PROVIDING TEXT-MODE ACCESS TO BLIND USERS WITH AN APPLICATION USING TEXT-TO-SPEECH SYNTHESIS

By

AOUNI HALLAL

Bachelor of science

Oklahoma State University

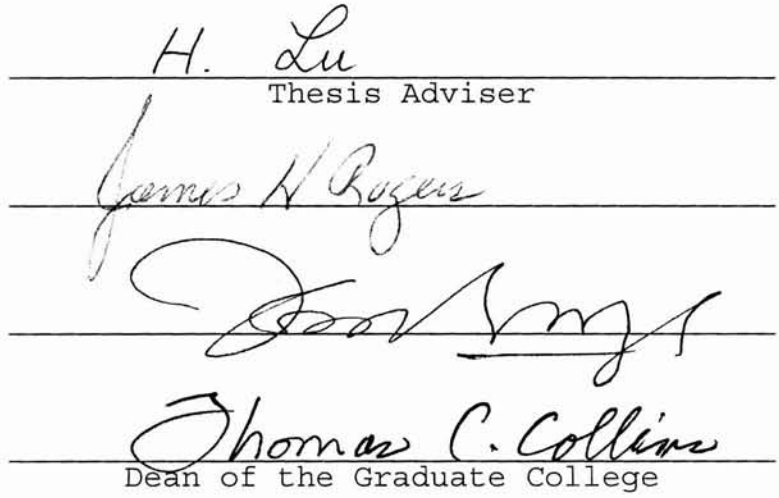
Stillwater, Oklahoma

1989

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTER OF SCIENCE
May, 1995

PROVIDING TEXT-MODE ACCESS TO BLIND USERS WITH AN APPLICATION USING TEXT-TO-SPEECH SYNTHESIS

Thesis Approved:



ACKNOWLEDGMENTS

I would like to express my sincere gratitude and appreciation to Dr. Huizu Lu for the opportunity to work under her supervision. Her advice and ongoing encouragement were invaluable in guiding the completion of this work. Many thanks are also due to the members of my advisory committee, Dr. K. M. George, and Dr. Jim Rogers for their generous support and constructive recommendations.

I also like to express my thanks to all the people on the misc.handicap newsgroup for their suggestions, to Ms. Susan Haaze of the Wellness Center for allowing me to use the adaptive equipment, and to all my friends and teachers that I have met during my college years.

Special thanks are due to my grandparents, my mother and father, and my brothers for their loving concern, selfless support, and strong encouragement at times of difficulty. To them I dedicate this thesis and I extend my deepest appreciation and love.

And above all, I thank God.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

ASCII	American standard code for information interchange
B	Binary representation
BG	Background screen color
BIOS	Basic input/output system
DOS	Disk operating system
FG	Foreground color
H	Hexadecimal notation
Kbyte	Kilo byte
MB	Mega byte
PAL	Profile access language
PC	Personal computer
RAM	Random access memory
ROM	Read only memory
TSR	Terminate and stay resident program
UI	User interface

CHAPTER I

INTRODUCTION

The underlying effort in the evolution of user interfaces (UIs) has been to facilitate the communication between the user and the machine. Most current UIs are either the traditional alphanumeric full-screen terminals with a keyboard and function keys, or the more modern WIMP interfaces with windows, icons, menus, and a pointing device [JAKO93].

The visual channel has a tremendous capacity for information transfer, and it has been heavily dependent upon in communicating the interface to the user [VAN89]. The transformation of this interface into non-visual channels to accommodate the blind community has raised many challenging design issues.

One's sight can quickly navigate or scan any part of the display, screen colors and text attributes communicate visually important information to the user, boxes tend to cluster and organize data in a visually appealing manner, and the location of screen objects enhance the screen layout and convey different information depending on their locality. An access software must substitute these tasks for the blind user which sighted users take for granted.

Currently, numerous application programs run under text-mode environments which are mainly popular under DOS operating systems. Unfortunately, there are practically no standards that these applications follow, and their user interfaces are highly dependent on the visual channel [TED92]. For instance, some applications use the whole screen to present a menu while others utilize an action bar with drop-down menus, and the location of these menus on the screen also differs widely from one program to another.

Consequently, there are many technical difficulties involved when translating the user interface into non-visual modalities [VAND92]. In addition, there are various human factors that are needed to be addressed in order for these adaptations to be effective solutions [DOUG90]. For instance, unlike sighted users who can glance at the screen to refresh their memories, blind users depend on short-term memory to form a mental model of the screen display. Therefore, any solutions has to provide sufficient functions which allow the blind user to form the proper mental model of the interface and without overloading his/her memory.

Chapter II reviews auditory adaptations, and briefly covers tactile and multi-sensory access systems. Chapter III presents major personal and technical constraints faced in the design of speech audio interfaces, and explores a group of user control facilities needed in order to provide efficient exploration and translation of the interface.

Chapter IV discusses the implementation mechanism of collecting the data for the access program, and the steps used to coordinate speech with the current screen activity, the keyboard state, and the disk drive status. Chapter v gives a summary and some recommendations for future research.

CHAPTER II

CURRENT ACCESS SYSTEMS

To provide non-visual access and regardless of the non-visual output modality, information about the running application user interface must first be captured. The retrieved information is then used to present a different, non-visual interface of the running application [BETH94b]. This new interface becomes the means with which the blind user interacts and controls the application.

Once the information is captured, presenting the user interface in a non-visual modality can take many forms. The available domains for a severely visually impaired user are the tactile and auditory channels. Consequently, access solutions have addressed [CARL92] [LAZZ93]:

1. The auditory mode using screen reading software which is normally combined with a text-to-speech synthesizer;
2. The tactile mode using mainly dynamic Braille displays;
3. Both channels by using a combination of sound and touch.

2.1 Scope

The implementation program for an auditory system is limited to a commercial text-based database system called Alpha4. Text-to-speech synthesis is the medium used to

produce the output. However, the details of the process of converting text into speech are beyond the scope of this paper; good reviews are found in [OMAL90] [HIRS90] [LEON91].

2.2 Auditory Adaptations

Auditory access systems include speech synthesis, auditory cues, and spatialized audio sounds.

2.2.1 Speech Synthesis

Historically speech synthesis technology has been one of the major factors which influenced non-visual user interface adaptations. Currently, synthesized speech is one of the most powerful and least expensive computer access devices for the blind user and improved algorithms and more powerful signal-processing chips are resulting in high quality speech [OMAL90] [LEON91] [LAZZ93]. As a result, the majority of access solutions thus far have utilized speech as the principle format of communicating the visual interface to the blind user.

Early commercial applications produced speech output by concatenating a sequence of pre-recorded words, syllables, or phonemes. A phoneme is the smallest unit of speech that distinguishes one utterance from the other [LEON91]; usually found in dictionaries to dictate how words are pronounced. This approach was most useful for systems requiring a relatively small output vocabulary such as automated

telephone directory assistance. Its disadvantages included the difficulty of handling voluminous or changing information [ELIM90]. In addition, the scope of words that can be produced is limited to what has been recorded. The way a sample is recorded determines the way it will sound when played back. All the samples must be recorded and stored in advance which gives little dynamic control over UI factors such as speech inflection, length of pauses, and speaking rate. These constraints have limited the flexibility of stored voice response systems, and in turn have affected the efficiency of the auditory user interface.

Text-to-speech synthesis systems, on the other hand, operate directly from an input text or data stream to produce a real-time, comprehensible speech instead of using pre-recorded human speech. These systems take arbitrary text as input together with optional user specified commands to control the system parameters (such as phrasing, prominence, and rate) and produce real-time synthetic speech. The process of converting text into speech parameters involves advanced information processing and detailed physiological control of articulatory organs. Text-to-speech synthesis approach has the advantage that the variety of words which can be produced is much greater than those produced when using the concatenation method; it allows the production of an unlimited vocabulary. The disadvantage is that the speech produced tends to sound less

natural than that produced when using concatenation.

To apply this auditory solution, a user has to install additional hardware and software on the workstation. The hardware is a speech synthesizer which can be either a circuit card or an external peripheral device. The software is a screen reader which controls the speech hardware and verbalizes text.

Typically, a speech synthesizer is a dedicated computer with the single task to run text-to-speech software. Current speech synthesizers vary in sophistication. Some provide means for the user to control the number of words spoken per minute, to simulate a variety of different voices (male/female and young/old voices), and to change voice characteristics such as pitch and volume [LAZZ93]. Whereas, screen readers are software solutions that are coresident in memory with the application program. They link the speech synthesizer with the computer's operating system so that the visual interface and input devices become verbally interactive and controlled by the user.

2.2.2 Auditory Cues and Spatial Audio

Some research have been carried out into communicating the UI information through sounds other than speech by using auditory cues [EDW89] [BGB91]. Auditory cues, first introduced by Gaver in a project called SonicFinder [GAV89], are mainly used to reinforce single-sensory solutions by

providing informative non-speech sounds which represent the interface components. They range from single audible beeps to sounds that are associated with everyday objects. For instance, touching a window sounds like tapping on glass, touching a text field sounds like an old fashioned typewriter, and searching through a menu creates a series of shutter sounds.

Much of the current research in non-speech audio interfaces has been based on mapping attributes of screen objects into parameters of sound. By adding sounds to the interface to provide system information, the interface bandwidth of communication can be significantly increased. However, the ultimate success of auditory cues depends on the development of good analogies between events in the computer and sound producing events in the real world. The mapping is not straight forward for the UI components such as menus and dialogue boxes which are abstract notions and have no innate sounds associated with them. Further, too many sounds can be detrimental in being difficult to remember or distinguish, and tend to slow the interaction [BETH94a]. In general, auditory cues have been used to communicate information quickly whereas speech have been used to give more precise information more slowly.

Research has also proven that the integration of real-time spatial audio can further enhance the non-visual interface [WENZ90] [BURG92]. Spatial audio to access

solutions can convey the locality of objects on the screen. With this approach, the words being read sound like they are coming from the location on the screen where they are located. As the user reads from left to right, the voice seems to float from left to right [GAVE89] [LUDW90] [LUDW91]. However, navigation in speech only interfaces remains a challenging design problem. Audio feedback does not provide an easy way for users to keep track of their position in the interface [BETH94a].

2.3 Tactile Access Systems

Dynamic Braille displays can be an alternative or a complement to voice output. When the Braille display receives text, it automatically displays it by the raising and lowering of pins on a strip that contains mechanical Braille cells. If the user presses a key on the keyboard or if the screen is updated, one or more of the cells are activated automatically. However, present commercial dynamic braille displays are constrained to only one line of braille output of either 20, 40 or 80 characters [LAZZ90] [DANI92] [VAN92] which have limited the efficiency of such adaptations.

The single line braille display does not yield ready access to information which is two-dimensional or screen oriented such as spreadsheets [HINT92]. In addition, since tactile provides a passive way of interaction, it is a

difficult medium to quickly alert the user [DANI92]. To overcome some of these liabilities, numerous efforts have addressed the implementation of full-page Braille. However, cost factors have rendered such efforts to be impractical [DANI92] [BETH94b]. Dynamic Braille displays are currently used in conjunction with voice adaptations for reviewing text documents such as reading source code since speech synthesizers have been notoriously bad at this task. Moreover, adding Braille to voice output has been most effective since redundancy is introduced to the user interface.

Research efforts have further demonstrated that tactile when combined with speech output can provide access to simple graphics, diagrams, and charts. A prototype called "System3" is a cooperative research and development project of the Trace R&D Center and Berkeley Systems . It is used with a screen reader to produce a multi-sensory access system. System3 presents tactile images of text or simple graphs to the blind user through a set of vibrating pins atop an absolute reference pointing device. This is significant in that, unlike the mouse, the relative position of the pointing device on the tablet corresponds directly to the position of the pointer on the screen. The user can feel under a fingertip the vibrating image of the pixels under the pointer as it moves on the screen, and piece together mental images of where things are on the screen

[VAN90] [VAN92] [TRAC94]. This effort aims at improving the functionalities of scanning, browsing, and memory jogging which are not efficiently performed when using a single sensory adaptations, such as speech or Braille alone.

A great milestone has been achieved in this field; however, the present non-visual adaptations by no means have efficiently solved access for blind users. This has been largely due to the limited experience with non-visual domains [BETH94a].

CHAPTER III

DESIGN CONSIDERATIONS FOR SPEECH AUDIO

Beyond the process of translating text into speech, there are a number of considerations which must be taken into account if an auditory system is to function as an effective tool for the blind user. The development of these access solutions have been restricted by various technical and personal constraints. Furthermore, a group of user-controlled functions have to be provided in order to allow an efficient exploration and translation of the user interface.

3.1 Design Constraints

Conveying the characteristics of the different aspects of the interface in an efficient non-visual form has not been an easy proposition. The auditory system has a higher information bandwidth than the tactile senses but not as high as the visual system. Thus, the non-visual interface has to be less complex, and able to achieve as many of the benefits that are provided in the visual interface [BETH94a]. Unfortunately, there are no non-visual media that are able to convey as much information due to the serial nature of access solutions as opposed to the parallel

capacity of sight [EDW88]. Therefore, additional functionality has been required to support efficient exploration of non-visual interfaces.

Persons with visual impairments have limited real-time access to computer information; thus, time has been one of the most valuable commodities. Short-term memory and strong concentration have also been important since blind users lack the means available to sighted people to check the screen and refresh their memories [EDW89]. Consequently, a highly relevant design issue for auditory solutions has been that feedback must be brief, yet informative in order to conserve time and to reduce the information that the user has to retain in memory. Nonetheless, all access solutions utilize procedures that add extra steps to the interface of the running application. Procedures that are automatically performed by a sighted user can require substantial processing resources for blind users [DOUG90]. For instance, a sighted user can determine at a glance the physical location of the cursor on the screen and its contextual location. On the other hand, at least two steps would be required for the blind user; one to find out the cursor location and the other to read the word at the cursor location.

An overriding aim in any access system has been to maintain coherent visual and non-visual interfaces. The primary reason for this goal has been to provide the blind

user with a reasonably similar mental model of the running application as that of a sighted user in order to support collaboration between them [BETH94a] [PAUL92]. Furthermore, access solutions need to be transparent to running applications [BROWN89] [BETH94b]; that is, when an application is running, it must be unaware of the presence of the non-visual system. This has been a highly relevant design factor since the access solution would not require the recoding of the application itself to support the access system. If a direct modification approach were to be used, it would be possible to introduce the mechanisms for the non-visual interface directly into the application. The drawback to this latter approach is that it requires modification to every application of interest. This is not feasible for a practical access system. Therefore, all access systems are Terminate ans Stay Resident programs (TSR) in order to be used concurrently with a running application [LAZZ90]. However, there exists no solution that provides access to all text-based application due to the fact that there are practically no standards followed by DOS application developers [TED92]. For instance, actions bars, cursor shapes, and error messages are not consistent amongst applications. The more complicated screens and non-standard conventions an application utilizes, the harder and more complex a customization becomes [EDW88] [VAND92].

Navigation and data manipulation can be communicated

entirely via a keyboard. Pointing devices such as mice and trackballs have been added to text-based application user interfaces; yet input can be communicated exclusively through the keyboard as blind users can be touch typists. The Keyboard, however, has been overloaded since it is heavily used by both the application program and the access system [EDW88] [TED92]. The keyboard has a limited number of keys which are already used up performing the application functions. There are usually few keys or key-combinations that are available for the access system functions. Some adaptations have provided an extra keypad in addition to the keyboard to get all the keys needed. However, this has added to the complexity of the solution since the user has to remember which keys to press and their respective locations.

To reduce the keyboard activity, access solutions have utilized macros which in turn reduce the memory required to control the computer and the access software [TED92]. Macros have traditionally been used as keyboard enhancers by combining keys and functions together into one key. In access programs, macros have been similarly used to relieve the user from entering repetitive commands and to watch the screen for any changes that may occur. For example, the screen reader can be instructed to watch the bottom line of information on the screen and whenever the line turns from blue to red, the screen reader macro is instantly triggered.

3.2 Screen Readers Major Control Facilities

When a screen reader is chosen to provide access, synthesized speech is the medium used to translate the interface to the blind user. Any screen reader has the responsibility of conveying the screen activity to the user, the job usually done by the eyes. Many different screens appear when an application program is running, the keyboard keys are used to perform several different functions, and the action of those keys depends on the mode that the application is in. For instance, pressing the left arrow key may take the cursor one character to the left or may move it to a previous field; or pressing the Enter key may produce a different result based on whether the application is in edit or command mode. Therefore, a key component of any adapted system is the control facilities provided to convey the screen information.

3.2.1 Verbalizing the Active Point

In order to convey the correct screen activity, screen access programs have to track and verbalize the active point on the screen so that the blind user can tell where the keyboard action will take place [JIM93]. The system cursor is usually the active point. If the application uses the system cursor, its information such as position, shape, and color is held in registers in the display hardware. On the other hand, if the application creates its own cursor such

as highlighting text, there has been no way for the access program to easily tell where the cursor is [VAND92].

Some applications display a list of commands on the screen and expect the user to be able to notice that one of these commands is distinguished from the others because it is highlighted in a different video enhancement or color. This area of text is usually known as a lightbar or selector. Users are often required to move this lightbar up, down, left, or right on the screen with the cursor arrow keys. In these situations, the lightbar replaces the cursor. If a screen reader only verbalizes where the system cursor moves, it would not be possible for the user to read the lightbar text. In these cases, some screen readers attempt to trace a set of video enhancements or color combinations that make up the lightbar [IBM92a] [WB93]. However, there are occasions where several lightbars may appear on the screen. For instance, many applications use the same lightbar attributes to highlight a title, or a status line . As a result, it is possible to verbalize the wrong lightbar instead of the desired lightbar. Two strategies have been suggested by Vanderheiden [VAN92] to mainstream software developers in order to facilitate tracking lightbars by screen access programs:

1. To drag the system cursor along with the lightbar, or
2. To carry a character along with the lightbar such as

[A.] Save or ▸ A. Save

3.2.2 Handling Window Frames

Many software programs use window frames to enclose a list of commands such as a menu, or to divide a split screen for example. A text-mode window frame refers to an area on the screen that is surrounded by a graphic border. In many cases, this window appears over existing text, and is enhanced to the sighted user by creating a graphic border around the text in question. Another common property of these window frames is that they do not always appear in the same location on the screen. In these cases, screen readers have created reading boundaries so that only the text contained within these windows is verbalized and not the whole screen [IBM92a] [WB93].

There are further some situations where the text appearing on the screen cannot be accessed by the system cursor. This is because the application program does not require a sighted user to move this cursor into these areas. These are usually referred to as protected areas. For instance, line 25 in many word processing programs is in protected mode since the system cursor cannot access this area of the screen.

In order to permit the blind user to read these protected areas, most access solution create another cursor or pointer [BROWN89] [IBM92b] [ERIC92]. This pointer is controlled in the same manner as the system cursor, but unlike the system cursor it is not visible on the screen to

the user. This cursor is referred to as scan cursor or review mode cursor. The review mode suspends the application program so that the user can explore with this pointer the current contents of the video buffer. Therefore, when in this mode, attempting to edit the contents of the screen, or to execute a command for interpretation by the application program will not be recognized or executed.

Most of the control facilities that are provided with screen readers are executed by the user from within the review mode. A user is able to control and set any of the speech synthesizer parameters or any of the screen reader operating parameters. This mode has also been beneficial for programs that do not permit the user to redisplay the current screen information or to navigate through it with the keyboard arrow keys. A user can freeze the current screen by entering the review mode and can utilize the provided facilities with the screen reader to explore it [LAZZ93].

3.2.3 Cursor Routing Facility

When in review mode, the scan cursor may be moved to a location on the current screen where the user may wish to make a change in the screen content. Since the user cannot enter text, or edit the current screen contents from within this mode, the user must leave the review mode before making

any changes on the screen. To facilitate this process, screen readers have provided a routing facility [BROWN89] [IBM92b].

The concept of cursor routing instructs the screen reader to move the system cursor to the location of the scan cursor. The only limitation to this is attempting to move the system cursor into a protected area of the screen. A screen reader would leave the review mode, and attempt to move the system cursor to this location. If this operation is successful, the system usually informs the user in some manner that the cursor has been routed. Otherwise, the user is told that the task has failed, and the screen reader attempts to position the system cursor on the closest accessible line or column to the desired location on the screen [WB93]. For example, trying to route the cursor to Line 25 will position the cursor on Line 24 if Line 25 is inaccessible.

3.2.4 Automatic Monitoring Facility

To alert the blind user of information such as error or status messages that are directly sent through the operating system to the display, a screen reader is programmed to automatically verbalize such messages. Although this provides transparent access to some programs, most application programs bypass the operating system when displaying output and do not organize their screen displays

into a scrolling stream of text [ERIC92]. Therefore, screen readers have provided another facility which automatically monitors the screen and announces any changes that may take place. All the data to be spoken is event actuated, the screen reader should notice the new information, decides what needs to be done or spoken and perform accordingly [TED92]. For instance, the status line in Word Perfect is usually one line of information which must be automatically spoken when it changes; however, it is sometimes two lines and the screen reader should verbalize both.

3.2.5 Emulating Visual scanning

To mimic the sighted user's visual scanning abilities of the interface, screen readers have provided a set of Hot keys or Read Areas [BROWN89] [TED92]. By entering a certain keystroke(s) or a Hot key, the user can instantaneously read any section of the screen, or perform various navigational tasks that are usually done by a sighted user, such as verbalizing the status bar, the cursor location, or the menu entries.

To further emulate most of the actions performed in a visually oriented reading process, screen readers have provided functions to allow the user to read single letters in order to identify possible spelling mistakes, words in order to orient within a line of text, and lines of text in

order to perceive the contents of a screen. In addition, just as sighted users skim documents to obtain a perception of overall coherence, screen readers have provided blind users with options which allow these same processes to occur such as reading a full screen of information or parts of it [CARL92].

3.2.6 Read Requests and Searching

A user is always in either the application mode or the review mode. When in application mode, any read request would be relative to the system cursor or the active point. For instance, if a command is issued to read the current line, the line containing the system cursor would be read, and the system cursor is not moved from its location. On the other hand, when in review mode, any read request is relative to the scan cursor, and the scan cursor moves to the last item spoken. For instance, if the user is reading a character at a time, the scan cursor stops at the last character read. If reading words, the scan cursor stops at the beginning of the last word read. However, if reading entire lines, the scan cursor movement is dependent on the speech synthesizer in use. If it has indexing, a way for the synthesizer and screen access program to keep track of the words spoken, the scan cursor moves and stops at the beginning of the last word spoken. Otherwise, when reading lines or entire screens, the scan cursor remains on the

beginning of the first spoken line [IBM92b] [ACC91].

There are times when the user is presented with a screen of information that may be time consuming to read in its entirety in search of a specific command, word, and so forth. If the application program being used does not offer a find feature to locate the desired text, it could be rather time consuming for the blind user to read the entire screen in search of this text. A find feature should be available so that a user can search the entire screen for the desired information [LAZZ93] [WB93] [IBM92a].

3.2.7 Other Control Features

A critical and important function that a screen reader has to provide a user is the answer to the following: "Where am I in the application?"
The response to this question has varied widely across applications; however, it should provide a point of reference and direction so that the proper mental model is formed by the user. For instance, the answer may include the number of windows present on the screen, the active window, the current mode, the current location of the system cursor, and so on. The system cursor position is obtained from the ROM BIOS and is usually spoken to the user in terms of its screen coordinates by row and column. However, this may not convey the proper current location in the application to the user. This would take place when the

system cursor is on one page, and the user is scanning a different page. Therefore, announcing the cursor position is not enough to convey the user location in the application. The only way of notifying the user of the proper message is when the application utilizes indicators that shows the active page [JIM93].

The majority of screen readers have provided a facility in their interface to temporary or permanently silence speech [BROWN89]. The main reason for the temporally silence feature is to allow the user from having to wait for the speech to catch up with the screen display during keyboard entry, output of a monitor area, or cursor movement and reading. This allows the user to quickly move the cursor without any lag time while the voice attempts to finish any remaining text. For instance, if a user is moving through a document line by line; without this feature, each time the cursor or pointer is moved down or up, the user has to listen to the entire line before hearing the next one. The permanent silence is also important in that the station can be used by a sighted user.

Most keyboards have lights to indicate to the sighted user the status of the lock keys which include Num Lock, Caps Lock, and Scroll lock. Furthermore, the Insert key can be toggled to Replace mode in many applications. Screen readers have provided a verbal method of indicating the status of those keys to the blind user whenever they are

changed. Some applications; however, provide their own on-screen indication whether or not those keys are turned on. In some cases, this feedback is independent of the flags in the system or the status of the lights on the keyboard. This situation have resulted in an inconsistent feedback to the blind user when an access program tends to check the status of these indicators [VAN92].

The screen reader operating parameters should further allow the user amongst many other things to filter out unwanted symbols as they appear on the screen via BIOS writes from being read, to control the enunciation of numbers (which can be spoken in terms of hundreds and thousands or one digit at a time), and to control the upper or lower case sensor (by directing the synthesizer to raise the pitch of the voice or if the subtle differentiation is not enough, a verbal distinction can be made by speaking "capital" or "upper" when one is detected) [IBM92a] [IBM92b] [WB93]. A relevant factor which affects the speech output is whether the speech synthesizer in use accepts the full ASCII set of characters or a portion of it. Some speech synthesizers only accept the lower portion of the ASCII set of characters which constitutes the first 128 characters. In the latter case, when a character on the screen whose ASCII value is over 128 is encountered by the speech synthesizer, a word such as "graphic" would be spoken or the ASCII equivalent value of the character is spoken.

CHAPTER IV

SCREEN ACCESS IMPLEMENTATION

The screen access program is a Terminate and Stay Resident program (TSR) which consists of a collection of functions that are automatically triggered when a change takes place on the screen, and a set of key sequences that are invoked by the user to perform a variety of tasks. Changes on the screen range from the movement of the cursor or a highlight bar to the appearance of a totally new screen of data. The key sequences are only invoked when the user is in review mode; this is when the running application is suspended and the access program is activated.

4.1 Text-Mode Screen Architecture

The screen of a personal computer (PC) is divided into a matrix of character cells. Most display adapters offer two text modes: one with a matrix of 40 columns by 25 rows, and the other (the default) with a matrix of 80 columns by 25 rows [ROBB91]. The cells are not visible, they just mark possible display locations on the screen. Each cell is identified by its row and column coordinates; the coordinates of the upper left cell are (0,0). On a screen that has 80 columns by 25 rows, the coordinates of the lower

right cell are (79,24).

In each cell, the display adapter can only display one of a predefined set of characters. With a color adapter, a character is made up of 5 X 7 dots within an 8 X 8 cell, and the pixel pattern for each character is stored in ROM on the display adapter. As a result, when a program wants to display a certain character, it does not have to specify the character's pixel pattern. Rather, a program needs to specify the character's identity which is the ASCII code. The display adapter hardware retrieves the corresponding pixel pattern from ROM and displays it on the screen [AITK92].

This set of pixel patterns includes letters, numerals, punctuation marks, special symbols, and other characters. These codes range from 0 to 255, representing the 256 possible combinations of binary digits contained in an eight-bit byte. A programmer can only display those predefined characters in the matrix of character cells on the screen. The ASCII codes to each letter or symbol can be divided into the following:

0 to 31 are Control codes. Generally, sending one of these codes will cause something to happen instead of causing a symbol to be displayed. For example, displaying code 13 will cause a carriage return.

32 to 127	are the fundamental 96 text characters. They include numbers, letters, and all the punctuation symbols.
128 to 175	are foreign language characters and few other miscellaneous characters.
176 to 178	are the 3 characters used for shading.
179 to 218	are line segment characters; forms and tables can be constructed on the screen by combination of these characters.
219 to 223	are the block graphic characters.
224 to 255	are Greek letters and mathematical symbols.

Furthermore, MS-DOS was developed on the 8086/8088 central processing unit, which can address a total of 1 megabyte of memory. The typical uses and location of this memory are shown in Figure 1. There are 16 segments and each segment's size is 64-Kbyte. Microprocessors developed after the 8088 can access memory above 1 MB (FFFFFH); However, for the present discussion, only the memory below FFFFFH is of interest.

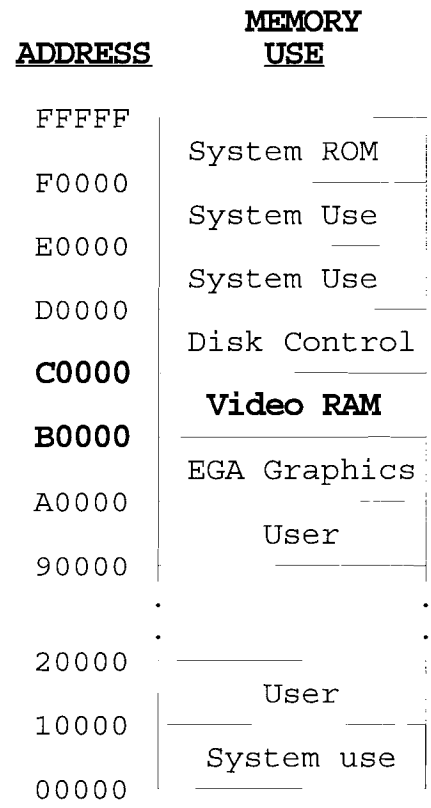


Figure 1. IBM PC/XT/AT Standard Memory Layout

All PC display adapters are memory mapped [STEV89], which means that they have a section of random access memory (RAM) that is devoted to the screen display (see figure 1). This memory has been referred to by various names: video RAM, video buffer, refresh buffer, and display buffer.

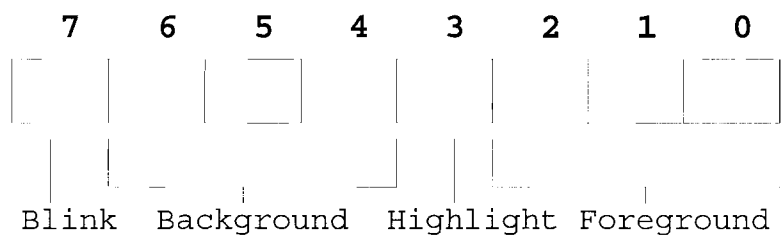
Each character cell on the screen is represented by two adjacent bytes of memory in Video RAM. The first, or low-order, byte specifies the ASCII code of the character displayed. The second byte is the attribute byte, and it

contains the ASCII code that specifies the manner in which the character is displayed. Each character can have one of several attributes and each bit in the attribute byte controls a different feature of the character. The attribute byte differs depending on whether the display is driven by a monochrome or color display adaptor [AITK92].

The attributes [NELS87] that are available on a display driven by a monochrome adaptor are:

Normal - white character on black ground
Reverse video - black character on white ground
Underline - used for white characters only
Blinking Character
High-intensity character

Figure 2 shows the attribute byte for a monochrome adaptor and the purpose of its bits. The attribute controller divides it into two nibbles, whereby the upper nibble (bits four to seven) describes the character background, and the lower nibble (bits zero to three) describes the character foreground.



7	6	5	4	3	2	1	0	Bit
	0	0	0		1	1	1	Normal Char.
	1	1	1		0	0	0	Reverse video
	0	0	0		0	0	1	Underlined Char.
	0	0	0		0	0	0	No Display
	1	1	1		1	1	1	White Char.

Bit 7 when set makes the character blink
Bit 3 when set makes the character high intensity

Figure 2. Attribute Byte for Monochrome Display

Blinking and high intensity are each caused by a specific bit in the attribute byte; bit 7 for blinking and bit 3 for high intensity. The other six bits have a limited range of effective values. Further, not all attribute combinations are available. For instance, underlined reverse video cannot be obtained. Table I shows a list of common attribute values and the effects they create.

TABLE I

COMMON MONOCHROME ATTRIBUTE BYTE VALUES

Attribute Value in Hex	Effect
00	No display, black on black
01	Underlined character
07	Normal character, white on black (this effect is also obtained with attribute values 02 through 06)
08	No display, white on white
09	Underlined, high intensity character
70	Reverse video
71	Underline only
78	Reverse video, high intensity
79	High intensity, underlined
81	Blinking underlined character
87	Blinking normal character
89	Blinking, underlined,high intensity char
F0	Blinking reverse video
F8	Blinking, reverse video, high intensity
FF	White character field, White on White

On the other hand, the attribute byte for color displays further includes the foreground and background colors (see Figure 3). Colors are made from combinations of red, green, and blue. In color text modes, the foreground and background fields control the color of the character (foreground) and the color of the character cell (background). There are eight colors to choose from because this is the number of values that can be specified by a 3 bit field. since the attribute byte's intensity bit (bit 3) applies to the foreground, there are 16 choices in foreground colors - the 8 colors of normal intensity plus 8

high intensity colors [AITK92]. Table II shows these 16 colors. On some monitors, the color brown is dark yellow, and white is light gray.

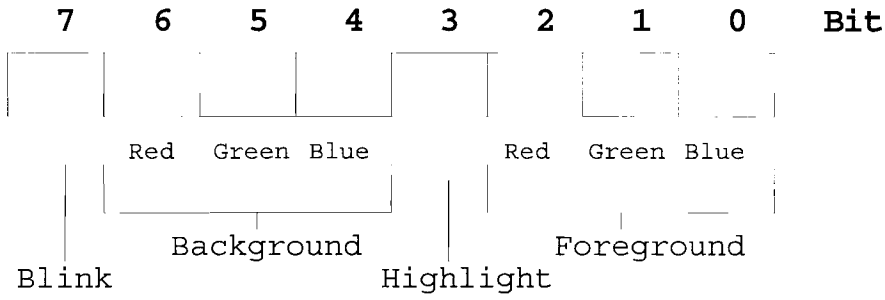


Figure 3. Attribute Byte for Color Displays

TABLE II

FOREGROUND AND BACKGROUND ATTRIBUTE BYTE SETTINGS COLOR FOR COLOR TEXT MODES

Decimal Value	Binary Value	Foreground or Background	Foreground With Intensity Bit Set
0	000	Black	Dark Gray or Black
1	001	Blue	Light blue
2	010	Green	Light green
3	011	Cyan	Light Cyan
4	100	Red	Light red
5	101	Magenta	Light magenta
6	110	Brown	Yellow
7	111	White	Bright White

4.2 Accessing Screen Data

A screen access program can intercept the computer's Basic Input/Output System (BIOS) Video Interrupt, which is interrupt 16 (10H) in the IBM PC family and compatibles [CHAR92]. This permits the screen access program to capture each character that is written to the screen via BIOS writes.

By using only the BIOS services, an application is virtually guaranteed to run on any IBM compatible machine and with any of the various compatible color or monochrome displays. The penalty paid for this flexibility is a lack of speed due to the extra step of generating an interrupt [NELS87]. As a consequence, many commercial programs bypass the BIOS Video services and characters are written directly to the video memory. This results in a faster application program but prevents the screen access program from capturing all the information that is written on the screen via BIOS services.

On the other hand, since the display adapter uses only the video buffer in RAM to form the image on the screen (see Figure 1), the screen access program is guaranteed a standard location from which to capture all the information that is presented to the sighted user. For instance, an uppercase letter "A", white foreground and green background, is represented by the decimal numbers (65,47) in memory; where 65 is the ASCII value of the character byte and 47 is

the value of the attribute byte. The ASCII numbers that represent characters are exactly what a screen reader sends to the synthesizer for it to translate ASCII text to speech. When 73 is sent to the text-to-speech synthesizer, the letter 'I' would be verbalized. Conversely, any character displayed on the screen can be determined at any point by examining its corresponding location in memory.

An 80 x 25 character display will show 2000 characters, and requires 4000 byte buffer to display the entire screen since each character is represented by a character byte and an attribute byte. The first character on the screen (the character in the upper left corner) is also the first character in video RAM, located at offset position 0000H. The next character to the right is located at offset 0002H. All 80 characters of the first screen line follow in this same manner. Since each screen character takes two bytes of memory, each line occupies 160 bytes of RAM. The first character of the second screen line follows the last character of the first line, and so on (see Figure 4). The character byte is found on an even address and the attribute byte on the following odd address.

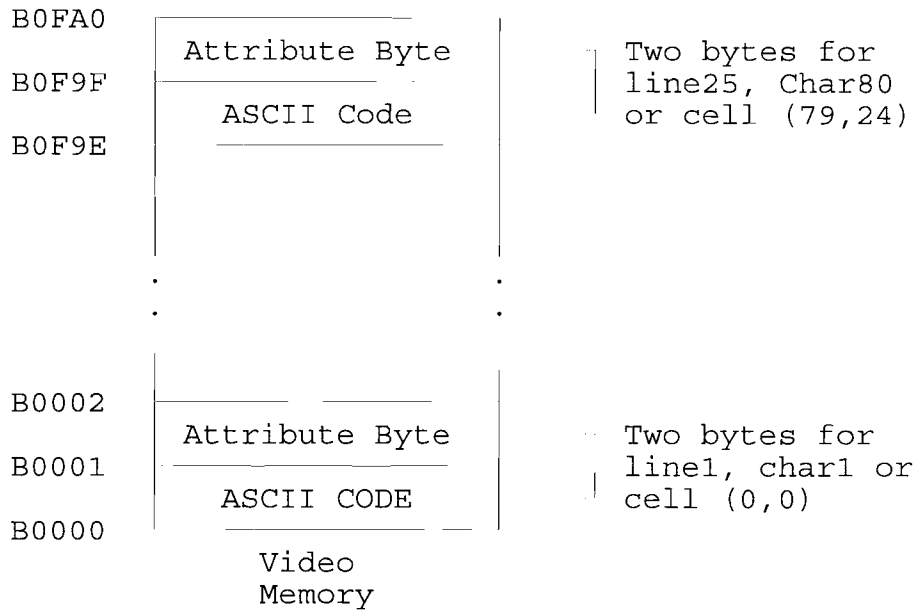


Figure 4. Screen Storage for Text-Based Applications

The starting address of a line within video RAM is found by multiplying the line number by 160 (starting with zero for first line). To get from the beginning of the line to a character within the line, the distance of the character from the start of the line must be added to this value. Since each character takes two bytes, the column number is multiplied by two (also starting at zero for first column). Adding both products together yields the offset position of the character in the video RAM. These calculations are combined to the following formula:

*offset_position(row, column) = (row * 160) + (column * 2)*

The RAM memory of the video card is integrated into the normal RAM of the PC system. The segment address of the video RAM must be known to be used together with the formula above to find the offset position. The address of the beginning of the monochrome video buffer is B0000H, the beginning of the color display video buffer is at B8000H [ANG89]. The actual physical address varies from machine to another.

4.3 Determining Starting Location for Speech

Messages that are spoken are either automatically generated by the program to convey screen changes, or initiated by a user read request. A read request is any user command which requests reading information from the screen such as a key command to read the current word under the cursor. In order to coordinate speech with the screen activity and according to any user specified read request, a temporary *local pointer* is used in the video buffer to indicate the current location for speech. Depending on the task sought, the local pointer takes as its starting position the position of the *system cursor*, the position of the *scan cursor* (also called screen reader *pointer*), or is placed directly at any desired screen coordinates. The system cursor coordinates (Crow, Ccol) are obtained through

BIOS services. The pointer is part of PAL to access protected areas on the screen that cannot be accessed by the system cursor, and acts as a marking device for speech. It moves along as speech commences to mark the character, word, or line just spoken. To allow this process, the following variables are used:

Row - the local pointer row.
Col - the local pointer column.
Crow - the system cursor row.
Ccol - the system cursor column.
Prow - the screen reader pointer row.
Pcol - the screen reader pointer column.

To determine where reading should start, a user can use either the cursor position by entering cursor mode, or the pointer position by entering pointer mode. The local pointer coordinates (row, col) are set to the the cursor coordinates (Crow, Ccol) if the mode is set to cursor, or to the pointer coordinates (Prow, Pcol) if the mode is set to pointer. Then, any read request used is relative to this location. Moreover, if the active mode is pointer, the pointer is set to the position of the local pointer at the end of the command sequence. In cursor mode, the cursor and the pointer do not move when a read request is made. For instance, a command to read the next word results in

speaking the word to the right of the cursor. If this same command is issued many times afterwards, the same word is read again and again. On the other hand, if the pointer mode is the active mode, the pointer moves along to items as they are read, and stops at the last item being spoken. For instance, in pointer mode, a command to read the next word moves the pointer to the beginning of the next word so that when the same command is issued again, the pointer keeps moving successively to the next word to the right.

To announce automatic messages, the local pointer is controlled in the program and positioned based on the change that takes place on the screen. For instance, if the highlight bar changes location, the pointer is placed at the beginning of the newly highlighted item, then an action is taken such as sending the highlighted text to the synthesizer to be verbalized. Section 4.5 explores the method used in locating screen changes to position the local pointer.

4.4 Controlling What Is Spoken

After the local pointer is set to the desired starting location for speech, the stop position for speech can be either the end of a single character, a word, a line, a field, a section of the screen, or the whole screen.

When reading a single character at a time, the local pointer moves character by character in the video buffer.

When reading a word at a time, the local pointer sends all the consecutive non-blank characters to the text-to-speech synthesizer and stops at the first blank character or when either the left or the right edge of the screen is reached. When reading a line at a time, all the characters starting at the local pointer until the end of the current line are sent to the synthesizer. When reading the whole screen, all the text in the video buffer, beginning at the position of the local pointer, is sent to the synthesizer.

In pointer mode, as previously mentioned, read requests are relative to the pointer. The pointer moves as messages are spoken, and always stops at the last item spoken. If the user is reading a character at a time, the pointer stops at the last character read. If the user is reading words, the pointer stops at the beginning of the last word read. If the user is reading lines, the pointer stops at the beginning of the last word spoken because the AICOM Accent text-to-speech synthesizer being used for this implementation uses indexing which allows the tracking of the words spoken.

By using a BitAnd operation between a mask and the current character, it becomes feasible to verbalize a string of characters that have the same background, foreground, or full color attribute as that of the character under the local pointer. To search based on the current character full color attribute, the mask is set to the hexadecimal

value FF where all the bits are equal to the binary value 1. To search based on the current character foreground or background color, the mask is set to 0FH (0000 1111 in binary) or F0H (1111 0000 in binary) respectively. A BitAnd operation between the mask and the attribute of the current character and each character preceding and following it until the result of the BitAnd operation is different leads to locating what needs to be spoken. In other words, the string of interest is comprised of all the contiguous characters whose attribute when bitanded with the mask yields the same result. This is useful when a highlighted text, such as a field, needs to be verbalized.

Finally, the region to be spoken can also be directly specified by using the starting and ending row and column confines on the screen.

4.4.1 Speaking Format

Five spelling tables are provided by the Profile Access Language [IBM92a], each of which has 256 entries numbered from 0 to 255. A list of the tables follows:

Table 1 - is the spelling table with standard sounds for most characters.

Table 2 - is the same as spelling table 1, but all the letters have phonetic names. For instance, the letter "a" is "alpha".

Table 3 - has miscellaneous uses. It contains in positions 0 through 15 sixteen colors. the characters ↑, ↓, ←, →, and ◂ are defined. To identify the color of the current character, this table is used.

Table 4 - Standard ASCII keys (alphabetic, numeric, special characters, Escape, Backspace, Tab, and Enter) are found in this table.

Table 5 - Extended ASCII keys (function keys, cursor movement keys, and keys combined with Ctrl and Alt) are named in this table.

These tables permit the presentation of the screen information to the user in four different formats. The active format determines how information is read. In text format, words are read without speaking punctuation or announcing blank lines. Instead, punctuation is used to create the proper pauses and intonation by the synthesizer. In pronounce format, words are read, punctuation and blank lines are announced. In spell format, words are spelled, punctuation and blank lines are announced. In phonetic format, each letter is represented by a word that begins with the same letter, punctuation and blank lines are announced. For instance, when the spell format is active, the word "act" is spoken as "alpha charley tango".

4.5 Managing Automatic Messages

In order to detect changes on the screen, the access program constantly checks the video memory and the display hardware registers, usually many times a second. To enable this continuous checking, the access program is activated 18.2 times a second by intercepting the timer interrupt (BIOS 6CH). This allows the monitoring of the video memory by comparing the active screen to a saved copy. If a change is found, a set of screen monitoring variables are updated. The drawback to this approach is whenever a message or an item is exactly repeated word by word, the screen reader would not sense this change on the screen.

By continuously checking the display hardware registers, changes to the system cursor position, the disk drive status, and the lockbyte are detected. Similarly, when a change is found, a set of variables are updated. Depending on the change that has taken place, the access program automatically speaks those changes to the user by executing a set of commands associated with the variable that has changed.

Since the timer is updated approximately 18 times per second during which the access program monitors the screen, a period of time (based on the clock ticks) is set for which a changed region or data has to persist before it is automatically spoken. The delay time for automatic messages is therefore set in eighteenths of a second increments.

This is useful in situations where a quick change on the screen is not worth verbalizing to the user. For instance, in some situations the indicator for Insert/Replace is sometimes changed rapidly without any significance to the user; a delay of 9 ticks (half a second) ensures that the indicator has stabilized before announcing it.

4.5.1 Screen Monitoring Variables

To effectively control the amount of information automatically spoken to the user and the point in time it is spoken, the screen is divided into smaller regions. Each section has a variable or a set of variables that are updated each time a change takes place. Anytime characters on the screen change, the PAL variable *ScreenChange* is incremented to signal such change. This variable is used mainly when characters are retrieved from the video buffer for comparison purposes.

4.5.1.1 Action Bar Variables

After providing as input to the access program the row number of the action bar and the color of its selector, it becomes possible to trace the color attribute of the selector whenever it moves on the screen. In Alpha Four, the action bar row number (Arow) is set to 25 and the selector attribute (ActAttr) is set to 111, which is highlighted white on brown.

When the selector moves, including when the field is stationary and the contents of the field changes, a comparison to a previous copy of the screen signals that a changes has taken place, then the PAL variables *actionchange* is incremented and *Acol* is set to the beginning column of the newly highlighted item. As a result, anytime the selector moves on the action bar, Acol marks the first character of the selector. By placing the local pointer at the screen coordinates (Arow, Acol), it becomes feasible to send the selector content to the speech synthesizer to be spoken, or any other action can be taken such as speaking the whole line the selector is on. For example the following pseudo code can be used:

```
If actionchange Then
  position local pointer at screen coordinates (arrow, acol)
   If attribute of current character = ActAttr Then
      Locate string with same BG color as current Character
      Send string found to the text-to-speech synthesizer
   EndIf
EndIf.
```

In Alpha Four, the action bar is not always present on the screen. It is replaced sometimes by a prompt message line requesting user input such as selecting a "Yes" or "No" for an answer. In this situation, the user would only hear

the current word highlighted by the selector and would not hear the message line. To correct this, the video RAM is directly checked for the presence of "yes" or "No" on line 25 prior to announcing the selector. If found, the local pointer is moved the left edge of the screen and the message line is spoken before the selector.

4.5.1.2 Selector Variables

To announce highlighted or selected items as they move anywhere on the screen, and to detect boxes, and pull down or pop-up menus as they appear on the screen, an area of the screen is watched where these changes are likely to take place. The attribute of the selector (SelAttr) is provided as input to the access program as well as the area on the screen to be monitored. The starting row is indicated by the variable *row1* and the last line is set to the variable *row2*. Since the action bar occupies row 25, row1 is set to 1 and row2 is set to 24, and SelAttr is set to 111. When a copy of the screen is being compared with the new screen, the screen reader watches for an attribute change to the specified attribute (SelAttr). When the screen reader recognizes that the attribute has changed to the specified value, the PAL variable *selchange* is incremented. the position of the change is recorded in the variables *Srow* and *Scol*. Thus, to speak the newly highlighted item, the local pointer coordinates (Row, Col) are set to the starting

position for speech at screen coordinates (Srow, Scol).

Simply verbalizing the selector content is not enough since there are situations where other actions need to be taken. The following lists some of these cases:

1. When a Warning, Error, Notice, Help, or Zap message appears on the screen, the whole block of text (more than one line) in addition to the selector needs to be spoken. Furthermore, each window frame utilizes different screen coordinates.
2. At times, a single line of text or a field located prior or after the selector location needs to be spoken.
3. If the user wants to hear the field name, it must be announced prior to speaking the selector. The field name location depends on the current mode. If in the system default Browse mode, field names are on line 1 but the column varies with the selector location. If in the system default View mode, the field name is to the left of the field content and each one occupies one line. Furthermore, if the user wants to hear the record number, it also must be announced prior to the field name and selector. In Browse or View mode, when a pop up menu appears on the screen and the record number and/or field name are turned on, care must be taken not to announce the record number or field name as long as the pop up menu is present on the screen.

To correct these situations, a set of video RAM conditions (such as finding unique characters on the screen associated with each case) and a number of flags are used to coordinate and control the speech. For more information refer to the program listing.

4.5.2 Cursor Variables

If the cursor changes position, the PAL variables *CursorChange* is incremented and *Crow* and *Ccol* are also updated to contain the new location of the cursor. When the system cursor is moving in a horizontal fashion, Crow is constant and Ccol changes. On the other hand, if the cursor is moving in a vertical fashion, Crow changes and Ccol may change depending on whether or not the cursor falls in the same column when it moves to another row. If either Crow or Ccol changes, CursorChange is incremented to also signal the change.

It is important to realize that when the cursor is not active, Crow and Ccol values do not fall within the boundaries of the screen coordinates. On the other hand, when the cursor is active, the value of Crow is between 1 and 25, and the value of Ccol is between 1 and 80.

Using Crow or Ccol signals that a change in the cursor position has taken place. However, when such a change occurs, there is an array of possible actions that can be taken. The user can be in either a Browse mode or a View

mode. In the default Browse mode, the field names are on row 1, and records are listed starting on row 3 with each record occupying one line. In the default View mode, only one record is shown on the screen and each field name is followed by its content on a different line, starting at row 1. If the user is in View mode and enters Change mode to modify an existing field entry, the character under the cursor needs to be spoken plus the field name which is located to the right of the cursor. If the user is in Browse mode and enters Change mode, the character under the cursor needs to be spoken plus the field name which is located on row 1 needs to be spoken (the column varies based on the cursor column location). Furthermore, the cursor appears in pop up menus when requesting user input. As in the case with selector changes, the cursor location is combined with screen data and/or variables tests in order to take any desired action.

4.5.3 ROM BIOS Internal Variables

BIOS reserves the area of memory between addresses 0040:0000 and 0050:0000 for storing internal variables. The content of most of these variables can be read using some BIOS functions, or by using direct access. The following list describes some selected variables, their purposes,and addresses. The address indicated is the offset address of segment address 0040H. For example, a variable with the

offset address 10H has the address 0040:0010 or 10H.

17H This is the keyboard status byte. Function 02H of BIOS keyboard interrupt 16H reads this byte. Accessing this byte allows checking the status of the shift and lock state.

18H This is the extended keyboard status byte, same as 17H except it indicates the active status of SysReq and Break keys (See Figure 5).

Bit Number	Corresponding Key	Value 1
0	Ctrl Key	Pressed
1	Alt Key	Pressed
2	SysReq Key	Pressed
3	Pause Mode	Active
4	Break Key	Pressed
5	Num Lock	Pressed
6	Caps Lock	Pressed
7	Insert	Pressed

Figure 5. Extended keyboard Status Byte

3EH The lowest four bits correspond to the number of installed PC disk drives. These bytes also indicated whether the connected drives must be calibrated. This

is mostly the case after an error occurs during read, write or search access. When an error occurs, the corresponding bit in this byte is set to 0.

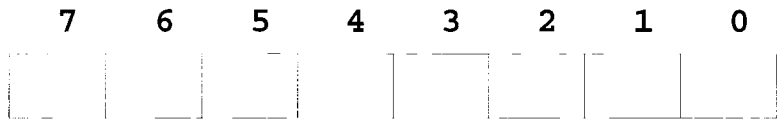
3FH The four lower bits of this byte indicate whether or not the disk drive motor is in motion. A 1 in the corresponding bit means the disk is running.

4AH This word (4AH and 4BH) contains the number of text columns per display line in the current display mode.

6CH the four bytes 6CH to 6FH act as a 32-bit counter for both BIOS and DOS. The counter is incremented by 1 on each of the 18.2 timer interrupts per second. This permits time measurement and time display. The value of this counter can be read and set with BIOS interrupt 1AH. If 24 hours have elapsed, it resets to 0 and counts up from there.

A bitwise operation on these variables can be used to trigger an automatic message. For instance, the ROM BIOS maintains as previously mentioned a set of keyboard flags which reflect the status of the Ctrl, Shift, Alt, and Lock keys. For the lock keys (Scroll Lock, Num Lock, Caps Lock, and Insert), the flags byte indicates whether they are set to on or off. Those flags can be read by calling Int 16H Function 02H [KING88]. Each bit in the flags byte corresponds to one of the Ctrl, Shift, alt, or Lock keys. A bit value of 1 means the key is pressed or turned on. For

the correspondence between bits and keys see figure 6.



Bit Number	Corresponding Key
0	Right Shift
1	Left Shift
2	Ctrl
3	Alt
4	Scroll Lock
5	Num Lock
6	Caps Lock
7	Insert

Figure 6. keyboard Flags Byte

To determine the status of a particular lock key, let *LockByte* be BIOS(17H), then the following BitAnd operation can be performed:

```
If (LockByte BitAnd 80H) Equal 80H then Insert is on.
If (LockByte BitAnd 40H) Equal 40H then Caps Lock is on.
If (LockByte BitAnd 20H) Equal 20H then Num Lock is on.
If (LockByte BitAnd 10H) Equal 10H then Scroll Lock is on.
```

In a similar fashion, to determine the status of the first disk drive motor, the last bit of the BIOS data value

at offset 3FH is tested with a bitwise operation. For instance, if a BitAnd operation between BIOS(3FH) and the 01B (B represents Binary) results in 01B would signal that drive a is running.

CHAPTER V

SUMMARY AND RECOMMENDATIONS

A variety of federal and state legislative actions, not the least of which is the Americans with Disabilities act, combined with public sentiment are resulting in increasing awareness and emphasis on accessibility. In concert with this movement, the software industry has been asked to make its products more accessible to individuals with disabilities. This has raised questions among the members of the industry as to what exactly the problems are, and which steps they can take to help make their products more accessible [VAN92]. This paper focuses on a small subset in an effort to narrow this gap for individuals who are blind or severely visually impaired by customizing access to a text based application called Alpha Four.

5.1 Summary

A user is always in either the review mode or the application mode. When in review mode, the running application is suspended and the control facilities provided are used to further clarify and explore the interface. Whenever the user enters the review mode, the processing of information becomes slower since extra steps are added to perform a task . An attempt was made to automatically

provide enough information about the interface so that the user can spend more time working with Alpha Four and less time using the review mode.

Unlike a sighted user, a blind person has to learn both the adaptation and the software, in this case Alpha Four. Furthermore, a good understanding of many computer and speech related issues are a must for a blind user. For instance, a typical computer user does not have to know about screen color attributes to move from one field to another when using a database. Whereas a blind user must have such knowledge in order to have access.

5.2 Recommendations for Future Work

The user interface for a blind user is unfortunately hard and not as advanced as that of a sighted user. There are several improvements which can be made to enhance the non-visual interface provided in this research.

5.2.1 Clustering

The control facilities provided have to be memorized or available on braille paper so that the user can remember each command and its purpose. An enhancement can be made by clustering commands into a tree of menus in a similar manner to present sighted user interfaces. Similar commands are grouped into a single menu. In this way traversing and selecting available commands would be much simpler and would

not result in overloading the blind user memory.

5.2.2 Non-Speech Audio

The interface could futher be improved through the use of non-speech audio. The drawback to this approach is that a user has to remember and distinguish the meaning of each beep signal. Moreover, an analogy must be made between the screen object and the signal used, which is not an easy task to be done through audible beeps.

5.2.3 Spatial Audio

Navigation in the interface is hard since audio feedback does not provide an easy way for users to keep track of their position on the screen. By integrating real time spatial audio to the interface, users are provided with an easier method to keep track of their position in the interface. With spatial sound, as the user reads the screen content, the voice seems to float from left to right.

REFERENCES

[ACC91] Accent™ User's Manual. AICOM Corporation, Fifth Edition, February 1991.

[AITK92] P. Aitken. MS-DOS 5 Programming. Microsoft Press, Redmond, Washington, 1992.

[ANG89] J. Angermeyer et al. The Waite Group's MS-DOS developer's Guide. Second Edition, Howard W. Sams & Company, 1989.

[BETH94a] W. Edwards, E. Mynatt, T. Rodriguez. A Non-Visual Interface to the X Window System. Mercator Project, Georgia Institute of Technology, 1994.

[BETH94b] E. Mynatt, G. Weber. Contrasting Two Approaches. Mercator Project, Georgia Institute of Technology, 1994.

[BROWN89] C. Brown. A Practical Guide to the Selection and Use of Adapted Computer Technology. Second Edition, 1989.

[CARL92] C. Brown. "Assistive Technology Computers and Persons with Disabilities." Communications of the ACM, Vol.35, No.5, pp. 36-45, May 1992.

[CHAR92] C. Opperman. "Application of Smart Screen Technology In Text-Based Voice Output, Screen Access Programs." In Proceedings of the Seventh Annual Conference on Technology and Persons with Disabilities, Los Angeles, California, pp.391-395, March 1992.

[DANI92] D. Hinton. "Braille Devices And Techniques To Allow Media Access." Braille Monitor, pp.15-21, March 1992.

[DOUG90] D. Griffith. "Computer Access for Persons Who Are Blind or Visually Impaired: Human Factors Issues." Human Factors, Vol.32, No.4, pp.467-475, August 1990.

[EDW88] A. Edwards. "The Design of Auditory Interface for Visually Disabled Users." In Proceedings of ACM Conference on Human Factors in Computing Systems, CHI'88, New York, pp.83-88, May 1988.

[EDW89] A. Edwards. "Soundtrack: An Auditory Interface for Blind Users." Human Computer Interaction, Vol.4, pp.45-66, 1989.

[ELIM90] Y. El-Imam. "Text-To-Speech Conversion on a Personal Computer." IEEE Micro, pp.62-74, August 1990.

[ERIC92] E. Bohlman. "Tinytalk: a Powerful, Low-Cost Screen Reader." In the IEEE Proceedings of the Johns Hopkins National Search for Computing Applications to Assist Persons with Disabilities, Maryland, pp.191-192, February 1992.

[GAV89] W. Gaver. "The SonicFinder: An Interface That Uses Auditory Icons." Human-Computer Interaction, Vol.4, pp.67-94,1989.

[HINT92] A. Johnson. "Modifying The Computer Interface." Proceedings of the Seventh Annual Conference on Technology and Persons With Disabilities, Los Angeles, CA, pp.212-223, March 1992.

[HIRS90] J. Hirschberg. "Voice Response Systems: Technologies And Applications." AT&T Technical Journal, pp.42-51, September/October 1990.

[IBM92a] Screen Reader/DOS 1.2 Reference. IBM Independence Series, Second Edition, June 1992.

[IBM92b] Screen Reader/DOS 1.2 User's Guide. IBM Independence Series, Second Edition, June 1992.

[JAKO93] N. Nielsen. "Noncommand User Interfaces." Communications of the ACM, Vol.36, No.4, pp.83-99, April 1993.

[JIM93] J. Thatcher. "The Problems and Challenges of the Graphical User Interface." Braille Monitor, pp.9-16, November 1993.

[KRIS91] K. Jamsa. DOS: The Complete Reference. Third Edition, Osborne McGraw-Hill, Berkeley, California, 1991.

[LAZZ90] J. Lazzaro. "Opening Doors for the Disabled." Byte, pp.258-268, August 1990.

[LAZZ93] J. Lazzaro. Adaptive Technologies For Learning and Work Environments. American Library Association, 1993.

[LEON91] M. Leonard. "Speech Poised To Join Man-Machine Interface." Electronic Design, pp.43-48, September 26 1991.

[LUDW90] L. Ludwig. "Extending The Notion of a Window System to Audio." Computer, pp.66-72, August 1990.

[NELS87] N. Johnson. Advanced Graphics in C: Programming and Techniques. Osborne McGraw-Hill, Berkeley, California, 1987.

[OMAL90] M. O'Malley. "Text-To-Speech Conversion Technology." Computer, pp.17-23, August 1990.

[PAUL92] P. Jubinski. "Virtac, a Virtual Tactile Computer Display." In the IEEE Proceedings of the Johns Hopkins National Search for Computing Applications to Assist Persons with Disabilities, Maryland, pp.208-211, February 1992.

[ROBB91] J. Robbins. Mastering DOS 5. Third Edition, Sybex Inc, Alameda, California, 1991.

[STEV89] R. Stevens. Graphics Programming in C. M&T Publishing, Redwood, California, 1989.

[TED92] T. Henter. "Macros and Screen Access." In Proceedings of the Seventh Annual Conference on Technology and Persons with Disabilities, Los Angeles, CA, pp.239-243, March 1992.

[VAN89] G. Vanderheiden. "Nonvisual Alternative Display Techniques for Output from Graphics-Based Computers." Journal of Visual Impairement and Blindness, Vol.83, No.8, pp.383-390, October 1989.

[VAN89] G. Vanderheiden. "Thirty Something Million: Should They Be Expectations?" Human Factors,

Vol.32, No.4, pp.383-396, August 1990.

[VAN92] G. Vanderheiden. Making Software More Accessible for People with Disabilities. Release 1.2, Trace Center, University of Wisconsin, 1992.

[WB93] SLIMWARE Window Bridge User's Manual. Syntha-Voice Computers, Ontario, Canada, 1993.

APPENDIXES

APPENDIX A

HARDWARE AND SOFTWARE REQUIREMENTS

The screen reader program runs in the following environment.

1. Machine Requirements
    A. Personal computer with 512k of memory, and a serial connector.
    B. Accent™ text-to-speech synthesizer.
    C. An IBM 18-key keypad (optional).

2. Program Requirements
    A. DOS operating system 3.3 or later.
    B. Alpha Four database system version 2.00.
    C. IBM Profile Access Language (PAL).
    D. Alpha 4 profile program.

APPENDIX B

STARTING SCREEN READER

At the DOS prompt, type *srd12* and press *Enter*; or type *srdload alpha4*. To control the screen reader program, PAL provides a block of 16 keys on the keyboard to be used for user commands. On the other hand, an optional 12-key keypad can be added to the keyboard and can be the source of input for the screen reader. The correspondence between the keyboard and keypad keys is shown in table III [IBM92a].

TABLE III

CORRESPONDENCE BETWEEN THE KEYBOARD AND KEYPAD KEYS

Keyboard	Keypad
1 2 3 4	1 2 3 A
Q W E R	4 5 6 B
A S D F	7 8 9 C
Z X C V	* 0 # D
Tab	Help
Alt	Stop

To activate or de-activate the keyboard keypad functions, *Ctrl* and *Shift* keys must be pressed and released together. Each time the pop-up key is activated, there is a

beep. Each time it is deactivated, there is a different beep. If the user wants to press a single key or key sequence, the Ctrl and Shift keys can be pressed with the key that corresponds to the desired command. When the Ctrl and shift keys are released, the keyboard returns to its normal operation.

While the pop-up keypad is active, most of the keyboard keys are disabled and pressing any key beside those of the pop-up keypad causes an error beep. However, the cursor movement keys, Page up and Page Down keys can be used.

APPENDIX C

USER FUNCTIONS

the following is a list of the key sequence definitions. The keypad equivalent keys are included in parentheses.

key 1 (1) - read previous line.
key 2 (2) - read current line.
key 3 (3) - read next line.
key Q (4) - read previous word.
key W (5) - read current word.
key E (6) - read next word.
key Z (7) - read previous character.
key X (8) - read current character.
key C (9) - read next character.

keys beginning with a 0
key XX1-D (001-9) - read lines 1 through 9.
key X1X-D (010-9) - read lines 10 through 19.
key X2X-5 (020-5) - read lines 20 through 25.
key X4 (0A) - read whole screen.
key XR (0B) - read the rest of screen from current position.
key XA (07) - read character above.

key XS (08) - what is color of current character.

key XD (09) - read character below.

key XF (0C) - read rest of line.

key XZ (0*) - move pointer to top of screen.

key XC (0#) - move pointer to bottom of screen.

key XV (0D) - move pointer to right edge.

key X Tab (0H) - manually enter a field mask number.

key X Alt (0S) - Use pointer position to get a field mask attribute value.

keys beginning with *

key Z1 (*1) - say previous field.

key Z2 (*2) - say current field.

key Z3 (*3) - say next field.

key Z4 (*A) - say rest of field.

key Z Tab (*H) - set field definition to full attribute, background, or foreground.

key ZQ (*4) - spell previous word.

key ZW (*5) - spell current word.

key ZE (*6) - spell next word.

key ZR (*B) - spell current word with caps.

key ZA (*7) - phonetic previous character.

key ZS (*8) - phonetic current character.

key ZD (*9) - phonetic next character.

key ZF (*C) - ASCII value of current character.

key ZZ (**) - text format.

key ZX (*0) - pronounce format.

key ZC (*#) - spell format.

key ZV (*D) - phonetic format.

keys that start with #

key C1 (#1) - search for a string.

key C2 (#2) - continue previous search.

key C3 (#3) - search for previous string from top of screen.

key CQ (#4) - keyboard lock keys status.

key CW (#5) - determine current format for reading information from the screen.

key CE (#6) - current window boundaries.

key CR (#B) - toggle LockStatus.

key CS (#8) - say pointer position.

key CD (#9) - current mode (cursor or pointer) and position status.

key CF (#C) - say cursor position.

key CZ (#*) - route cursor to pointer.

key CX (#0) - switch to pointer mode.

key CC (##) - route pointer to cursor.

key CV (#D) - switch to cursor mode.

key C Tab (#H) - start ignoring a character.

key C Alt (#S) - stop ignoring a character.

keys that start with A

key 41 (A1) - set speech pitch; values to enter are between

1 and 9, with 1 having the lowest pitch and 9 the highest.

key 42 (A2) - set speech rate; value 1 is the slowest and value 9 the fastest.

key 4Q (A4) - toggle caps; when on, capital letters are preceded by the word Cap.

key 4W (A5) - toggle spaces announcement when in spell or phonetic format.

key 4R (AB) - toggle graphics; when on, graphic characters are read.

key 4S (A8) - toggle screen wrap.

key 4D (A9) - toggle numbering.

keys that start with B

key R1 (B1) - read first word of line.

key R2 (B2) - read middle word of line.

key R3 (B3) - read last word of line.

key RQ (B4) - read rest of word above.

key RW (B5) - read rest of current word.

key RE (B6) - read rest of word below.

keys that start with C

Key FX (C0) - say message line (row 23).

Key F1 (C1) - say help line (row 24).

Key F2 (C2) - say function key line or action bar (row 25).

Key F3 (C3) - say current mode, which can be Browse, View,

Edit, Pause, and so on.

Key FQ (C4) - report settings for selector and action bar.

Key FW (C5) - announce active record number.

Key FE (C6) - toggle record number automatic announcement.

Key FA (C7) - toggle action bar automatic announcement.

Key FS (C8) - toggle selector automatic announcement.

Key FD (C9) - toggle column heading automatic announcement.

Key F4 (CA) - Announce header title when cursor is active.

Key FR (CB) - cycle through boxes present on the screen.

Key FF (CC) - say active point, selector or cursor line.

Key FV (CD) - where am i.

Key FZ (C*) - announce menus present on the screen.

Key FC (C#) - location of active point.

APPENDIX D

PROGRAM LISTING

```
/Alpha4.kpd profile
Key listing
FX (C0) - say message line ( row 23)
F1 (C1) - say help line ( row 24)
F2 (C2) - say function key line or action bar ( row 25)
F3 (C3) - say current mode
FQ (C4) - report settings
FW (C5) - announce active record number
FE (C6) - toggle record number
FA (C7) - toggle action bar autospeak
FS (C8) - toggle selector autospeak
FD (C9) - toggle Column Heading
F4 (CA) - profile name
FR (CB) - cycle through boxes
FF (CC) - say active point, selector or cursor line
FV (CD) - where am i, box count and current box
FZ (C*) - announce menus present on the screen
FC (C#) - location of active point/

$include 'core'

/ Initialize the following command sequence to be
  processed automatically when the profile is loaded /
```

```
[] = Initialize {

     Msg ( 'alpha 4 profile'),
     AutoDelay ( 1, 1),          || Set the time to 1/18th
                                 || of a second
     AutoDelay ( 2, 1),          || changed region must
                                 || persist before
     AutoDelay ( 3, 1),          || an autospeak takes
                                 || action.
     AutoDelay ( 4, 27),         || Wait for startup
                                 || screen to settle

     AutoDelay ( 5, 1),
     AutoDelay ( 6, 1),
     AutoDelay ( 7, 1),
     AutoDelay ( 8, 1),
     AutoDelay ( 9, 1),
     AutoDelay ( 10, 1),

     Set ( actattr, 111),        || selector attribute
     Set ( Arow, 25),            || action attribute
     Set ( nostop, True),        || coordinate speech
                                 || between autospeaks
     Set ( Sayhead, True),       || field names in browse
                                 || and view modes
     Set ( sayrec, True),        || announce record
                                 ||  number

     Set ( row1, 1),             || Start of monitor area
     Set ( row2, 24),            || End of monitor area
     Set ( selattr, 111),        || Selector attribute

     Monitor( row1, row2),       || Area of th escreen to
                                 || be watched

     Set ( first1, True),        || Variables to control
                                 || first
     Set ( first2, True),        || triggers of
                                 || autospeaks
     Set ( first3, True),

     Set ( first6, True),

     Set ( first7, True),

     Set ( first9, True),

     Set ( auto1, True),

     Set ( auto2, True),

     Set ( auto3, True),
```

```
Auto ( 4, On),                  || Turn off all
                                || autospeaks but Auto4

Auto ( 2, Off),
Auto ( 3, Off),
Auto ( 1, Off),
Auto ( 5, Off),
Auto ( 6, Off),
Auto ( 7, Off),
Auto ( 8, Off),
Auto ( 9, Off),
Auto ( 10, Off),
Auto ( 30, Off),
Auto ( 31, Off),
Auto ( 32, Off)
}

|| Read line 23 on the screen and announce whether
|| the database order is ascending or descending.

   /$h message line/
   [C0] =
     Save( Wind, Ignore),              || save setttings
     Wind( FullScreen),                || Set reading
                                       || boundries to
                                       ||full screen
     Ignore( ':'),
     Get( 23, 1),
     Right( 78),     || Set right edge reading boundry

     Stop,
     Say( Line),
     Right( 80),
     Get (23, 79),
     If (RC Eq 0) And (Char Eq "A") Then
        Msg( ' ascending order ')
     ElseIf (RC Eq 0) And (Char Eq "D") Then
        Msg( ' descending order ')
     EndIf


   /$h help line /
   [C1] =
     Save( Wind, Pointer),
     Wind( FullScreen),
     Get( 24, 1),
     Stop,
     Say( Line)


   || Say line 25 which is at times the action bar
```

```
|| and the function
|| keys available for use.

/$h function key or action bar/
[C2] =
  Save( Wind, Pointer),
  Wind( FullScreen),
  Get( 25, 1),
  Stop,
  If Attr Eq 118 Then    || to correct
                         || pronunciation when

     While RC EQ 0       || words have no spaces
                         || between them
       Say( Field, Fg),  || but appear in different
                         || colors.
       Get( NextFld, Fg)
     EndWhile
  Else Say( Line)
  EndIf

|| Announce current mode, which can be Browse,
|| View, etc...
/$h current mode
[C3] =
  Save( Wind, Pointer, Wrap),
  Wind( FullScreen),
  Wrap( Off),
  Get( 23, 1),
  Stop,
  Right( 7),
  Say( Line),
  Msg( ' mode') /

/$h report settings/
[C4] =
  Stop,
  Out( 'action bar set at row '),
  Out( Arow),
  Out( ' for attribute '),
  Msg( actattr),

  Out( 'monitoring row '),
  Out( row1),
  Out( ' through '),
  Msg( row2),

  Out( 'monitoring for selector '),
  Msg( selattr),

  If Not( Auto1)  Then
     Msg( 'action bar off')
```

```
  EndIf,

  If Not( Auto3)  Then
     Msg( 'selector off')
  EndIf

/$h announce active record number/
[C5] =
  Stop,
  If ( display( 23, 29, 3) EQ 'Rec' ) Then
                          || Make sure correct screen
     Get( 23, 38),
     Msg( 'reckord '),
     Say( Word)
  Else Msg( 'records are not on active screen')
  EndIf

/$h toggle record number /
[C6] =
  Cycle{ Set( SayRec, False),
              Msg( 'no record numbers');
         Set( SayRec, True),
              Msg( 'record numbers') }

/$h toggle action bar announcement/
[C7] =
      If State( Auto, 1)
      Then Auto( 1, Off),
           ActionBar( Off),
           Out( 'no '),
           Set( auto1, False)
      Else Auto( 1, On),
           ActionBar( On),
           Set( auto1, True)
      EndIf


/$h toggle selector announcement /
[C8] =
      If State( Auto, 3)
      Then Auto( 3, Off),
           Out( 'no '),
           Set( auto3, False),
           If Not( State( Auto, 2))
                   || Possibly leave on box change
              Then Monitor( Off)
           EndIf
      Else Auto( 3, On),
           Monitor( On),
           Set( auto3, True)
      EndIf,
      Msg( 'selector')
```

```
/$h toggle column heading /
[C9] =
  Cycle{ Set( Sayhead, False),
              Msg( 'no headings');
         Set( Sayhead, True),
              Msg( 'headings') }

/$h announce header title when cursor is active/
[CA] =
  Save( Wind),
  Wind( Fullscreen),
  If Crow Lt 25 Then
     Get( 2, Ccol),
     Get( Word),
     Say( Word)
  EndIf


/$h cycle through boxes
[CB] =  /

/$h current active point /
[CC] =
  Set( none, True),

  If Crow LT 26 Then
     Get( Crow, Ccol),
     Set( none, False),
     Say( Line)
  EndIf,

  If (Attr( Arow, Acol) Eq actattr) Then
     Get( Arow, Acol),
     Msg( ' action at '),
     Say( Field, Bg),
     Set( none, False)
  EndIf,

  If (Attr( Srow, Scol) Eq selattr)  Then
     Get( Srow, Scol),
     Msg( ' selector at '),
     Say( Field, Bg),
     Set( none, False)
  EndIf,

  If none Then
     Msg( 'no active point ')
  EndIf


/$h where am i/
```

```
[CD] =
  Save( Mode, Wrap, Trap, Wind),
  Wind( FullScreen),
  Mode( Pointer),
  Trap( Off),
  Wrap( On),
  Get( Top, Left),
  Set( Found1, False),
  Set( Found2, False),
  Set( Found3, False),
  Stop,

  Get( SameAttr, 62, 255, +),
                                || Yellow on Cyan
  If RC EQ 0 Then
     Set( savr1, Row),          || Save row and column
     Set( savc1, Col),          || of local pointer
     Set( Found1, True)
  EndIf,
  If Found1 Then
     Get( Row+1, Left)
  Else
     Get( Top, Left)
  EndIf,

  Get( SameAttr, 63, 255, +),       || White on Cyan
  If RC EQ 0 Then
     Set( savr2, Row),
     Set( savc2, Col),
     Set( Found2, True)
  EndIf,
  If Found2 Then
     Get( Row+1, Left),
     Get( SameAttr, 63, 255, +),
     If RC EQ 0 Then
        Set( savr3, Row),
        Set( savc3, Col),
        Set( Found3, True)
     EndIf
  EndIf,

  If found3 then              || Start with top menu
     Get( savr3, savc3),
     Msg( ' menu '),
     Say( Field, 63)
  ElseIf found2 then
     Get( savr2, savc2),
     Msg( ' menu '),
     Say( Field, 63)
  ElseIf found1 then
     Get( savr1, savc1),
     Msg( ' menu '),
```

```
   Say( Field, 62)
EndIf,

Wrap( Off),

If (Attr( Srow, Scol) Eq selattr) Then
   Msg( ' selector at '),
   Get( Srow, Scol),
   Say( Field)
EndIf,

If ( Crow Lt 26) And ( Crow Gt 0) Then
    Msg( ' cursor at '),
    Get( Crow, Ccol),
    Say( line)
EndIf,

If (Attr( Arow, Acol) Eq actattr)  Then
   Out( ' action item at '),
   Get( Arow, Acol),
   Say( Field)
EndIf


/$h announce menus present on the screen/
[C*] =
       Save( Mode, Wrap, Trap, Wind),
       Wind( FullScreen),
       Mode( Pointer),
       Trap( Off),
       Wrap( On),
       Get( Top, Left),
       Set( Found1, False),
       Set( Found2, False),
       Set( Found3, False),
       Stop,

       Get( SameAttr, 62, 255, +),
                                  || Yellow on Cyan
       If RC EQ 0 Then
         Msg(' menu 1 '),
         Say( Field, 62),
         Set( Found1, True)
       EndIf,
       If Found1 Then
          Get( Row+1, Left)
       Else
          Get( Top, Left)
       EndIf,

       Get( SameAttr, 63, 255, +),
                                  || White on Cyan
```

```
        If RC EQ 0 Then
          If Found1 Then
             Msg( ' menu 2 ')
          Else Msg( ' menu 1 ')
          EndIf,
          Say( Field, 63),
          Set( Found2, True)
        EndIf,
        If Found2 Then
           Get( Row+1, Left)
        ElseIf Not( Found2) And Not( Found1) Then
           Msg( ' no active menu '),
           exit
        EndIf,

        Get( SameAttr, 63, 255, +),
        If RC EQ 0 Then
          If Found1 And Found2 Then
             Msg( ' menu 3 ')
          Else Msg( ' menu 2 ')
          EndIf,
          Say( Field, 63),
          Set( Found3, True)
        EndIf,
        If Found3 Then
           Get( Row+1, Left)
        Else exit
        EndIf,

        Get( SameAttr, 63, 255, +),
        If RC EQ 0 Then
          If Found1 And Found2 Then
             Msg( ' menu 4 ')
          Else Msg( ' menu 3 ')
          EndIf,
          Say( Field, 63)
        EndIf

/$h announce current active menu/
[C3] =
  Set( Found1, False),
  Save( Trap, Wind, Mode, Wrap),
  Trap( Off),
  Wrap( On),
  Wind( Fullscreen),
  Mode( Pointer),
  Stop,

  If Crow GT 25 Then
     Get( Srow, Scol),
     Get( SameAttr, 63, 255, -),
                                    || White on Cyan
```

```
   If RC EQ 0 Then
      Set( nostop, true),
      Get( row, left),
      Get( SameAttr, 63, 255, +),
      Msg(' menu '),
      Say( Field, 63),
      Set( Found1, True)
   EndIf,

   If Not( Found1) Then
      Get( Srow, Scol),
      Get( SameAttr, 62, 255, -),
                                  || Yellow on Cyan
      If RC EQ 0 Then
         Get( row, left),
         Msg(' menu '),
         Say( Field, 62)
      EndIf
   EndIf

ElseIf ( Crow GT 1) Or ( Crow LE 25) Then
   Get( Crow, Ccol),
   Get( SameAttr, 63, 255, -),
                                  || White on Cyan

   If RC EQ 0 Then
      Set( nostop, true),
      Get( row, left),
      Get( SameAttr, 63, 255, +),
      Msg(' menu '),
      Say( Field, 63),
      Set( Found1, True)
   EndIf,

   If Not( Found1) Then
      Get( Crow, Ccol),
      Get( SameAttr, 62, 255, -),
                                  || Yellow on Cyan
      If RC EQ 0 Then
         Get( row, left),
         Msg(' menu '),
         Say( Field, 62)
      EndIf
   EndIf
EndIf,

If Not( Found1) Then
   Msg( ' no active menu on the screen ')
EndIf
```

```
/$h location current active point /
[C#] =
  Set( none, True),
  If (Attr( Arow, Acol) Eq actattr)  Then
     Out( 'action item at '),
     Msg( Arow), Msg( Acol),
     Set( none, False)
  EndIf,

  If (Attr( Srow, Scol) Eq selattr) Then
     Msg( 'selector at '),
     Msg( Srow),
     Msg( Scol),
     Set( none, False)
  EndIf,

  If ( Crow Lt 26) And ( Crow Gt 0) Then
      Msg( 'cursor at'),
      Msg( Crow),
      Msg( Ccol),
      Set( none, False)
  EndIf,

  If none Then
     Msg( 'no active point')
  EndIf



/ action bar change /
[] = Autospeak { 1: <ActionChange>
True =
  If first1                  || Ignore first trigger
   Then Set( first1, False),
   Exit
  EndIf,

  Save( Trap),
  Trap( Off),
  Set( ReadIt, False),
  Get( Arow, Acol),

  If Attr Eq actattr
  Then      || Check cases when action bar becomes
            || a message line
       Set( @stat, Display( Arow, Acol, 2)),
       If ( @Stat EQ 'Ye') OR ( @Stat EQ 'No')Then
         Set ( ReadIt, True)
       EndIf,

       If Not( ReadIt) AND NOT( nostop) Then
         Stop
```

```
      ElseIf Readit Then
                  || Action Bar is a message line
         Get( Arow, 1),
         Save( Ignore),
         Wind( FullScreen),
         AddIgnore( ':'),
         Say( Field, FG)
      EndIf,

      Get( Arow, Acol),
                     || read action bar selector
      Get( Field, Bg),
            || by using the background attribute
      Say( Field, Bg)

  EndIf

}


/ selector change /
[] = Autospeak { 3: <SelChange>
         || only say if just selector movement
True =
  If first3                || Ignore First trigger
     Then Set( first3, False),
     Exit
  EndIf,

  Set( @stat, Display( 7, 36, 4)),

  If ( @stat EQ 'Warn') Or ( @stat EQ ' Not') Then
                  || Warning or Notice
     Stop,
     Save( Wind, Trap, Wrap),
     Wind( 7, 11, 13, 69),
     Trap( Off),
     Wrap( On),
     Get( 7, 15),
     Say( Line),
     Msg( ' press enter or escape to continue '),
     If Display( 8, 12, 3) EQ 'Bad' Then
                           || To control critical
        Set( Flag1, False)    || error flag
     EndIf,
     exit
    EndIf,

  If ( @stat EQ 'Erro')  Or    || Warning or Error
     ( @stat EQ ' Err')  Then
     If Not( Skip) Then
        Stop,
```

```
        Set( nostop, True),
        Save( Wind, Trap, Wrap),
        Wind( 7, 11, 13, 69),
        Trap( Off),
        Wrap( On),
        Get( 7, 15),
        Say( Line),
        Set( Skip, True),
        If Display( 8, 12, 3) EQ 'Bad' Then
                                || To control critical
          Set( Flag1, False)||error flag
       EndIf
     EndIf
 EndIf,

 If ( @stat EQ 'Erro')  Or     || Warning or Error
    ( @stat EQ ' Err')  Then
    Set( Skip, False)
 EndIf,

Set( @stat, Display( 7, 12, 4)),
If ( @stat EQ 'Crit') Then      || Critical error
    If Not( Flag1) Then         || To allow speaking of
                                || Retry or Fail
        Stop,
        Save( Wind, Trap, Wrap),
        Wind( 7, 11, 15, 69),
        Trap( Off),
        Wrap( On),
        Get( 7, 11),
        Say( Line),
        Set( Flag1, True),
        exit
    EndIf
EndIf,

If ( Display( 2, 38, 4) EQ 'Help')  Then
   Save( Wind, Trap, Ignore),
   Wind( 3, 9, 17, 70),
   Trap( Off),
   Ignore('-'),
   Set( nostop, True),
   Set( InHelp, True),
   Get( 3, 9),
   Say( line)
Else Set( InHelp, False)
EndIf,

If ( Display( 4, 36, 3) Eq 'Zap') Then
                                    || Zap menu
   If Not( Inzap) Then
       Save( Wind, Trap, Wrap),
```

```
        Wind( 5, 1, 9, 75),
        Trap( Off),
        Get( Top, Left),
        Wrap( On),
        Set( Inzap, False),
        Say( Line)
    EndIf
EndIf,


If ( Srow Eq 24) Then
    Set( @stat, Display( Srow, Scol, 2)),
    If ( @Stat EQ 'Ye') OR ( @Stat EQ 'No') Then

        Save( Wind, Trap, Ignore),
        Wind( FullScreen),
        Trap( Off),
        Set( nostop, True),
        Get( Srow, 1),
        Wind( FullScreen),
        AddIgnore( ':'),
        If ( Display( 4, 36, 3) Eq 'Zap') Then
           Set( Inzap, True)
        EndIf,
        Say( Field, FG)
    EndIf
 EndIf,

 Save( Trap, Wind),
 Trap( Off),
 Wind( FullScreen),

 Set( @STAT1, Display( 23, 1, 6) ),

 If Sayhead And ( @STAT1 EQ 'BROWSE') And
                          || Say Header in Browse
    ( Display( 25, 6, 4) EQ 'Chan') And
                          || Action Bar is active
    Not( InHelp) Then

    Get(Srow, Scol-1),
                 || Check if Scol is at
                 || lefmost position of screen

    If RC EQ 0 Then

      If Char( Srow, Scol-1) NE "▸"  Then
                 || Avoid Pop Up menus When
         Save( Wind, Mode, Trap),
                          || in Browse Mode
         Mode( Pointer),
         Wind( FullScreen),
```

```
        Trap( Off),

        If Not (SayRec) Then
          Stop
        EndIf,

        Set( nostop, True),
                    || read field header in Browse
        Get( 1, Scol),
        Say( Word)
      EndIf

   Else   || Selector Column is not in Column 1
     If (Char( 2,1) EQ "◂") And Not( InHelp)
     Then  || Left of screen

        If Not (SayRec) Then
          Stop
        EndIf,

        Set( nostop, True),
        Get(1,1),
        Say(Word)
      EndIf
   EndIf

EndIf,

Save( Wind, Trap),
Wind( FullScreen),
Trap( Off),
Get( Srow, Scol),
If ( Scol Eq 1) And                || Save As menu
   ( Display( 1, 34, 5) Eq 'S a v') Then
   Set( Nostop, True),
   Get( 6, 1),
   Say( Line)
EndIf,


Save( Trap, Ignore),
Trap( Off),
Get( Srow, Scol),
If RC Eq 0 Then
  Save( Graphics),
  Graphics( On),

|| For cases when selector announcement is not
|| enough,rather the whole line in the active
|| menu or screen needs to be spoken. i.e
|| Index selections
```

```
      If ( Display( Srow, Scol+1, 2) EQ '..') AND
         ( Char( Srow, Scol-1) EQ "▸") AND
      || Avoid saying Memo fields twice
         ( Attr( Srow, Scol+3) NE Selattr) Then
      ||Avoid saying a field with ..
           Save( Trap, Ignore),
           Trap( Off),
           Ignore('.'),
           Set( SayNext, True)
      Else Set( SayNext, False)
      EndIf,


      If nostop Then     || Control Stopping speech
         Set( nostop, False)
      Else Stop
      EndIf,

      If ( Char( Srow, Scol+1) EQ ".") And
      || Correct speech since
      || periods cause spell mode
         ( Char( Srow, Scol+2) EQ ".")  Then
         Save ( Ignore),
         AddIgnore( '.')
      EndIf,

      Get( Field, Bg),
                        || Verbalize selector content
      Say( Field, Bg)
   EndIf,

   If SayNext Then
|| cases that require verbalizing next field
     Get( Srow, Scol+3),
|| caused when selector is highlighting only
     Say( Field, FG),          || first word.
     exit
   EndIf,

   If ( Display( 1, 26, 7) EQ 'F i e l') Then
            || For Field Selection menu
        Save( Mode, Wrap, Trap, Wind),
        Wind( FullScreen),
        Mode( Pointer),
        Trap( Off),
        Wrap( On),
        Get( 2, Left),
        Get( SameAttr, 63, 255, +),
                                  || White on Cyan
        If RC EQ 0 Then
          Wrap( Off),
          exit
```

```
      EndIf,
      Get( NextFld),
      If Rc Eq 0 Then
         msg( ' test 2 '),
         Say( line)
      EndIf,
      exit
  EndIf,

  || Some Menus require user input, and selector
  || is only on parts of
  || the fields.  This forces rest of line to be
  || verbalized.
  Get( NextFld, Fg),

  If ( Char( Srow, Col) EQ ":") Or
              || verbalize next field(s)
     ( Char( Srow, Col+1) EQ ":")  Then
     Save( Wind, Trap, Wrap ,Ignore),
     Wind( FullScreen),
     Trap( Off),
     Wrap( Off),
     /Set( Nostop, True),/
     Ignore( '_'),
     Say( line)
  EndIf,

  If ( Crow Eq 24) Or ( Crow Eq 25) Then
     Set( First6, True),
     Save( Wind, Trap, Wrap ,Ignore),
     Wind( FullScreen),
     Trap( Off),
     Wrap( Off),
     Ignore( '_'),
     Get( Crow, left),
     Say( line)
  EndIf

}


  || start up autospeak, it monitors the first
  || screen of Alpha 4 and
  || turns on other autospeaks
 [] = Autospeak { 4: (23,1)
 'XXXX'= Continue    || This a nonsense value
 Else

  Save( Wind, Mode, Trap),
  Wind( FullScreen),
  Trap( Off),
```

```
  Set( @status, Display( 24, 1, 8)),

  If ( @status Eq 'Default ') Then
                              || Startup Screen message
      Save( Wind, Ignore),
      Wind( FullScreen),
      Ignore              || Backslash delays speech
      Msg( ' default data base set'),
      Get( 24, 23),
      Say( Line),
      Msg(' f 10 continue f 9 change data)'
      Msg( 'base set '),
      Delay( 300), || Stop processing 3 seconds
                   || To allow user totake action

      Selector( selattr),    || Activate Monitor
      Monitor( On),

      ActionBar( Arow, actattr), || Activate
                                 || Action Bar
      ActionBar( On),

      Auto( 3, On),
      Auto( 1, On),
      Auto( 2, On),
      Auto( 8, OFF),
      Auto( 9, On),
      Auto( 6, On),
      Auto( 7, On),
      Auto( 5, Off),
      Auto ( 30, On),
      Auto ( 31, On),
      Auto ( 32, On)

  EndIf

 }


  [] = Autospeak { 5: (25,33)
'Alt-MM' =              || Memo field message
   Save( Graphics),
   Graphics(On),
   If ( Char( 2, 3) NE " ")  Then       || Trial
      Save( Wind, Mode, Trap),
      Wind( FullScreen),
      Set( nostop, True),
      Trap( Off),
      Get( Crow, left),
      Get( NextWord),          || Skip field numbers
      Say( Word)
```

```
 EndIf
}


/ speak cursor field whenever cursor row changes
  in Change mode /
[] = Autospeak { 6: <Crow>
True =
      If first6       || Ignore first trigger
         Then Set( first6, False),
         Exit
      EndIf,

      Save( Trap),
      Trap( Off),
      If ( Attr( Crow, Ccol) EQ 25)
      || Attribute of field in Change Mode
      AND ( Display( 23, 1, 6) EQ 'CHANGE')  Then

           If Sayhead Then
                                  || Say field name
              Save( Graphics),
              Graphics( On),
              If ( Char( 2, 3) EQ " ")  Then
                                 || Browse Mode,
                 Save( Wind, Mode, Trap),
                          || "-" is ASCII (196)

                 Wind( FullScreen),
                 Trap( Off),
                 Get( 1, Ccol),
                 If RC EQ 0 Then
                    Set( nostop, True),
                    Say( Word)
                 EndIf

              Else                    || View Mode
                 Save( Wind, Mode, Trap),
                 Wind( FullScreen),

                 Set( nostop, True),
                 Trap( Off),
                 Get( Crow, left),
                 Get( NextWord),|| Skip field
                                || numbers
                 Say( Word)
              EndIf
           EndIf,

           Save ( Mode, Trap, Ignore),
           Mode ( Cursor),
```

```
      Trap( Off),
      Ignore( '_'),
      Get ( Crow, Ccol),

      If RC EQ 0 Then
         If nostop Then
           Set( nostop, False)
         Else Stop
         EndIf,

         Say ( Char),
      || Say cursor field by using foreground
         Set ( first7, True)
      EndIf

EndIf,

If Display( 25, 37, 5) EQ 'MMemo' then
   || In View mode, record
   If nostop Then
   || Number is constant
      Set( nostop, False)
   || and Cursor Row changes
   Else Stop
   EndIf,

   Msg( 'Memo Field'),
   Set( First7, True)
EndIf,

If Crow Eq 24 Then
   /Get( Crow, Ccol-2),
   If ( Char Eq ":") Then/
    Get( ':', -),
    If Rc Eq 0 Then
      Set( nostop, true),
      Get( Crow, Left),
      Say( Line)
   EndIf
EndIf,

Get(Crow, Ccol),
Wind( Fullscreen),
Get( 'Enter', -),
If Rc Eq 0 then
/ If ( Display( 11, 23, 5) EQ 'Enter') Then/
   Save( Wind),
   Wind( FullScreen),
  / Get( 24, 1),/
  Get( Crow, Left),
   Say( Line)
EndIf,
```

```
      If ( Display( 6, 27, 4) Eq 'Rang') Then
         Msg( ' range description ')
      EndIf
}


/ speak cursor field whenever cursor column
  changes in Change mode /
[] = Autospeak { 7: <Ccol>
True =
      If first7          || Ignore first trigger
         Then Set( first7, False),  || Stop auto7
         || from repeating what auto6 verbalizes
         Exit
      EndIf,

      || In Browse mode, speak cursor field when
      || column changes
      Save( Trap),
      Trap( Off),
      If ( attr( Crow, Ccol) EQ 25)
       AND ( Display( 23, 1, 6) EQ 'CHANGE') Then


           If Sayhead  And     || Say field name
              ( Char( 2, 3) EQ "_") And
              || for Browse mode only
              ( (Attr( Crow, Ccol-1) NE 25) OR
              (Crow EQ 1) ) Then

              Save( Wind, Graphics),
              Wind( FullScreen),
              Graphics( On),
              Get( 1, Ccol),
              If RC EQ 0 Then
                 Set( nostop, True),
                 Say( Word)
              EndIf
           EndIf,

           Save( Mode),
           Mode( Cursor),
           Get ( Crow, Ccol),
           If RC EQ 0 Then
              Save( Ignore),
              Ignore( '_'),

              If nostop Then
                Set( nostop, False)
              Else Stop
              EndIf,
```

```
                Say ( Char),|| Use field foreground
                Set ( first6, True)   || Stop auto6
                                ||from repeating what
          EndIf                 || Auto7 verbalizes
       EndIf,

       If Display( 25, 37, 5) EQ 'MMemo' then
          Msg( 'Memo Field'),
          Set( first6, True)
       EndIf

}



/ speak record number /
[] = Autospeak { 2: (23,38)
'XXXXX' = Continue
 Else

  Set( @stat, Display( 23, 1, 6) ),
  Set( @stat2, Display( 7, 36, 4) ),
|| Take care of warnings

  If ((( Crow LT 23 ) OR ( @stat EQ 'BROWSE') OR
    (@stat EQ 'VIEW  ')  OR
     (@stat EQ 'CHANGE')) AND ( Display( 23,29,6)
         EQ 'Record')) Then

     If sayrec Then
       Save( Trap, Wind),
       Trap( Off),
       Wind( FullScreen),
       Stop,
       Set( nostop, True),
       Get( 23, 38),
       Msg( 'reckord '),
       Say( Word)
     EndIf

  EndIf,

|| Say Memo fields in Change mode when row changes
|| since cursor is
|| not active when in memo fields
  If (Display( 25, 37, 5) EQ 'MMemo') then
     If nostop Then
        Set( nostop, False)
     Else Stop
     EndIf,
```

```
    Msg( 'Memo Field'),
    Set( first6, True)
 EndIf

}


[] = AutoSpeak { 9: <ScreenChange>
True =
   If first9 then   || Ignore first trigger
      Set( first9, False),
      Exit
   EndIf,


  / If ( Attr( Crow, Ccol) EQ 25) AND
       ( Char( 2, 1) EQ "◂") And (Ccol EQ 1) Then
             Save( Ignore),
             Ignore( ' '),
             If SayHead Then
                Get( 1, 1),
                Say( Word)
             EndIf,
             Get ( Crow, 1),
             Say ( Char),
             Set ( first6, True),
             Set ( first7, True)
   EndIf, /


 || Detect Some menus that pop up on the screen
    || with the following if statements

 || Detect the find menu
 If ( Display( 10, 38, 4) EQ 'Find') And
    ( (Ccol EQ 44) OR (Ccol EQ 5) ) Then

     Stop,
     Save( Wind, Ignore),
     Wind( FullScreen),
     Ignore( ' '),
     AddIgnore( ':'),
     If Crow EQ 12 Then
 || Find has 2 menus based on index being used
        Get( 12, 22)
     Else Get( 12, 5)
     EndIf,
     Say( Field, BG)      || Say all fields with
                          || same background color
```

```
EndIf,

|| Detect the Index menu
If ( Display( 6, 35, 3) EQ 'Ind') And
                                    || Ind for Index
   ( Ccol EQ 30) Then
   Stop,
   Save( Ignore),
   Ignore( ' '),
   AddIgnore( ':'),
   Get( 8, 17),
   Say( Field, BG)  || Say all fields with same
                       || background color
EndIf,


|| Detect the locate menu
If ( Display( 10, 37, 6) EQ 'Locate') And
   ( Ccol EQ 18) Then
   Stop,
   Save( Ignore),
   Ignore( ' '),
   AddIgnore( ':'),
   Get( 12, 5),
   Say( Field, BG)        || Say all fields with
                       || same background color
EndIf,

If ( Display( 23, 1, 4) Eq 'Paus') Then
   Stop,
   say(' pausing')
EndIf,

If ( Display( 23, 1, 3) Eq 'End') Then
   Stop,
   say(' End')
EndIf,

Get( 23, 1),
Get( 'Col ', +),
If RC Eq 0 Then
   /Stop,/
   Right( 75),
   Get( 23, 1),
   Say( Line),
   Right( 80)
EndIf,

|| Detect the Search menu
If ( Display( 10, 31, 6) EQ 'Search') Then
   Stop,
```

```
   If ( Crow EQ 12) And (Ccol EQ 20) Then
      Msg( ' search for ')
   EndIf,

   If ( Crow EQ 13) And (Ccol EQ 20) Then
      Msg( ' replace with ')
   EndIf,

   If ( Srow EQ 12) Then
      Msg( ' verify each replacement ')
   EndIf

EndIf,


If ( Display( 1, 35, 6) Eq 'S t a ') Or
|| Status menu
   ( Display( 1, 35, 6) Eq ' a s e')   Then
|| datadase info
   Save( Wrap, Trap, Wind),
   Wrap( On),
   Trap( Off),
   Wind( FullScreen),
   Bottom( 22),
   Get( 2, 1),
   Say( line),
   exit
EndIf,

If ( Display( 4, 37, 4) Eq 'Pack')  Then
|| Pack notice menu
   Save( Wrap, Trap, Wind),
   Wrap( On),
   Trap( Off),
   Wind( 5, 5, 20, 70),
   Get( Top, Left),
   Say( Line)
EndIf,



If ( Display( 8, 31, 4) Eq 'Fiel') And
|| Field Statistics
   ( Crow Eq 12) And ( Ccol Eq 10) Then
   Save( wind),
   Wind( FullScreen),
   Get( 24, 1),
   Say( Line)
EndIf,

If ( Display( 15, 35, 4) Eq 'Info') And
|| Statistics information
```

```
      ( Crow Eq 6) And ( Ccol Eq 15) Then
      Save( wind),
      Wind( FullScreen),
      Get( 24, 1),
      Say( Line),
      Msg( ' information '),
      Get( 16, 10),
      Say( Line),
      Get( 17, 10),
      Say( Line)
   EndIf,

   If ( Display( 2, 1, 8) Eq 'C:\ALPHA') Then
      Auto ( 4, Off),
      Auto ( 2, Off),
      Auto ( 3, Off),
      Auto ( 1, Off),
      Auto ( 5, Off),
      Auto ( 6, Off),
      Auto ( 7, Off),
      Auto ( 8, Off),
      Auto ( 9, Off),
      Auto ( 10, Off),
      Auto ( 30, Off),
      Auto ( 31, Off),
      Auto ( 32, Off),
      Msg( 'Exiting Alpha 4')
   EndIf

  }


  [] = AutoSpeak  { 10: <BIOS( &H3F) BitAnd &B01>
    True = If (BIOS( &H3F) BitAnd &B01) Eq &B01
           Then
               Msg( ' running drive a: ')
           Else Msg( ' ready drive a: ')
           EndIf
  }


/ Modified Core profile/


[] = Initialize {
     Set( StrLen, 0), / current search string
      length /
     Set( FldMask, &hFF),       / field mask
                    initialized to full field /
     Set( @cormsg1, 'enter a number '),
     Set( @cormsg2, 'between 1 and 9') }
```

```
/ To fix situations when local pointer is
     undefined /
$MACRO FixPointer()
 If ( Row LT 1) Or ( Row GT 69) OR
    ( Col Lt 1) OR ( Col GT 80) Then
   Get( 1, 1)
 EndIf
$ENDM


/------------------------/
/ keys 1 - 9             /
/------------------------/

/$h previous line/
[1] =
 FixPointer(),
 Get( Row-1, Left),
 Say( Line)

/$h current line/
[2] =
 FixPointer(),
 Get( Row, Left),
 Say( Line)

/$h next line/
[3] =
 FixPointer(),
 Get( Row+1, Left),
 Say( Line)

/$h previous word/
[4] =
 FixPointer(),
 Get( Prevword),
 Say( Word)

/$h current word/
[5] =
 FixPointer(),
 Get( Word),
 Say( Word)

/$h next word/
[6] =
 FixPointer(),
 Get( Nextword),
 Say( Word)
```

```
/$h previous character/
[7] =
 FixPointer(),
 Get( Row, Col-1),
 Say( Char)

/$h current character/
[8] =
 FixPointer(),
 Say( Char)

/$h next character/
[9] =
 FixPointer(),
 Get( Row, Col+1),
 Say( Char)

/------------------------/
/ keys that start with 0 /
/------------------------/

/$h one more key gives lines 1 through 9/
[00] =
  Mode( Pointer),
  Get( ?, Left),
  Save( Numbering),
  Numbering( On),
  Say( Line)

/$h one more key gives lines 10 through 19/
[01] =
  Mode( Pointer),
  Get( 10+?, Left),
  Save( Numbering),
  Numbering( On),
  Say( Line)

/$h one more key gives lines 20 through 25/
[02] =
  Mode( Pointer),
  Save( Numbering),
  Numbering( On),
  Get( 20+?, Left),
  Say( Line)


/$h whole screen/
[0A] =
  Save( Wrap, Table, Numbering),
  If State( Table) Gt 2 Then
     Table( 1)
  Endif,
```

```
  Get( Top, Left),
  Wrap( on),
  Numbering( Off),
  Say( Line)


/$h rest of screen/
[0B] =
  Save( Wrap, Table, Numbering),
  If State( Table) Gt 2 Then
     Table( 1)
  Endif,
  Wrap( on),
  Numbering( Off),
  Say( Line)

/$h character above/
[07] =
   FixPointer(),
   Get( Row-1, Col),
   Say( Char)

/$h color of current character/
[08] =
  Save( Table),
  Table( 3),
  Out( ' attribute '),
  Out( Attr),                          || say numeric
value for attribute
  Say( Attr BitAnd &H0F),    || say 4 bit
                          || foreground color
  Out(' on'),
  Say( ( Attr BitAnd &HF0) Div 16),
          || say 4 bit background color
  If ( Attr BitAnd &H80) BitEq &H80 Then
     || if blinking bit is ON then
    Out(' or blinking '),
    Say( Attr BitAnd &H0F),
        ||  say 4 bit foreground color
    Out(' on '),
    Say( ( Attr BitAnd &H70) Div 16)
      || say 3 bit background color
  Endif

/$h character below/
[09] =
  FixPointer(),
  Get( Row+1, Col),
  Say( Char)

/$h rest of line/
[0C] =
```

```
  FixPointer(),
  Say( Line)

/$h pointer to top left/
[0*] =
  Mode( Pointer),
  Get( Top, Left),
  Msg( ' top left ')

/$h pointer to bottom left/
[0#] =
  Mode( Pointer),
  Get( Bottom, Left),
  Msg( ' bottom left ')

/$h pointer to right edge/
[0D] =
  Mode( Pointer),
  Get( Row, Right),
  Msg( ' right edge ')

/$h manually enter a field mask /
[0H] =
  Set( temp, 0),
  Set( Key, 18),
  Msg( ' enter a field mask ending with pound'),
  Msg( ' key or press stop to cancel'),

  While ( Key Ne 16) And ( Key Ne 15)
    Keypad( Key),
    Stop,
    If ( Key Ne 16) And ( Key Ne 15) Then
       Set( temp, ( temp * 10) + Key)
    EndIf
  EndWhile,

  If Key Ne 16 Then
    Set( FldMask, temp),
    Msg( ' field mask is '),
    Say( FldMask)
  Else Msg (' field mask setup cancelled')
  EndIf

/$h use pointer position to set up field mask
     attribute value/
[0S] =
  Get( Prow, Pcol),
  If RC EQ 0 Then
     Msg( ' pointer at row '),
     Say( Prow),
     Msg( ' column '),
     Say( Pcol),
```

```
      Set( FldMask, Attr),
      Msg( ' field mask is '),
      Say( Attr)
   Else Msg( ' move pointer to desired position
            first ')
   EndIf

/------------------------/
/ keys that start with * /
/------------------------/

/$h previous field/
[*1] =
   Save( Wrap, Trap),
   Wrap( On),
   Trap( Off),
   Get( PrevFld, FldMask),
   If RC Eq 0 Then
      Say( Field, FldMask)
   Else Msg( ' first field')
   EndIf

/$h current field/
[*2] =
   Save( Wrap),
   Wrap( On),
   Get( Field, FldMask),
   Say( Field, FldMask)

/$h next field/
[*3] =
   Save( Wrap, Trap),
   Wrap( On),
   Trap( Off),
   Get( NextFld, FldMask),
   If RC Eq 0 Then
      Say( Field, FldMask)
   Else Msg( ' last field ')
   EndIf

/$h rest of field/
[*A] =
   Save( Wrap),
   Wrap( On),
   Say( Field, FldMask)

/$h set field definition/
[*H] =
   Cycle{ Set( FldMask, BG),
               Msg( 'background fields only');
          Set( FldMask, FG),
               Msg( 'foreground fields only');
```

```
          Set( FldMask, &hFF),
               Msg( 'full fields') }

/$h spell previous word/
[*4] =
  Save( Format, Caps),
  Format( Spell),
  Caps( Off),
  Get( Prevword),
  Say( Word)

/$h spell current word/
[*5] =
  Save( Format, Caps),
  Caps( Off),
  Format( Spell),
  Get( Word),
  Say( Word)

/$h spell next word/
[*6] =
  Save( Format, Caps),
  Caps( Off),
  Format( Spell),
  Get( Nextword),
  Say( Word)

/$h spell current word with caps/
[*B] =
  Save( Format, Caps, Table),
  Format( Spell),
  Caps( On),
  Table( 1),
  Get( Word),
  Say( Word)

/$h phonetic previous character/
[*7] =
  Save( Table, Caps),
  Table( 2),
  Caps( Off),
  Get( Row, Col-1),
  Say( Char)

/$h phonetic current character/
[*8] =
  Save( Table, Caps),
  Table( 2),
  Caps( Off),
  Say( Char)

/$h phonetic next character/
```

```
[*9] =
  Save( Table, Caps),
  Table( 2),
  Caps( Off),
  Get( Row, Col+1),
  Say( Char)

/$h ascii value of current character/
[*C] = Save( Format, Caps),
       Caps( On),
       Format( Ascii),
       Say( Char)

/$h text format/
[**] =
  Format( Text),
  Table( 1),
  Msg( 'text format')

/$h pronounce format/
[*0] =
  Format( Pronounce),
  Table( 1),
  Msg( 'pronounce format')

/$h spell format/
[*#] =
  Format( Spell),
  Table( 1),
  Msg( 'spell format')

/$h phonetic format/
[*D] =
  Format( Spell),
  Table( 2),
  Msg( 'phonetic format')

/------------------------/
/ keys that start with # /
/------------------------/

/$h search from top/
[#1] =
  Msg( 'enter search string'),
  Read( @searchst),
  If Length( @searchst) Eq 0 Then
  /null string entered/
    Msg( 'string search cancelled')
  Else
    Out( 'looking for '),
    Say( @searchst),
    Save( Wrap, Trap, Caps),
```

```
    Wrap( On),
    Trap( Off),
    Caps( Off),
    Mode( Pointer),
    Get( Top, Left),
    Get( @searchst, +),
    If RC Eq 0 Then
      Out( 'found '),
      Msg( Pointer)
    Else Get( Top, Left),
         Msg( 'not found')
    Endif
 Endif

/$h continue search/
[#2] =
  If Length( @searchst) Eq 0 Then
   /null string
           entered/
    Msg( 'Search string not defined')
  Else
    Msg( 'continuing search'),
    Save( Wrap, Trap, Caps),
    Wrap( On),
    Trap( Off),
    Caps( Off),
    Mode( Pointer),
    Set( r1, Row),
    Set( c1, Col),
    Get( Row, Col+1),
    If RC Eq 0 Then
      Get( @searchst, +),
      If RC Eq 0 Then
        Out( 'found '),
        Msg( Pointer)
      Else Get( r1, c1),
           Msg( 'not found')
      Endif
    Else
      Get( Row+1, Left),
      If RC Eq 0 Then
        Get( @searchst, +),
        If RC Eq 0 Then
          Out( 'found '),
          Msg( Pointer)
        Else Get( r1, c1),
             Msg( 'not found')
        Endif
      Else
        Msg( 'not found')
      Endif
    Endif
```

```
  Endif

/$h search from top for previous string/
[#3] =
  If Length( @searchst) Eq 0 Then
   /null string
        entered/
    Msg( 'Search string not defined')
  Else
    Out( 'looking for '),
    Say( @searchst),
    Save( Wrap, Trap, Caps),
    Wrap( On),
    Trap( Off),
    Caps( Off),
    Mode( Pointer),
    Get( Top, Left),
    Get( @searchst, +),
    If RC Eq 0 Then
      Out( 'found '),
      Msg( Pointer)
    Else Get( Top, Left),
         Msg( 'not found')
    Endif
  Endif


/$h lock keys/
[#4] =
  If( LockByte BitAnd &h80) Eq &h80
     /announce locks status/
     Then Out( 'insert ')
  Else Out( 'replace ')
  Endif,

  If( LockByte BitAnd &h40) Eq &h40
     Then Out( 'caps lock ')
  Else Out( 'no caps lock ')
  Endif,

  If( LockByte BitAnd &h20) Eq &h20
     Then Out( 'num lock ')
  Else Out( 'no num lock ')
  Endif,

  If( LockByte BitAnd &h10) Eq &h10
     Then Out( 'scroll lock ')
  Else Out( 'no scroll lock ')
  Endif,
  Msg(' ')

/$h current format/
```

```
[#5] = Msg( Format)

/$h current window/
[#6] =
  Out( 'top '),
  Out( Top),
  Out( ' left '),
  Out( Left),
  Out( ' bottom '),
  Out( Bottom),
  Out( ' right '),
  Msg( Right)

/$h lock status on or off/
[#B] =
  If State( LockStatus) Then
     Out( 'no '),
     LockStatus( Off)
  Else
    If IgnoreInsFlag Then
         LockStatus( On, -)
    Else LockStatus( On)
    Endif
  Endif,
  Msg( 'lock status')


/$h pointer position/
[#8] = Out( 'pointer at '),
       Msg( Pointer)

/$h current mode and position/
[#9] =
  If State( Pointer) Then
     Out( 'pointer mode at '),
     Msg( Pointer)
  Else Out( 'cursor mode at '),
       Msg( Cursor)
  Endif

/$h cursor position/
[#C] = Out( 'cursor at '),
       Msg( Cursor)

/$h cursor to pointer /
[#*] =
  Save( FastRoute),
  FastRoute( On),
  Route( Prow, Pcol),
  If NOT (( Prow Eq Crow) And ( Pcol Eq Ccol))
    Then FastRoute( Off),
    Route( Prow, Pcol)
```

```
  Endif,
  If (( Prow Eq Crow) And ( Pcol Eq Ccol)) Then
    Out( 'cursor to pointer at '),
    Msg( Cursor)
  Else
    Msg( 'cursor to pointer unsuccessful,'),
    Out( 'cursor at '),
    Msg( Cursor),
    Out( 'pointer at '),
    Msg( Pointer)
  Endif,
  Mode( Cursor)

/$h pointer mode/
[#0] = Mode( Pointer),
       Out( 'pointer mode at '),
       Msg( Pointer)

/$h pointer to cursor /
[##] =
  Mode( Pointer),
  Get( Cursor),
  Out( 'pointer to cursor at '),
  Msg( Pointer)

/$h cursor mode /
[#D] = Mode( Cursor),
       Out( 'Cursor mode at '),
       Msg( Cursor)

/$h start ignoring 1 or more characters/
[#H] =
  Msg(' enter characters to start ignoring'),
  Read( @ignorest),
  If RC Eq 0 Then          /null string entered /
    Msg(' cancelled ')
  Else
    Out(' start ignoring '),
    Save( Format),
    Format( Spell),
    Say( @ignorest),
    AddIgnore( @ignorest)
  Endif

/$h stop ignoring 1 or more characters/
[#S] =
  Msg(' enter characters to stop ignoring'),
  Read( @ignorest),
  If RC Eq 0 Then          /null string entered /
    Msg(' cancelled ')
  Else
    Out(' stop ignoring '),
```

```
    Save( Format),
    Format( Spell),
    Say( @ignorest),
    DelIgnore( @ignorest)
  Endif


/------------------------/
/ keys that start with A /
/------------------------/

/$h set pitch/
[A1] = Keypad( Pitch),
       SetPitch( Pitch),
       Out( 'pitch '),
       Msg( Pitch)

/$h set rate/
[A2] = Keypad( Rate),
       SetRate( Rate),
       Out( 'rate '),
       Msg( Rate)


/$h caps on or off/
[A4] =
  If State( Caps) Then
     Out( 'no '),
     Caps( Off)
  Else Caps( On)
  Endif,
  Msg( 'caps')

/$h spaces on or off/
[A5] =
  If State( Spaces) Then
     Out( 'no '),
     Spaces( Off)
  Else Spaces( On)
  Endif,
  Msg( 'spaces')


/$h graphics on or off/
[AB] =
  If State( Graphics) Then
  Out( 'no '),
  Graphics( Off)
  Else Graphics( On)
  Endif,
  Msg( 'graphics')
```

```
/$h wrap on or off/
[A8] =
  If State( Wrap) Then
     Out( 'no '),
     Wrap( Off)
  Else Wrap( On)
  Endif,
  Msg( 'wrap')


/$h numbering on or off/
[A9] =
  If State( Numbering) Then
     Out( 'no '),
     Numbering( Off)
  Else Numbering( On)
  Endif,
  Msg( 'numbering')



/------------------------/
/ keys that start with B /
/------------------------/

/$h first word of line/
[B1] = Get( Row, Left),
       Say( Word)

/$h middle word of line/
[B2] = Get( Row, Right),
       Get( Row, (Right Div 2)),
       Get( Word),
       Say( Word)

/$h last word of line/
[B3] = Get( Row, Right),
       Get( Word),
       Say( Word)

/$h rest of word above/
[B4] = Get( Row-1, Col),
       Say( Word)

/$h rest of word/
[B5] = Say( Word)

/$h rest of word below/
[B6] = Get( Row+1, Col),
       Say( Word)
```

```
/------------------------/
/ help and stop keys     /
/------------------------/

[H] = Help

/$h stop speech now/
[S] = Stop
```

VITA

Aouni Hallal

Candidate for the Degree of

Master of Science

Thesis: PROVIDING TEXT-MODE ACCESS TO BLIND USERS WITH AN APPLICATION USING TEXT-TO-SPEECH SYNTHESIS

Major Field: Computer Science

Biographical:

Personal Data: Born in Habbouche, Lebanon, on June 23, 1967, the son of Samih and Fatima Hallal.

Education: Received Bachelor of Science degree in Management Science and Computer Systems from Oklahoma State University, Stillwater, Oklahoma in December 1988. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in May 1995.

Experience: Employed by Oklahoma State University, Wellness Center as a graduate research assistant; Oklahoma State University, Wellness Center, 1991 to 1993.

Professional Memberships: Phi Kappa Phi Honor Society, Golden Key National Honor Society, Student Government Association.