroll */
864:     if( vt_top <= 0 ) vt_top = 1;   /* make sure is legal            */
865:     if( vt_top > 24 ) vt_top = 24;  /* = 1 to 24 inclusive           */
866:
867:     vt_bot = strtol( ++seq, &seq, 10);      /* get bottom vt row for scroll */
868:     if( vt_bot <= 0 ) vt_bot = 24;  /* make sure is legal            */
869:     if( vt_bot > 24 ) vt_bot = 24;  /* = 1 to 24 inclusive           */
870:
871:     pc_top = vt_top - 1;            /* top pc row is 0,...           */
872:     pc_bot = vt_bot - 1;            /* top vt100 row is 1            */
873:
874:     home_up();                              /* home up cursor            */
875:
876: }
877:
878: int home_up()                               /* func to home up cursor   */
879: {
880:
881:     if( decom )                     /* origin mode set?...   */
882:   biosetcurpos ( pc_top, 0);        /* ...yes, go margin     */
883:     else
884:   biosetcurpos ( 0, 0);             /* ...no, go  1,1        */
885:
886: }
887:
888: int do_h_seq()          /* decode DEC private mode set state esc seq( `h' ) */
889: {
890:     char    *seq;
891:     int i;
892:
893:     seq = esc_seq + 1;                      /* point past "<ESC>"        */
894:     esc_seq [0] = `\0';             /* null out old string           */
895:
896:     for(i=0; i < strlen(seq); i++ ) /* make `?' into `;' so easy decode  */
897:   {
898:   if(seq[i] == `?') seq[i] = `;' ;
899:   }
900:     while ( *seq != NULL )
901:   {
902:   i = strtol( ++seq, &seq, 10);         /* get next set mode option */
903:   switch(i)
904:       {
905:       case 1:decckm =~NULL;ckm_set() ;break;/*set cursor key mode      */
906:       case 2:decanm =~NULL;anm_set() ;break;/*set ANSI/VT52 mode = ANSI*/
907:       case 3:deccolm=~NULL;colm_set();break;/*set column mode = 132    */
908:       case 4:decsclm=~NULL;sclm_set();break;/*set scroll mode to smooth*/
909:       case 5:decscnm=~NULL;scnm_set();break;/*set screen to reverse vid*/
```

```
910:        case 6:decom  =~NULL;om_set();  break;/*set origin mode            */
911:        case 7:decawm =~NULL;awm_set(); break;/*set auto wrap mode         */
912:        case 8:decarm =~NULL;arm_set(); break;/*set auto repeat mode      */
913:        }
914:    }
915:
916: }
917:
918: int do_l_seq()          /* decode DEC private mode reset state esc seq( `l' )          */
919: {
920:     char *seq;
921:     int i;
922:
923:     seq = esc_seq + 1;                      /* point past "<ESC>"             */
924:     esc_seq [0] = `\0';             /* null out old string           */
925:
926:     for(i=0; i < strlen(seq); i++ ) /* make `?' into `;' so easy decode              */
927:    {
928:    if(seq[i] == `?') seq[i] = `;' ;
929:    }
930:      while ( *seq != NULL )
931:    {
932:    i = strtol( ++seq, &seq, 10);         /* get next set mode option */
933:    switch(i)
934:        {
935:        case 1:decckm =NULL;ckm_off() ;break;/*reset cursor key mode    */
936:        case 2:decanm =NULL;anm_off() ;break;/*reset ANSI/VT52 mode=ANSI*/
937:        case 3:deccolm=NULL;colm_off();break;/*reset column mode = 132  */
938:        case 4:decsclm=NULL;sclm_off();break;/*reset scroll mode smooth */
939:        case 5:decscnm=NULL;scnm_off();break;/*reset screen reverse vid */
940:        case 6:decom  =NULL;om_off();  break;/*reset origin mode           */
941:        case 7:decawm =NULL;awm_off(); break;/*reset auto wrap mode     */
942:        case 8:decarm =NULL;arm_off(); break;/*reset auto repeat mode   */
943:        }
944:    }
945:
946: }
947:
948: int do_m_seq()          /* decode Set Graphic Rendition (SGR) esc seqs( `m' )          */
949: {
950:     char *seq;
951:     int i;
952:     BYTE fc=0xf8, bc=0x8f; /* fc = foregrnd clear, bc = backgrnd            clear */
953:
954:     BYTE blk=0, red=4, grn=2, yel=6, blu=1, mag=5, cyn=3, wht=7;    /*              colors*/
955:
956:     seq = esc_seq + 1;                      /* point past "<ESC>"             */
957:     esc_seq [0] = `\0';             /* null out old string           */
958:
959:     for(i=0; i < strlen(seq); i++ ) /* make `?' into `;' so easy decode              */
960:    {
961:    if(seq[i] == `?') seq[i] = `;' ;
962:    }
963:      while ( *seq != NULL && *seq != `m' )
964:    {
965:    i = strtol( ++seq, &seq, 10);         /* get next set mode option */
966:    switch(i)
967:        {
968:        case 0: attr=0x07           ;break;       /* attrbutes off wht/blk*/
969:        case 1: attr=attr | 0x08   ;break;        /* bold on - hi intens. */
970:        case 5: attr=attr | 0x80   ;break;        /* blink on - set bit   */
971:        case 7: attr=( ( attr & 0x70 ) >> 4 ) | /*old forgrnd -->bakgrnd*/
972:                    ( ( attr & 0x07 ) << 4 ) | /*old bakgrnd -->forgrnd*/
973:                    ( attr & 0x80 ) |           /* keep blink bit        */
974:                    ( attr & 0x08 ) ;           /* keep hi-intensity bit*/
```

```
                    break;              /* ...thus reverse video*/
      case 8: attr=0                ;break;       /* cancelled on - invis.*/
      case 30:attr=attr & fc | blk     ;break;/* black   foreground   */
      case 31:attr=attr & fc | red     ;break;/* red     foreground   */
      case 32:attr=attr & fc | grn     ;break;/* green   foreground   */
      case 33:attr=attr & fc | yel     ;break;/* yellow  foreground   */
      case 34:attr=attr & fc | blu     ;break;/* blue    foreground   */
      case 35:attr=attr & fc | mag     ;break;/* magenta foreground   */
      case 36:attr=attr & fc | cyn     ;break;/* cyan    foreground   */
      case 37:attr=attr & fc | wht     ;break;/* white   foreground   */
      case 40:attr=attr & bc | (blk<<4);break;/* black   background   */
      case 41:attr=attr & bc | (red<<4);break;/* red     background   */
      case 42:attr=attr & bc | (grn<<4);break;/* green   background   */
      case 43:attr=attr & bc | (yel<<4);break;/* yellow  background   */
      case 44:attr=attr & bc | (blu<<4);break;/* blue    background   */
      case 45:attr=attr & bc | (mag<<4);break;/* magenta background   */
      case 46:attr=attr & bc | (cyn<<4);break;/* cyan    background   */
      case 47:attr=attr & bc | (wht<<4);break;/* white   background   */
      }
  /*fprintf(stdprn, "%s %d %x\n", seq, i, attr);*/
  }

    /* flip hi-intensity state */
    if( (attr & 8) == 0 )
  attr = attr | 8;
    else
  attr = attr & 0xf7;

}

int do_j_seq()          /* decode & do Erase in Display(ED)esc seqs(cap `J')       */
{
    char *seq;
    int i, row, col, ntimes;

    seq = esc_seq + 1;                      /* point past "<ESC>"            */
    esc_seq [0] = `\0';             /* null out old string           */

    i = strtol( ++seq, &seq, 10);   /* get ED option */
    switch(i)
  {

  case 0: /* erase from cursor to end of the screen   */
      biogetcurpos( &row, &col ); /* save present row and column     */
      ntimes = ( 23 - row ) * 80 + ( 80 - col );    /* # of blanks   */
      if(ntimes < 0) ntimes = 0;
      bioputch( 0, ntimes, ` ', attr);      /* erase    */
      biosetcurpos( row, col);
      break;

  case 1: /* erase from beg of screen to cursor         */
      biogetcurpos( &row, &col ); /* save present row and column       */
      ntimes = row * 80 + col;      /* # of blanks to put out to video */
      if(ntimes < 0) ntimes = 0;
      biosetcurpos( 0, 0);
      bioputch( 0, ntimes, ` ', attr);      /* erase    */
      biosetcurpos( row, col);
      break;

  case 2: /* erase all of screen do not move cursor */
      biogetcurpos( &row, &col ); /* save present row and column       */
      ntimes = 1920;    /* # of blanks to put out to VT100 screen    */
      biosetcurpos( 0, 0);
      bioputch( 0, ntimes, ` ', attr);      /* erase    */
      biosetcurpos( row, col);
```

```
1040:       break;
1041:
1042:    }
1043:
1044: }
1045:
1046: int do_k_seq()         /* decode & do Erase in Line(EL)esc seqs( cap `K' )          */
1047: {
1048:     char *seq;
1049:     int i, row, col, ntimes;
1050:
1051:     seq = esc_seq + 1;                      /* point past "<ESC>"            */
1052:     esc_seq [0] = `\0';             /* null out old string           */
1053:
1054:     i = strtol( ++seq, &seq, 10);   /* get ED option */
1055:     switch(i)
1056:    {
1057:
1058:    case 0: /* erase from cursor to end of the line   */
1059:        biogetcurpos( &row, &col ); /* save present row and column     */
1060:        ntimes = 80 - col;   /* # of blanks   */
1061:        if(ntimes < 0) ntimes = 0;
1062:        bioputch( 0, ntimes, ` ', attr);      /* erase    */
1063:        biosetcurpos( row, col);
1064:        break;
1065:
1066:    case 1: /* erase from beg of line to cursor         */
1067:        biogetcurpos( &row, &col ); /* save present row and column       */
1068:        ntimes = col;    /* # of blanks to put out to video */
1069:        if(ntimes < 0) ntimes = 0;
1070:        biosetcurpos( 0, 0);
1071:        bioputch( 0, ntimes, ` ', attr);      /* erase    */
1072:        biosetcurpos( row, col);
1073:        break;
1074:
1075:    case 2: /* erase all of the line do not move cursor         */
1076:        biogetcurpos( &row, &col ); /* save present row and column       */
1077:        ntimes = 80;    /* # of blanks to put out to VT100 screen      */
1078:        biosetcurpos( 0, 0);
1079:        bioputch( 0, ntimes, ` ', attr);      /* erase    */
1080:        biosetcurpos( row, col);
1081:        break;
1082:
1083:    }
1084:
1085: }
1086:
1087: int openvirt()                          /* func to open virtual input file    */
1088: {
1089:     if( virtfile != NULL) return;
1090:     virtfile = fopen("virtfile.txt", "r");
1091:
1092: }
1093:
1094: int do_virt()                          /* do virtual input - open coded sub*/
1095: {
1096:     static char strptrs [20] [32];  /* host strings to load */
1097:     static char ctrly[2];
1098:
1099:     if(pip == 0)
1100:    {
1101:    ctrly[0] = 25;
1102:    ctrly[1] = 13;
1103:    strcpy(strptrs[17], "" );
1104:    *hostline = NULL;
```

```
1105:   host_cnt = 0;
1106:   }
1107:
1108:     switch(pip)
1109:   {
1110:   case     0: pip += send("DTR"); return;
1111:   case     1: pip += waitfor(hostline, "DTR>"); return;
1112:   case     2: pip += send(":GO_CDAS"); return;
1113:   case     3: pip += waitfor(hostline, "DTR>"); return;
1114:   case     4: pip += send(":PUT"); return;
1115:   case     5: pip += waitfor(hostline, "NEXT CLASSIFIED DOCUMENT");return;
1116:   case     6: pip++;
1117:               if( getdata(virtfile, strptrs) == NULL) pip = -1;
1118:               return;
1119:   case     7: pip++;
1120:               if( strstr(hostline, "Document Number") != NULL) {
1121:                 do_send(strptrs, 0); pip = 7;} return;
1122:   case     8: pip++;
1123:               if( strstr(hostline, "Requester      ") != NULL) {
1124:                 do_send(strptrs, 32); pip = 7;} return;
1125:   case     9: pip++;
1126:               if( strstr(hostline, "Date Ordered   ") != NULL) {
1127:                 do_send(strptrs, 64); pip = 7;} return;
1128:   case    10: pip++;
1129:               if( strstr(hostline, "Ordered From   ") != NULL) {
1130:                 do_send(strptrs, 96); pip = 7;} return;
1131:   case    11: pip++;
1132:               if( strstr(hostline, "Date Received  ") != NULL) {
1133:                 do_send(strptrs, 128); pip = 7;} return;
1134:   case    12: pip++;
1135:               if( strstr(hostline, "Copy Series Num") != NULL) {
1136:                 do_send(strptrs, 160); pip = 7;} return;
1137:   case    13: pip++;
1138:               if( strstr(hostline, "Ordered By(HHL ") != NULL) {
1139:                 do_send(strptrs, 192); pip = 7;} return;
1140:   case    14: pip++;
1141:               if( strstr(hostline, "Date Returned F") != NULL) {
1142:                 do_send(strptrs, 224); pip = 7;} return;
1143:   case    15: pip++;
1144:               if( strstr(hostline, "Loaned To      ") != NULL) {
1145:                 do_send(strptrs, 256); pip = 7;} return;
1146:   case    16: pip++;
1147:               if( strstr(hostline, "Date Loaned    ") != NULL) {
1148:                 do_send(strptrs, 288); pip = 7;} return;
1149:   case    17: pip++;
1150:               if( strstr(hostline, "Date Returned T") != NULL) {
1151:                 do_send(strptrs, 320); pip = 7;} return;
1152:   case    18: pip++;
1153:               if( strstr(hostline, "Returner       ") != NULL) {
1154:                 do_send(strptrs, 352); pip = 7;} return;
1155:   case    19: pip++;
1156:               if( strstr(hostline, "Received Until ") != NULL) {
1157:                 do_send(strptrs, 384); pip = 7;} return;
1158:   case    20: pip++;
1159:               if( strstr(hostline, "Loaned Until   ") != NULL) {
1160:                 do_send(strptrs, 416); pip = 7;} return;
1161:   case    21: pip++;
1162:               if( strstr(hostline, "Record Entry Pe") != NULL) {
1163:                 do_send(strptrs, 448); pip = 7;} return;
1164:   case    22: pip++;
1165:               if( strstr(hostline, "Remarks        ") != NULL) {
1166:                 do_send(strptrs, 480); pip = 7;} return;
1167:   case    23: pip = 7;
1168:               if( strstr(hostline, "Continue Puttin") != NULL) {
1169:                 send("Y");*hostline=NULL;host_cnt=0;pip = 6;} return;
```

```
1170:                 break;
1171:
1172:    default: send(ctrly);
1173:             send("lo");
1174:             virtmode = NULL;
1175:
1176:    }
1177:
1178: }
1179:
1180: int do_send( strptrs, offset)          /* help `do_virt'    */
1181: char strptrs[20] [32];
1182: int offset;
1183: {
1184:     char *place;
1185:
1186:     place = strptrs;
1187:     place += offset;
1188:     send(place);
1189:     /*fprintf(stdprn, " %s %s %s %d %d\n", place, strptrs, hostline,                host_cnt,
1190:                                         offset);*/
1191:
1192:     *hostline = NULL;
1193:     host_cnt = 0;
1194:
1195: }
1196:
1197: int send( string )              /* send "string" and <carr.ret> to host*/
1198: char *string;
1199: {
1200:     char cr_string[2];
1201:
1202:     sendslow(string);
1203:     cr_string[0]=13;
1204:     cr_string[1]= NULL;
1205:     wait(1);
1206:     sendslow(cr_string);
1207:     return(1);
1208:
1209: }
1210:
1211: /* sendslow = func to send chars to host slowly */
1212: int sendslow( string )
1213: char *string;
1214: {
1215:     long i=0, j;
1216:
1217:     while( *string != NULL && i < 80 )
1218:    {
1219:    for(j=0;j<1000;j++);
1220:    if(*string != 10) outp(COMM_DATA, *string++);
1221:    if(*string == 10) string++;
1222:    i++;
1223:    }
1224: }
1225:
1226: int waitfor( string1, string2)          /* see if "string2" in "string1" */
1227: char *string1, *string2;
1228: {
1229:     int val = 0;
1230:     if( strstr(string1, string2) != NULL )
1231:    {
1232:    *string1 = NULL;
1233:    host_cnt = 0;
1234:    val = 1;
```

```
1235:    }
1236:      return(val);
1237:
1238: }
1239:
1240: int getdata( infile, strary)           /* get data from virtual file*/
1241: FILE       *infile;
1242: char       strary [20] [32];
1243: {
1244:      int i;
1245:      char string[96];
1246:      char *place;
1247:
1248:      for(i=0; i < 17; i++)
1249:    *strary[i] = NULL;
1250:
1251:      for(i=0; i < 17; i++)
1252:    {
1253:    if( fgets(string, 90, infile) == NULL ) return( NULL );
1254:    place = strary;
1255:    place += i*32;
1256:    strcpy( place, string);
1257:    }
1258:
1259:      while( fgets(string, 90, infile) != NULL
1260:    && strcmp(string, "*********") == NULL)
1261:     ;
1262:
1263:      return(~NULL);
1264:
1265: }
1266:
1267: int wait( nsecs)                /* function to wait `nsecs' seconds */
1268: int nsecs;
1269: {
1270:      long start, now;
1271:
1272:      start = time();
1273:      now   = time();
1274:      while ( ( now - start ) < nsecs ) now = time();
1275:
1276: }
```

APPENDIX E:

VDE ASSEMBLER LANGUAGE SOURCE CODE

ASCINT.ASM AND DSINIT.ASM

ASCINT.ASM

```
 1:         TITLE   ascint
 2: PAGE ,132
 3:         .287
 4: _TEXT   SEGMENT  BYTE PUBLIC `CODE'
 5: _TEXT   ENDS
 6: _DATA   SEGMENT  WORD PUBLIC `DATA'
 7: _DATA   ENDS
 8: CONST   SEGMENT  WORD PUBLIC `CONST'
 9: CONST   ENDS
10: _BSS    SEGMENT  WORD PUBLIC `BSS'
11: _BSS    ENDS
12: DGROUP  GROUP   CONST,  _BSS,   _DATA
13:         ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP
14: EXTRN   _count:WORD
15: EXTRN   _rsbuffer:BYTE
16: EXTRN   storeds:WORD
17: _TEXT      SEGMENT
18: ;|*** extern int count;
19: ;|*** extern unsigned char rsbuffer[4096];
20: ;|*** ascint()
21: ;|*** {
22:
23: kb_flag_1       equ     18h
24: hold_state      equ     08h
25:
26:         PUBLIC  _ascint
27: _ascint       PROC NEAR
28: push     ax                                 ;save registers we use
29:         push    bx
30:         push    dx
31:         push    ds
32: mov      ax,cs:storeds                     ;get VDE's data segment...
33: mov      ds,ax                             ;...stored by dsinit...
34: ;|***      rsbuffer[count++] = inp(COMM_DATA);     ;... so can get          `count'
35: mov      dx,3f8h                  ;...address of COM1 data reg
36: in       al,dx                             ; get character
37: mov      bx,_count                         ; get circular buffer pointer
38: inc      _count                            ; inc pointer
39: mov      BYTE PTR _rsbuffer[bx],al         ; stick char in buffer
40: mov      ax,_count                         ; get ptr again
41: and      ax,4095                  ; wrap it at 4095 to zero
42: mov      _count,ax                         ; thru with actual get char
43: mov      ax,40h                            ; address segment at 400 hex
44:         mov     ds,ax
45: mov      bx, kb_flag_1                     ; get keyboard ISR flags
46: and      byte ptr ds:[bx], not hold_state; disable CTRL-BREAK
47: mov      al,20h                            ; reset interrupt...
48: out      20h,al                            ; ...chip
49: pop      ds                                ; restore regs
50:         pop     dx
51:         pop     bx
52:         pop     ax
53: iret                                       ; bye till next rs232 interupt
54: _ascint ENDP
55: _TEXT   ENDS
56: END
```

DSINIT.ASM

```
 1:         TITLE   dsinit
 2: PAGE ,132
 3:         .287
 4: _TEXT   SEGMENT  BYTE PUBLIC `CODE'
 5: _TEXT   ENDS
 6: _DATA   SEGMENT  WORD PUBLIC `DATA'
 7: _DATA   ENDS
 8: CONST   SEGMENT  WORD PUBLIC `CONST'
 9: CONST   ENDS
10: _BSS    SEGMENT  WORD PUBLIC `BSS'
11: _BSS    ENDS
12: DGROUP  GROUP   CONST,  _BSS,   _DATA
13:         ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP
14: EXTRN   _count:WORD
15: EXTRN   storeds:WORD
16: _TEXT      SEGMENT
17:         PUBLIC  _dsinit
18: _dsinit       PROC NEAR
19:         push    ax
20: mov      ax,_count        ; count presently has its (&VDE's) data segment
21: mov      cs:storeds,ax    ; store ds in code segment for ascint use
22:         pop     ax
23:         ret
24: _dsinit       ENDP
25: _TEXT   ENDS
26: END
```

GLOSSARY

4GL - Fourth Generation Language - a very high level language for applications programming. Such a language is more sophisticated than third generation languages like FORTRAN, C, dBASE, or Cobol. The 4GL of most DBMSs combines SQL and high-level constructs or extensions including access to the operating system, set functions, Boolean functions, arithmetic operators, string operators/functions, and report generation functions. 4GLs are not standardized and are usually vendor-proprietary.

ADP - Automated Data Processing - computers, networks, software, etc.

ANSI - American National Standards Institute.

CALS - Computer-Aided Acquisition and Logistics Support - a DoD program that will use the OSI protocol suite to let the military electronically access maintenance and specification information including graphical database forms.

CASE - Computer-Aided Software Engineering - modern software tools for software planning, design, analysis, and maintenance based on popular methodologies of DeMarco, Gane and Sarson, and Yourdon, among others. The CASE tools generally include diagramming tools, screen and report painters for specification and prototyping, dictionaries, specification-checking tools, code generators, and documentation generators, among others. CASE is generally utilized on large and/or complex software projects. Distributed database system applications developments are likely candidates for CASE

Configuration Management Software - software used to track and control software system development for activities such as tracking what changes were made to software, and when and why these changes are made. By identifying and controlling all versions of an application throughout its life cycle, earlier versions can be easily reconstructed while parallel versions are still maintained.

Data Dictionary - a means to specify both the DBMS contents and the integrity constraints that apply to each data element. Traditionally the Data Dictionary is used to protect consistency of the database. Data Dictionaries offer features such as structured storage media, automated documentation, integration of separately developed projects, impact analysis, and change control for maintaining live systems. Standards are under development such as Information Resource Dictionary System (IRDS).

DBMS - Data Base Management System - a software system for data storage, retrieval, and presentation usually composed of elements such as the user interface, screen painter, report generator, query optimizer, 4GL, network-communications interface, and data manager/engine. The DBMS's may be based on hierarchical, relational, codasyl, or object models of design.

DEC - Digital Equipment Corporation.

DEC/ACMS - Application Control and Management System.

DECnet - a communications and file transfer protocol for LAN's and WAN's provided by DEC. The first version of OSI was essentially a copy of DECnet.

Ethernet - CSMA/CD, a carrier sense multiple access collision detection protocol for networking. The standard is IEEE 802.3.

Expert System - software that uses a rulebase and a database to infer (compute) conclusions. It is the dominant application for artificial intelligence, at present.

GOSIP - Government Open Systems Interconnection Profile - a developing protocol standard that will replace, TCP/IP. It is the U.S. Government's implementation of the developing interconnection standard, OSI. Two current shortcomings of GOSIP are its deficient gateway-to-gateway protocol and its lack of a virtual terminal capability.

LAN - Local Area Network - a local area collection of computers that are connected by either baseband coaxial cables, or twisted-wire pairs, or broadband coaxial or fiber optics cables. The LAN may have a star, ring, or distributed bus network topology. The access rules may be of many types including token passing (Token Ring, IEEE 802.5) or carrier sense multiple access including collision detection (Ethernet IEEE 802.3), as examples. Communications and file transfer protocols may include DECnet and TCP/IP.

MS-DOS - the original single-tasking operating system for the IBM PC or clones. It also runs on the PS/2. MS stands for Microsoft.

NFS - Network File System of SUN Computer, Inc. - NFS is a de facto file transfer networking standard supported by more than 60 vendors. The software allows remote files on servers to be accessed over a network as though the files were present on the local node's disk system.

OS - Operating System.

OS/2 - the new multitasking operating system developed by Microsoft for the IBM PS/2 or for other AT-class (or higher) PCs.

OSF - Open Software Foundation - a consortium of private companies including IBM, DEC, HP, Apollo, and many others with the goal of providing open, vendor-independent software.

OSI - the seven-layer Open Systems Interconnection reference model of the International Standards Organization.

POSIX - Federal computing interface standard FIPS 151 based on IEEE 1003.1 - developed to promote applications portability across heterogeneous hardware and OSs. POSIX provides a migration path of existing UNIX applications to non-UNIX OSs.

RDBMS - Relational Database Management System - advanced form of DBMS where all data are stored in relations and manipulated using relational operations such as union, intersection, and difference.

SNA - System Network Architecture of IBM - standard for the design and fabrication of communication products. SNA controls the protocol used in IBM communication networks (a form of SDLC, Synchronous Data Link Control). DEC has a DECnet/SNA gateway allowing direct channel attachment from DECnet LANs to IBM mainframes.

SQL - Structured Query Language - an advanced language that has become an ANSI standard. Grown out of the theories of relational calculus, the language consists of numerous commands for adding, deleting, modifying, and reporting information/data.

TCP/IP - Transmission Control Protocol/Internet Protocol - a mature communication protocol that has become an industry standard. TCP/IP was originally developed by the Defense Advanced Research Projects Agency (DARPA) for its ARPANET communications network in the 1960s. It is the protocol of the Department of Defense Data Network (DDN). TCP/IP runs on most vendors' hardware platforms and under many different operating systems.

UNIX - a dominant multitasking operating system originally developed at AT&T Bell Laboratories. UNIX has the distinction of being able to run on most vendors' hardware platforms and is becoming increasingly mandatory in the federal government.

VMS - the dominant DEC operating system for the VAX.

VT100 - a terminal standard based on DEC terminal hardware.

WAN - Wide Area Network.

X/Open - a common applications environment consortium that brands products as conforming or nonconforming with computing standards.

X Windows - a window system for multitasking operating systems; developed at MIT with participation of DEC and other workstation vendors. X-compliance allows applications the portability and network-transparency required for open systems. It is the de facto window-management standard, particularly for UNIX workstations.

INTERNAL DISTRIBUTION

1. T. D. Anderson
2. J. M. Barnes
3. A. L. Beckwith
4. M. A. Bjerke
5. P. G. Bohanan
6. D. W. Bradford
7. W. A. Bratten
8. B. K. Bryan
9. R. D. Burris
10. D. N. Clark
11. L. A. Clinard
12. J. M. Corum
13. J. G. Craven
14. G. A. Dailey
15. C. O. Doty
16. L. D. Duncan
17. R. C. Durfee
18. M. L. Emrich
19. W. E. Ford III
20. V. M. Forsberg
21. J. F. Francis
22. S. J. Freeney
23. C. E. Hammons
24. B. H. Handler
25. R. E. Haney
26. D. S. Hartley
27. P. B. Hartman
28. M. B. Heath
29. O. W. Hermann
30. D. M. Hetrick
31. B. M. Horwedel
32. J. E. Hough
33. R. L. Huddleton
34. G. L. Johnson
35. K. L. Kannan
36. K. L. Kruse
37. D. K. Lee
38. R. P. Leinius
39. A. S. Loebl
40. R. C. Lushbaugh
41. G. S. McNeilly
42. J. R. Merriman
43. J. D. Morris
44. B. J. Noble
45. L. W. Owen
46. M. R. Patterson
47. C. E. Penland
48. R. W. Reid
49. G. D. Robbins
50. M. C. Salmons
51. D. D. Schmoyer
52. F. L. Sexton
53. K. E. Shaffer
54. V. A. Singletary
55. C. E. Snyder
56. B. Thomas, Jr.
57. R. E. Tittsworth
58. H. E. Trammell
59. V. M. Voorhees
60. C. F. Weber
61. G. W. Westley
62. G. T. Yahr
63-72. J. A. Clinard
73-82. J. T. Phillips
83-92. J. J. Robinson
93. ORGDP Plant Records
94. ORNL Records
95. ORGDP Plant Records--RC
96. ORNL Records--RC
97. Enrichment Technology Library
98. Central Research Library
99. DSRD Library
100-102. A. S. Quist

EXTERNAL DISTRIBUTION

103-112. R. Blackburn, Pacific Missile Testing Center, Code 0300, Point Mugu, CA 93042
113. F. Malabarba, Pacific Missile Testing Center, Code 0300, Point Mugu, CA 93042
114. Office of Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations Office, Oak Ridge, TN 37831