RC 19825 (06/02/94)
Computer Science/Mathematics

IBM Research Report

How to Make OS/2 Talk ... And Why Access to OS/2 with IBM Screen Reader/2

Jim Thatcher
IBM Research Division
T.J. Watson Research Center
Yorktown Heights, NY 10598

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden . T.J. Watson . Tokyo . Zurich

How to Make OS/2 Talk ... And Why
Access to OS/2 with IBM Screen Reader/2

Jim Thatcher

Interaction Technology
Mathematical Sciences Department
IBM Research
Yorktown Heights, NY 19598

Thatch at Watson.IBM.com
Compuserve 73234,1544
914-945-2193 (Fax 914-945-3434)

Abstract Wouldn't it be nice if, as you moved around the desktop, or an application's dialog, the items you traversed spoke to you. You're doing an 18 disk installation, working at something else, and you hear, "Insert disk 9 in drive a." "Maybe it would be nice," you say. If you are blind, it is absolutely essential.

I will explain the basics of speech output, and how Screen Reader/2 works. The key to this is the Profile Access Language which can be used by developers to add speech for purposes of of prototyping and testing a user interface with speech added. With this background you will be able, I hope, to ask, and even answer, your own questions about a talking OS/2.

Screen Reader/2 is IBM's access system for OS/2, providing computer users who are blind and visually impaired access to the graphical user interface of Presentation Manager, to Windows programs running under OS/2, and to text-mode DOS and OS/2 programs.

1. Making OS/2 Talk

If you use OS/2 everyday, you know how wonderful it is to be able to be doing three or four things at once. You know how easy it is to just point and click to launch an application. You know how “friendly” it is. Imagine if OS/2 talked to you too. Well it can!

There is one obvious reason to make OS/2 talk. OS/2 talks with Screen Reader/2 so that people who are blind can use all the power and versatility and excitement of OS/2, like their sighted colleagues.

But, are there other reasons to make OS/2 talk?

Educational software developers such as Bröderbund, Crayola, Davidson and EduQuest have been using combinations of digitized and synthesized speech for years. EduQuest's Writing to Read is really based on the theory that people learn more effectively when they use different modalities.

If you wanted to use your computer over the telephone – not with terminal and a modem, just the telephone – then you would need a talking operating system.

And what about laptops in automobiles, or personal digital assistants, where the displays are either distracting or too small? A talking OS/2 could come in handy then.

Besides telling you about Screen Reader/2 for OS/2 users who are blind, I also want to recommend Screen Reader/2 to OS/2 developers. Screen Reader/2 provides an environment for adding speech to OS/2 or to applications, for prototyping and testing new ideas in audio output. We will discuss this capacity below.

2. How Can I Make OS/2 Talk?

Since speech output may be unfamiliar to many developers I want to begin by describing the available computer-based speech alternatives; you can used digitized speech, speech generated by a recording, or you can use synthesized speech, speech generated by your computer.

2.1 Digitized or Synthesized Speech

Digitized speech is recorded, in a studio or at your computer, and stored in a .wav file. You have to have some kind of a multimedia card that converts the .wav file back into speech. You also need a multimedia interface like MMPM/2.
Variation in the quality of digitized speech output is minimal. Digitized speech sounds good, except for the possibility that you just don't like the voice.

Synthesized speech is speech produced by computer from input strings of ASCII text. There is tremendous variation in the quality of synthesized speech. I have heard a children's game that uses software to produce synthesized speech on the PC speaker. Technically it was synthesized speech, but the words were mostly unintelligible.

Of text-to-speech products on the market today there are four hardware categories:

Peripheral serial devices
Internal cards
PCMCIA cards
Multimedia cards with text-to-speech capability

Peripheral serial text-to-speech synthesizers have been around for the longest time (perhaps 15 years) and, as a group, provide a consistent and simple interface. There are dozens of serial synthesizers on the market today and they have a tremendous range in speech quality, size, and price. Prices range from around $100 to over $2000. Sizes vary from that of a notebook computer to the size of a personal digital assistant or large hand-held calculator.

There are far fewer text-to-speech internal cards; I think there are only five or six cards for ISA machines, one for MCA, and only two PCMCIA cards.

The advent of text-to-speech in multimedia cards is a relatively new phenomenon. The Sound Blaster card, by Creative Labs, has had a synthesized speech capability for a while, and recently (March 1994) they introduced a card which includes high quality text-to-speech. IBM's AudioVation multimedia card also has text-to-speech capabilities.

Recorded, digitized speech is far superior to anything that can now be produced by text-to-speech synthesizers. But superior in what sense? The answer has to be that the speech sounds ‘natural’.

How else can speech output be compared? Consider a simple example; a search dialog for entering a search string to search for text in a database or in an editor. And assume it is interesting to *hear*:

```
Enter a search string
```

That text string is only 21 bytes. My recorded version of that message, depending on recording format ranged from 30,000 to 100,000 bytes. That is a *tremendous* amount of storage to take up for a 21 character message.

In this simple sample dialog, after entering a search string, it would be at least as likely for us to want to hear a confirmation of what we just did:

```
you are searching for "speech synthesis"
```

You can speak that with synthesized speech and you cannot speak that with digitized speech. The distinction here is very important and often missed. The part of the above message, 'you are searching for' is no problem. That could be prerecorded. But the search string cannot be prerecorded, because no one knows what it might be. Synthesized speech will handle the string 'speech synthesis' from the message above as well as (or as poorly as) it handles 'jtisby psdhfodvx'!

The bottom line is, if you want to make OS/2 talk in any way but with canned messages, then you must use text-to-speech synthesis.

2.2 More on Text-to-Speech Devices

Whether you are using a serial device, or an internal card, the basic format for doing text-to-speech is the same. There are two kinds of data you send to the synthesizer. Text to be spoken and control strings which can be, and very often are, embedded in the text.

The text is whatever you want to speak. The control sequences can, depending on the synthesizer, change any of the following speech characteristics:

Pitch
Speech rate
Volume
Intonation
Resonance
Breathiness

In addition to modifying speech characteristics, control strings are used with some synthesizers to produce non-speech audio, to send phonemes and bypass the text-to-speech processing, to change languages and to perform *indexing*. Indexing provides communication so that the application can be informed exactly where, in the input text string, the speech synthesizer is talking.

For someone who depends on speech output, control of speech may be more important than speech quality. For them, the maximum speech rate, the responsiveness to starting and stopping speech, the articulation, and distinction between similar sounds, like vowels, and like the 'b v t d g p' sounds; these criteria may be far more important than how 'natural' the speech sounds to the uninitiated.

3. Making OS/2 Talk With Screen Reader/2

If you decide you want to try to use synthesized speech in the computer interface, or in your application, then you could tackle a whole spectrum of issues ranging from the interface to the text-to-speech device, to accessing data on the display. All of these are solved for you with Screen Reader/2. This system was developed to make OS/2 talk as an access system for users who are blind.

The key for application developers' use of Screen Reader/2 is the *Profile Access Language (PAL)*. PAL is the programming language in which profiles are written and profiles drive everything that Screen Reader/2 does. In the subsequent sections, I will

describe how Screen Reader/2 works as an access system, and then, describe PAL in overall structure and with several examples.

In this section, I will describe the contents of the Screen Reader/2 package. Then, I will discuss two Screen Reader/2 concepts, the *state* and the *view*. It is best to have these ideas in mind before taking the 'test drive' in which I will indicate the kind of audio responses one gets when using Screen Reader/2 with OS/2.

3.1 The Screen Reader/2 Package

Screen Reader/2 consists of an 18-key keypad, a keypad cable, audio cassettes, printed and braille documents, and diskettes with on-line documentation and software.
The keypad is attached through the mouse port. If the mouse port is not available, you can install an adapter card that simulates it. It is possible for Screen Reader/2 can use the PC keyboard instead of the keypad.

The Screen Reader/2 user must have a separate serial text-to-speech device.

Screen Reader/2 speaks information from the display or changes settings in the speech environment as the result of a user request through the keypad or because of some *autospeak*.

3.2 The Screen Reader/2 View

For text-mode computing, screen access programs could refer to display memory to know everything that was displayed at any time. For the graphical user interface, display memory contains only pixels, dots of color, and this is essentially useless for screen access. Screen Reader/2 creates a model of the display, called the off-screen model (OSM), containing all the displayed text and icons together with detailed position, font and window information.
Screen Reader/2 organizes that information in what is called a *view*. The view is determined by a window, usually the frame of the active window, and includes all text and icons drawn to that and any child windows.

Screen Reader/2, unlike other screen access programs for a GUI, structures a view as if it were a text window. A *row* is a text string all of whose characters have the same baseline *and* the same window handle. The rows are sorted by baseline. This text interpretation of the view is extended to make the assumption that all rows have the same length, that being the length of the longest text string in the view. Shorter text strings are filled out with spaces, just as in text-based computing.

With this translation from the graphic display to a text model, we use terms like *line* and *column*, recognizing that it is somewhat an abuse of terminology. It is terminology that we have found to be a great a simplification for the blind users of the graphical user interface.

3.3 The Screen Reader/2 State

There are many components of the Screen Reader/2 state that can be set by the user. The state effects the way commands are interpreted. The following list is not exhaustive but it illustrates some of the main state components.

- Mode cursor or pointer (*Mode*).
- Format: text, pronounce, spell or phonetic (*Format*).
- Ignore capitalization or not (*Caps*).
- Treat entire screen as single line or not (*Wrap*).
- Use the dictionary or not (*Dictionary*).

- Error announcement or not (*Trap*).
- Speech characteristics (*Rate, Pitch, ...*).
- Current output device (*Device*).

The following are GUI specific:

- Hear drawing noise or not (*Noise*).
- Ignore icons or not (*Icons*).
- Current view.

When Screen Reader/2 is in *cursor* mode, requests for data from the display are relative to the cursor (insertion bar) position. If you ask for the next line, you get the line below the cursor. If the cursor does not move and the 'next line' is requested again, the same line is announced.

Screen Reader/2 has its own *pointer*, which cannot be seen and which travels around as requests are made. In pointer mode, 'next line' requests just move down the view.

In general now, we can talk about the current line, word, or character as meaning the line, word, or character at the cursor if the mode is cursor, or at the Screen Reader/2 pointer, if the mode is pointer.

There are four reading *formats* called *text*, *pronounce*, *spell*, and *phonetic*. In text format, Screen Reader/2 tries to read as if reading a book aloud. In pronounce format, the reading is the same but punctuation and blank lines are also announced. In spell format, all words are spelled. In phonetic format, everything is spelled using the International Phonetic Alphabet.

3.4 A Test Drive

Before we discuss more technical matters relating to the Profile Access Language, I will give the reader an idea of how a Screen Reader/2 user hears the OS/2 desktop. This test drive, I hope, will stimulate interest in the more technical subjects.
Bring up the Window List with Ctrl+Esc. You hear:

```
Window list, Desktop - Icon view
```

The first is the title of the window (Window List), then the selected item (Desktop - Icon view).

You can double check the selector with the appropriate sequence on the Screen Reader/2 keypad:

```
selector, Desktop - Icon View
```

Or, you could check what kind of window is being read:

```
container, PM
```

It is a PM (Presentation Manager) window (as opposed to DOS or Windows) and its window class is *container*.

As the user moves the selection cursor around the items in the Window List, each item is announced automatically. The entire window could be read as well. Mine sounds something like:

```
Window List, System Menu Icon,
Minimize Hidden Icon, Maximize Icon,
Desktop - Icon View, Scroll Up-arrow Icon,
OS/2 window, System ...
```

This kind of output should suggest that reading the whole window is not something one wants to do very often.

Let's bring the Desktop to the foreground.[1] We're still using the Window List and we find ‘Desktop’ with the arrow keys, or with 'D', and then Enter. When the DeskTop folder opens, you hear:

```
Desktop - Icon View, OS/2 Window
```

That's the title and the currently selected item again. Screen Reader/2 could be used to read the whole window, or read it line at a time. It is easier to move around using letter or arrow keys. I repeatedly use 'S' and get ‘Shutdown,’ ‘Special Needs,’ ‘Start Here,’ and then ‘System’ (I have renamed the OS/2 System folder to just ‘System,’); this is what I want, so I press Enter.

Again, the title and the selected item are spoken:

```
System, Drives
```

‘Drives’ is the highlighted item in the OS/2 System folder.

Let's listen to a couple more things in here. These could have been checked out in any of the previous windows. We heard above that the Window List was a *container*. If we ask Screen Reader/2 for the window information after reading the system menu icon we hear:

```
PM, Menu
```

If you read the title, and ask for window information, you hear:

```
PM, Titlebar
```

These queries of the window information demonstrate that Screen Reader/2 knows the *window class* of the current window. This is essential information, not so much for titlebars and containers as for check boxes, radio buttons and entry fields, where the user needs to know the window class in order to know what to do.

In our tour of OS/2 we are still using the OS/2 System folder; use 'P' to get to the Productivity Folder and press Enter. Now use ‘C’ for the Calculator and press Enter. This time there was more information than might otherwise have been expected. You hear:

```
Calculator, Num Lock on,
memory contents zero, zero
```

‘Calculator’ is the title of the window as we have heard titles before. ‘Num lock on’ announces the status of that lock key which was turned on by the calculator application.

The next two announcements are different, they should be surprising, and they exemplify an extremely important concept. We are being told that the memory contents are zero and the current subtotal is also zero. These are application-specific announcements. Screen Reader/2 knows about the current foreground process, and based on that information, has added screen reading function specific to the calculator.

The calculator is *not* a high-powered OS/2 application, but it illustrates the importance of automatic announcements from the display.

When you use the Calculator, the number keys are spoken as you press them, and the answer is automatically spoken. The sequence of keyboard keys, ‘12-25=’ is announced:

1 The Screen Reader/2 OS/2 Desktop is non-standard. You can bring it to the foreground just like any other folder. You can't minimize, hide, or close it, however.

```
1, 2, minus, 2, 5, equals, negative 13.
```

Screen Reader/2 here echoes keystrokes *and* automatically reads the result of the Calculator operation.

All the information that is automatically spoken in this Calculator example is information that could be found using standard Screen Reader/2 read requests from the keypad. The user could check the results line each time a calculation was complete. That would not be very efficient. Instead, Screen Reader/2 makes this application into a talking calculator through the use of an application-specific profile.

4. Profiles and The Profile Access Language

Profiles are the key to Screen Reader/2. Everything that happens is determined by the currently active profile. Even which profile is active is a result of some profile fragment!

In this section I will outline the general form of PAL, and then illustrate some of the important commands and control features. Basically, PAL is a Pascal style language and most of the syntax can be inferred from the examples. I will return to the test drive scenario, so that the reader can understand how PAL code relates to some of the announcements there.

4.1 The Larger Structure of PAL

A profile is written using any text editor, like other programs, and it is compiled. The compiled code is code for a hypothetical computer which is interpreted by Screen Reader/2.

The base profile is loaded is loaded when Screen Reader/2 is started. That profile cannot be removed. If Screen Reader/2 is so set up, an application-specific profile is loaded when an application starts, and that profile is removed when the application closes. Lastly, one can use the keypad (or PAL) to add or delete profiles.

A profile consists of variable, constant, and procedure declarations, *key sequence* definitions and *autospeak* definitions.

Variable declarations have the form:

```
Var I, M: Integer;
Var A: autohandle;
```

The types are Boolean, integer (32 bit), string (various lengths), arrays and two special basic types, *autohandle* and *viewhandle* which refer to autospeaks and views respectively.

A procedure declaration takes the form:

```
Proc <procedure name> [<parameter declarations>]
   [ returning <type> ] is
      <statement sequence>
```

A key sequence definition takes the form:

```
{ <key sequence> }  [ <help comment> ]
  <statement sequence>
```

The keypad has 18 keys named like the telephone keypad, with 'A B C D' down the side, and two keys labeled Help (H) and Stop (S) to the right of the lettered keys. A key sequence is just a string of key names like ‘0A’. Key sequences can also include chords consisting of exactly two keys: ‘(0A)’ is the chord obtained by pressing both the 0 and A keys.

The most exciting construct in PAL – in Screen Reader/2 for that matter – is the autospeak. This is the concept that differentiates IBM Screen Reader/2 from other screen access systems. Others do have automatic speaking, but the totally programmable automatic speaking is unique.

Autospeaks take the form:

```
Autospeak <autohandle>
   watch <expression> | <event>
     do <statement sequence>
```

The autohandle is, as mentioned above, just a way to refer to the autospeak, to turn it on or off for example. If an expression is being watched, it is evaluated approximately every 100 milliseconds. If the value of that expression changes, then the statement sequence, called the *body* of the autospeak, is executed. There is a reserved variable called *trigger* which, in the body of the autospeak, has the same type as the watched expression, and whose value is that which triggered the execution.

In addition to expressions, autospeaks can watch events including changes in cursor or selector positions, in process or view, or events that monitor communication from the serial text-to-speech device, or even external events generated by another process. The value of the reserved variable, *tigger*, depends on the event. For example, *EventKeyPressed* is triggered for keys on the PC keyboard. In this case *trigger* is an integer containing ASCII and scan code information for the key that has been pressed.

PAL has standard control constructs for conditional execution and iteration:

```
If <boolean> Then
   <statement sequence>
[else
   <statement sequence>]
Endif;

While <boolean> Do
   <statement sequence>]
EndWhile;
```

These constructs, together with the fact that autospeaks can watch procedures, provide a kind of abstract universality with respect to the off-screen model. This means that you can, *in principal*, not necessarily in practice, program whatever speech output you want using Screen Reader/2 and the Profile Access Language.

4.2 Commands and Functions

We now have the structure of PAL; the commands and functions of PAL put meat on the bones of this structure. Rather than enumerate all the commands and functions, I will give relatively simple examples and discuss how parts of those examples can be elaborated or modified, thus, I hope, giving a flavor of the Profile Access Language.

Example 1

```
{2} 'current line'
    Get( row, 1); Say( Line);
```

Probably the simplest key definition in the Screen Reader/2 base profile, this example defines key 2 to say the current line. The *Get* command positions Screen Reader/2 to the position specified by its arguments. In this example, *row* is the cursor or pointer row, depending on mode. The *Say* command speaks it argument (one line in this case) using the current Screen Reader/2 state. E.g., if the format were spell then key 2 would spell the current line.

There are many forms of the *Get* command, and one related command, *Find* which searches for text strings in the current view. These all have the same kind of effect, to position Screen Reader/2 ready so speak something. Here are some other examples of this family of commands:

- *Get(Cursor | Mouse)*. Move to the cursor (insertion bar) or to the mouse pointer.
- *Get(x, y, DeskTop)*. Get can be used to position the pointer to a pixel position as well as the row, column view coordinates.
- *Get(Word | NextWord | PrevWord)*. Move along by words, positioning at the beginning of the current word, the next or previous word, respectively. You can do this for sentences as well.
- *Get(Field, Mask)*. This form of the command looks to colors and fonts. A field is defined to be a contiguous string of text with the same display attributes and the the *Mask* parameter specifies which combination (foreground or background color, font, pitch, style) of attributes are wanted. This form of the command allows *NextField* or *PrevField* as well.

The *Say* command also has several forms.

- *Say(Word | Line | Sentence)*.
- *Say(string)*.
- *Say(Field, Mask)*.

Reiterating, the important thing about the *Say* command is that Screen Reader/2 uses all the information in the current Screen Reader/2 state to speak the requested text. In contrast, the *Msg* command sends its argument, or arguments, to the synthesizer independent of the Screen Reader/2 state.

Example 2

```
Var time: autohandle;
Autospeak time Watch DateTime( DT_MINUTE) Do
  if (trigger mod 15) = 0 then
     Msg( 'it is ', DateTime( DT_HOUR),
          ' hours ', trigger, ' minutes')
  endif;
```

DateTime is a function (the PAL version of the corresponding OS/2 function) returning date and time values. Constants like DT_MINUTE and DT_HOUR are defined in a PAL include file.

This autospeak will trigger every minute because the value of DateTime(DT_MINUTE) changes every minute. The conditional is checking if the minute is a multiple of 15. If so, a message is announced. The *Msg* command takes several arguments, including numbers. In particular, you hear

```
it is fifteen hours thirty minutes
```

at 3:30 PM. The reserve variable, *trigger*, being the value of the expression that triggered the autospeak, has the value 30 at 3:30 PM.

That is not a very ‘user friendly’ clock. I hope the reader (if he or she is a programmer) would have a feeling of how the clock could be greatly improved. Beyond better words, like ‘it’s half past three,’ you could use the pc speaker for chimes with the the following commands.

- *Beep (frequency, duration)*
- *Sleep (duration)*

Even better, we can get sound effects with string commands going to multimedia device manager within PAL.[2]

- *MDM_String('open chimes.wav alias chimes')*
- *MDM_String('play chimes')*

The *Read* command accepts input from the keyboard, and the *Keyval* command takes its input from the Screen Reader/2 keypad. Using these commands one could expand this clock example to include alarms. In the following example I have avoided the details of input conditions taking only an hour and message for the alarm.

Example 3

```
Var hour : Integer;
Var message : String;
Var alarm : Autohandle;

{A} 'get input for alarm'
var time : string;
   Msg( 'enter time for alarm');
   Read( time );
   hour := Int ( time);
   Msg( 'enter alarm message');
   Read( message);

Autospeak alarm watch DateTime (DT_HOUR) = hour Do
   If Trigger Then Msg( message); Endif;
```

In this example key definition for key A, the *Msg* command gives prompts followed by the *Read* command to get data. The hour is read in as a string and converted to an integer with the *Int* function. The message to accompany the alarm is just read in as a string.

The simple autospeak for the alarm watches the expression DateTime(DT_HOUR) = hour which will *change* to True when the hour is reached, and then the alarm message is announced with the Msg command.

The following example illustrates functions related to the state of the operating system and windowing system, similar to that information obtained in the test drive earlier.

Example 4

```
{8} 'window information'
    Say( program);    -- Current foreground executable
    If view.type = PM
      Say( title );   -- Title of foreground window
      if view.style Bitand WS_MINIMIZED then    -- minimized?
         msg( 'icon ');
      endif;
      Say( View.class ( ViewFromRowCol( row, col ) ) );
   endif;
```

The read-only variable (function with no arguments) *program* contains the name of the foreground executable. The *view* function appears in this example with three modifiers, *type*, *style*, and *class*. The *type* is a numeric value distinguishing between the kinds of applications that can be running under OS/2, PM, Seamless, DOS or OS/2 window, or DOS or OS/2 full screen. The *style* and *class* are familiar (for developers) PM and Windows concepts. The modified *view* function also appears in the example both without arguments, when it applies to the current view, i.e., current main window,

2 Multimedia string commands are a recent addition to Screen Reader/2 and not part of the current Screen Reader/2 product.

and with a view argument. In particular, *ViewFromRowCol* returns the handle of the view containing the text where Screen Reader/2 is currently positioned.

The next example illustrates the *mouse* command to position the mouse pointer and to click.

Example 5

```
{(23)5} 'double click mouse button 1'
   Msg( 'click click');
   Mouse( row, col );    /* Move mouse to current row and col */
   Mouse( Button1, ButtonDoubleClick);
```

There also is a *mouse* function, which returns the mouse pixel coordinates or the handle of the window (view) containing the mouse. In addition a *Map* function is provided for mapping between various pixel coordinate systems, and the row and column coordinate system.

As mentioned above, autospeaks can watch events as well as expressions. The following example of an autospeak fragment watches for the event that Screen Reader/2 will generate when it detects that a keyboard key has been pressed.

Example 6

```
Var WatchKey : Autohandle;
Autospeak WatchKey Watch EventKeyPressed Do
var TempChar : integer;
    TempChar := Trigger BitAnd &H00FF; -- Get ASCII code
    If TempChar = equal then Msg( 'equals');
    ElseIf TempChar = hyphen then Msg( 'minus');
    ElseIf TempChar = Asterisk then Msg( 'times' );
    ...
    ElseIf TempChar <> 0 Then Say( TempChar);
    Endif;
```

This autospeak is similar to the one in the calculator profile used to echo the keys as we saw in the Test Drive. There are several integer constants used here that are declared in the separate include file mentioned earlier.

The following autospeak is the simplest kind of a monitor of changes in data on the display.

Example 7

```
Var Status autohandle;
Autospeak Status watch Display( 1, 76, 5) Do
  Msg( trigger );
```

Display is a function which returns some part of the text in the current view. Its arguments are a starting position and a length. When running text-mode Lotus 1-2-3 this autospeak will announce the status indicator (‘Ready’, ‘Point’, ‘Help’, etc.).

Example 8

```
Var MarginBell autohandle;
Autospeak MarginBell Watch Ccol>72 Do
   if trigger then Beep( 666, 55); endif;
```

This autospeak watches the Boolean expression *Ccol>72*. The read-only variable (function with no arguments) *Ccol*, is the cursor column. When it exceeds 72 the watched expression changes to true and the PC speaker beeps. While the cursor stays in that region, there is no action. When the cursor moves left, the expression becomes false, but, because of the conditional, there is no sound.

5. Conclusion

On the one hand Screen Reader/2 is a remarkable screen access system for blind computer users who want to access Windows, DOS, or OS/2 applications. We have illustrated very briefly how OS/2 sounds as one navigates around the Desktop.
In addition, however, Screen Reader/2 comes with a complete programming language, PAL. The IBM Screen Reader/2 product includes about 63 profiles all written in PAL, amounting to about 800,000 bytes of profile source code.

Besides the interest in a talking OS/2 for blind users, it is my belief that Screen Reader/2 provides a tool for prototyping speech and non-speech audio output for applications. It is the PAL that gives me confidence in saying that.

Acknowledgement

Screen Reader is the result of the efforts of many people. The author has had a project in the Mathematical Science Department of IBM Research for almost 10 years and several people have worked on that project. The Special Needs Organization of the IBM PC Company is the organization responsible for making a research project into a product. They took PC SAID and made IBM Screen Reader. Many people have been involved there as well, developers, planners, and writers. Fran Hayden is one of those writers and she has been invaluable in making this paper more readable. Finally there are users. Those individuals both inside and outside IBM provided both the direction and motivation for the Screen Reader effort.