**RC 19826 (06/02/94)**
**Computer Science/Mathematics**

# IBM Research Report

## Making OS/2 Talk: Programming Speech and Non-Speech Audio

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**

# Making OS/2 Talk
# Programming Speech and Non-Speech Audio

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

Screen Reader/2 makes OS/2 talk. It makes OS/2 talk whether you are running DOS, Windows, or OS/2 applications. Screen Reader/2 makes OS/2 talk so that blind computer users can have access to OS/2.

However, Screen Reader/2 is more than an access system. It provides a programming language to produce speech and non-speech audio from OS/2. Everything that Screen Reader/2 does for its blind user, it does under the control of profile code written in the *Profile Access Language*, *PAL.*

It is the purpose of this paper to introduce the idea of PAL as a tool for adding audio in the interface.

# 1.0 Introducing Screen Reader/2

Try to imagine using a graphical user interface such as OS/2 or Windows on your system with the display turned off. What information would you need? What windows are open? What programs are running? What is the foreground window? What is the program name and title? What type of control is currently selected and what is the text associated with that control? Thousands of computer users who are blind or visually-impaired are faced with these questions each day.

Screen Reader/2 is a product that can help them answer these questions. It enables blind and visually impaired people to use OS/2, seamless Windows, and DOS applications, all running under OS/2 Version 2.1. Everything the blind user hears is a consequence of some *profile* written in the Screen Reader/2 *Profile Access Language*, or *PAL.*

Screen Reader/2 is more than an access system for visually impaired users. It augments the OS/2 user interface with both speech and non-speech audio, and it does this in a completely programmable way. As such it also serves as a prototyping system for adding audio to the computer human interface. New and different profiles could be written to add audio to a particular application or to the graphical user interface in general.

This paper first presents a 'test drive' of Screen Reader/2. It then discusses the concept of *view*, and of *state*. The general structure of the Profile Access Language is then presented, its commands and functions are briefly described with examples.

The final section on watching events and procedures highlights one of the most powerful features of

## 1.1 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.

The Screen Reader/2 user must have his own serial text-to-speech synthesizer; and the product supports about 15 models.

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*) – more later.

## 1.2 A Test Drive

Let's look at some examples of how the Screen Reader/2 user interacts with OS/2.

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 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 it is a 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. We're still using the Window list and we find ‘Desktop’ with the arrow keys, or with 'D', and then Enter. (The sighted mouse user would click on the Desktop folder rather than be bothered with the Window List. The blind user with Screen Reader/2 has an equally fast track, pressing a preset key sequence for the Desktop folder.)

When the DeskTop folder opens, you hear:

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

That's the title and the current 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,’ which 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 look at a couple more things in this window. These we could have 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.

A key sequence from the keypad tells us that the current application is calc.exe, and another request tells us that the *profiles* are calc and srd2. We are going to discuss the very important issue of profiles in the next section. Suffice it to say for this test drive that the behavior of Screen Reader/2 depends on the Screen Reader/2 profile that is loaded and that, in turn, can depend on the foreground application.

The calculator is *not* a high powered OS/2 application, but it illustrates well 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 or error information is automatically spoken. The sequence of keyboard keys, ‘12-25=’ is announced:

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

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

It is important to understand that 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. To make using the Calculator easier, there is a Screen Reader/2 profile that is automatically loaded each time the Calculator starts. That profile provides the speech interface to the calculator.

## 2.0 Basic Concepts

There is a framework that we need before we can usefully talk about Screen Reader/2 profiles. That framework is built on the Screen Reader/2 concepts of *view* and *state*. The view is the way Screen Reader/2 organizes the text and icons displayed on the graphical screen, and the state, which includes the current view, effects how most of the profile commands work.

## 2.1 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 (refer to Schwerdtfeger [8]).

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 (compare Mynatt and Webber [7]). 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 an 'abuse' that we have found to be a great a simplification for the blind users of the graphical user interface.

On my system, the view for the Window List we heard in the test drive has 14 rows and 19 so-called columns. The first six rows are as follows, laid out with row and column numbers.

```
    1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
1   W i n d o w   L i s t
2   X
3   Y Z
4   D e s k t o p   -   I c o n   V i e w
5   O S / 2   W i n d o w
6   S y s t e m   -   I c o n   V i e w
```

'X, Y, and Z' each represent one character position and stand for the system menu icon, the minimize hidden icon, and the maximize icon, respectively.

## 2.2 The Screen Reader/2 State

There are many components of the Screen Reader/2 state that can be set by the user either with the keypad (using profiles supplied with the product) or as part of PAL programming to be discussed below. 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.

Two of these state components require a little explanation: *mode* and *format*.

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 moves down the view.

In general now, we can talk about the current line, word, or character as meaning mean 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.

## 3.0 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.

## 3.1 The components 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 (refer to Jameson [6]) which is interpreted by Screen Reader.

A profile consists of variable, constant, and procedure declarations, together with *key-sequence* 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), array, 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>
```

In the evolution of Screen Reader/2 dating back to 1984, PAL was seen simply as a way to allow user flexibility in the definitions of the keys on the keypad. A key 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 right side, and two keys labeled Help (H) and Stop (S) to the right of the lettered keys. A key sequence is just a sequence of key names like a like **0A**. Key sequences can also include chords consisting of exactly two keys. For example, *(0A)* is the chord that is performed by pressing both the 0 and A keys. The Help key on the keypad puts Screen Reader/2 in to Help state, then key sequences entered on the keypad repeat the help comment coded in the corresponding PAL key definition.

The most interesting 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 <event> | <expression>
     do <statement sequence>
```

The autohandle is just a way to refer to the autospeak, to turn it on or off for example. If an event is being watched, its occurrence triggers the execution of the statement sequence.

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.

Here are some autospeaks examples.

***Example 1***

```
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 (just the PAL version of the corresponding OS/2 function) returning date and time values. This autospeak announces the time every 15 minutes. In particular you hear

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

at 3:30 PM. I hope the reader will imagine that some simple PAL coding will change this staid announcement to a friendly voice clock. A time speaking profile is part of the basic Screen Reader/2 package.

***Example 2***

```
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 123 this autospeak will announce the status indicator (‘Ready’, ‘Point’, ‘Help’, etc.).

***Example 3***

```
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.

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.

***Example 4***

```
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' );
    ...
    Endif;
```

The fragment of an autospeak in Example 4 watches for the event that Screen Reader/2 will generate when it detects that a keyboard key has been pressed. 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 a separate file supplied with PAL, e.g.,

```
const equal = &h3D;
```

&h is used to signal an integer constant in hex notation. *BitAnd*, in the first statement of the Example 4, is the bitwise *and* operator.

Pal comments are allowed with either the C-like notation (PAL's can be nested), '/* ... */', or ADA notation, '--' to end of line.

## 3.2 Control Statements

In addition to the conditional construct illustrated above, PAL has two iteration constructs, *While* and *For*:

```
For <assignment> to <expression> Do
   <statement sequence>
Endfor;

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*, program whatever speech output you want from the off-screen model, using Screen Reader/2 and the Profile Access Language.

## 3.3 Commands

At this point, the reader should have an understanding of the over-all structure of PAL, ready now to add serious content with commands and functions, used in autospeaks, key definitions, or procedures. The following is a brief list (again, not exhaustive) of Screen Reader/2 commands. It is intended to illustrate the kinds of actions one can take within a Screen Reader/2 profile. The list emphasizes commands that might be used outside the area of screen reading, *per se*.

| | |
|---|---|
| *After* | Schedule a Pal procedure. |
| *Beep* | Sound PC speaker, frequence and duration. |

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

| | |
|---|---|
| *Find* | Find a string. |
| *Get* | Position Screen Reader/2 pointer. |
| *Keyin* | Send keyboard keys to the foreground application. |
| *Keypad* | Send keypad keys to Screen Reader/2. |
| *Keyval* | Read from Screen Reader/2 keypad. |
| *MDM_String* | Send multimedia string commands[1]. |
| *Mouse* | Take mouse actions, clicks and positioning. |
| *Msg* | Send simple speech output. |
| *OpenFile* | Open file. |
| *Read* | Read from keyboard. |
| *Pop* | Remove a profile. |
| *Push* | Add a profile. |
| *ReadFile* | Read from file. |
| *Run* | Run OS/2 process, synchronous or asynchronous. |
| *Save* | Save specified components of the state. |
| *Say* | Send data from display to speech output. |
| *Set* | Set a component of the state. |
| *Stop* | Stop current speech output. |
| *View* | Set current view. |
| *WriteFile* | Write to a file. |

Two of the commands above are used to position the Screen Reader/2 pointer in order to be able to speak or otherwise process text, icons or window controls. The *Find* command searches for a string. The *Get* command has many different parameters depending on what is being sought.

```
Get( x, y | cursor | mouse
          | word | nextword | prevword
          | sentence | nextSentence )
```

There are three Screen Reader/2 read-only variables, *row*, *col*, and *RC* (Return Code) which are effected by these positioning commands. If the command is successful, then RC is set to 0; and row and col give the new position. Otherwise, (coordinates or object not in the current view) the return code is non-zero and row and col are unchanged.

The following examples illustrate these commands in conjunction with others listed above.

***Example 5***

```
{5} 'current word'
    Get( Word); Say( Word);
```

The 5 key on the Screen Reader/2 keypad says the current word. The *Say* command uses the state information – format, mode, treatment of capitalization, dictionary, etc.

***Example 6***

```
{#1}   'search forward from top'
Var Search : String[25];
   Msg( 'enter search string');  /* Prompt to the user      */
   Read( Search);                /* Get the string          */
   If Length( Search) = 0 then
      Msg( 'search cancelled' ); /* Cancel search           */
   else
      Msg( 'looking for'); Say( Search);  /* Confirm input  */
      Save( Wrap, Trap);         /* Save state components   */
      Set( Wrap, on);            /* Whole view as one line  */
      Set( Trap, off);           /* No error announce       */
      Set( mode, Pointer);       /* No cursor mode          */
      Get( 1, 1);                /* Go to top               */
      Find( Search );            /* Search                  */
      If RC = 0 Then             /* Announce results        */
        Msg( 'found ', row, ' ', col);
      Else
        Msg( 'not found');
      EndIf;
  EndIf;
```

The *Read* command opens a dialog and accepts text in an entry field. The string variable, Search, gets resulting text string. The *Save* command saves the specified components of the state, to be restored at the end of the statement sequence, or with an explicit *Restore*.

Components of the Screen Reader/2 state are changed with the *Set* command. *Wrap* (whole view as one line) and *Trap* (no error announcements) are set in Example 6.

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

***Example 7***

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

## 3.4 Functions

Although the view is organized by so-called rows, PAL has a plethora of functions to get at all the relevant GUI-based data, like positions, sizes, handles, window classes, id's, and window text.

Most of these functions have the general form of:

```
view.class [ ( viewhandle) ]
```

*View* is the name of the function, we call *class* a *modifier*, and the function is applied to the current view if there is no parameter, or a specific view (think window) if the parameter is supplied.

There are many standard numeric and string handling functions in PAL, just as one would expect in such a language. Below is a list, again brief and not exhaustive, of functions of the Profile Access Language related specifically to the GUI.

| | |
|---|---|
| *Char* | Position, size, font, style, color of a character position. |
| *Display* | Horizontal or vertical string from the view. |
| *Map* | Convert between various coordinate systems. |
| *Mouse* | Position of mouse. |
| *Program* | Current foreground process. |
| *Selector* | Size, position, text, window of current selection object. |
| *Title* | Title of view (window). |
| *View* | Size, position, id, class, text of view (window). |
| *ViewFromRowCol* | ViewHandle given row, col position. |

## 4.0 More About Autospeaks

Autospeaks in the DOS-based Screen Reader could only watch specific (constant) regions of the screen and simple numeric expressions. Screen Reader/2 adds events and opens the watching to all expressions, including procedures.

The following is a list of events available for programming access to OS/2 and its GUI.

*Event_MDM_Notify*.Multimedia command notification.
*EventCursorChange*. A cursor being created, deleted, or moved.
*EventDeviceIn*. Input from the synthesizer or Braille device.
*EventExternal*. Another process can generate this event.
*EventIndexIn*. An index marker returned to Screen Reader/2. causes this event.
*EventKeyPressed*. Key pressed on the keyboard.
*EventMouseMove*. The mouse moves.
*EventPointerChange*. The Screen Reader/2 pointer changes.
*EventProcessChange*. Foreground process change.
*EventSelectorChange*. A selector being created, deleted, or moved.
*EventShiftChange*. A shift key being pressed,
*EventScroll*. When window scrolling is detected.
*EventSpinButton*. Whenever a spin button control is used to change a value.
*EventViewChange*. Screen Reader/2 changes the view.

The fact that expressions include procedures, and that autospeaks can watch such expressions is, as I have already said, very important.

Many conditions arise in access to GUI applications, where one must search for some string or window, in order to make an automatic announcement. The procedure can do the searching and the autospeak can watch the result of the procedure.

***Example 8***

```
proc WatchMemory returning string is
var s: string[25];
   s:='';
   if View.class(viewfromrowcol(view.rows, 1)) = 'PMDCalc' then
       Get( 5,1);
       save( trap, wrap);
       Set( Trap, off); Set( Wrap, on);
       Find( 'Memory');
       Get( row+1, 1); get( nextWord);
       s := display ( row, col, view.cols);
       restore;
    Endif;
    return s;

/* watch for changed memory contents */
autospeak memory_watch watch WatchMemory do
  if trigger <> '' then out( 'memory '); saynum( trigger ); endif;
```

In this example you can't just watch a fixed position because the position changes during execution of the program, or between different machines. In general, the configuration of the graphical display is much more flexible that the twenty-five by eighty text-mode display. As a consequence, it is often the case that procedures must be written to analyze the current state of the view, or even of the whole display, and to take action based on the result of that analysis.

---

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

## 5.0 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.

We really don't know how or whether the Profile Access Language will be used for for main-stream applications, adding both speech and non-speech sound to the interface. It is the thesis of this paper that that is a viable direction, and we have tried to present the language in a way that would stimulate interest in its use.

## 6.0 Acknowledgement

Screen Reader/2 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/2 effort.

## Bibliography

**[1]** Adams, F.R., Crepy, H, Jameson, D.H. and Thatcher, J.W., “IBM Products for Persons with Disabilities,” IEEE GlobeCom '89. November, 1989. (1990) 496-502.

**[2]** Adams, F.R. and McTyre, J.H., “Screen Reader: An Audio Access System,” ICAART Conference Proceedings, Montreal June, 1988.

**[3]** Boyd, L.H., Boyd, W.L., and Vanderheiden, G.C., “The Graphical User Interface: Crisis, Danger and Opportunity,” **Journal of Visual Impairment and Blindness 84** (1990) 496-502.

**[4]** Emerson, M.R., Jameson, D.H., Pike, G.G., Schwerdtfeger, R.S., and Thatcher, J.W., “Screen Reader/PM - GUI Access by blind computer users,” Proceedings, World Congress on Technology, December, 1991.

**[5]** “IBM Independence Series Screen Reader, Reference,” The IBM Corporation, Amonk, New York, 1989.

**[6]** Jameson, D.H., “PC SAID: Computer Access for the Vsually Disabled,” M.Sc. Thesis, Trinity College, Dublin, Ireland, 1986.

**[7]** Mynatt, E.D., and Weber, G., “NonVisual Presentation of Graphical User Interfaces: Contrasting Two Approaches,” *Proceedings*, CHI ‘94., April, 1994.

**[8]** Schwerdtfeger, R.S., “Making the GUI Talk,” **BYTE**, 118-128 (1991).

**[9]** Thatcher, J.W., “Screen Reader/2 - Programmed Access to the GUI,” Submitted, International Conference on Computers for Handicapped Persons, November, 1994.

**[10]** Thatcher, J.W., “Screen Reader/2 - Access to OS/2 and the Graphical User Interface” Submitted, ASSETS '94 (Sponsonsered by SIGCAPH - Computers And the Phiysically Handicapped), October, 1994.