RC 19823 (04/19/94)
Computer Science/Mathematics

IBM Research Report

Screen Reader/2 - Programmed Access to GUI

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

Screen Reader/2 - Programmed Access to the GUI

Jim Thatcher

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

Abstract: In this paper I will describe the Screen Reader/2 personality, that is the 'look and feel' of Screen Reader/2. As I do that I hope to give the reader an introduction to the Profile Access Language (PAL), which is one of the most important components of Screen Reader/2. This PAL introduction will not be formal, in the sense of careful syntax and semantics; I want to get across the idea of PAL so that one might want to program in the Profile Access Language to access the Graphical User Interface (GUI) of OS/2 and of Windows applications running under OS/2.

1. Introduction

IBM Screen Reader/2 is the only access system that gives blind computer users access to OS/2, DOS and Windows applications (running seamlessly). It is the only screen reader to have a fully programmable interface. Screen Reader/2 has grown over nine years in response to the demands of users in IBM and outside who needed the function for their work, education and home environments.

2. Basic Screen Reader/2

Screen Reader/2 for OS/2 is a follow-on to the DOS based product that was released in January of 1988 and upgraded two times since. Two things about IBM Screen Reader distinguish it from all other screen access software; the separate 18 key keypad and its total programmability. On the latter count, the power of the programming language, PAL, is a significant distinguishing feature.

2.1 The Screen Reader/2 Package

Screen Reader/2 for OS/2 consists of an 18 key keypad, a keypad cable, some audio cassettes, some hard copy documents, and two diskettes with on line documentation and software. The keypad is attached through the mouse port. If such is not available, there is an adapter card for the keypad that can be used instead.
The software can be grouped into roughly five sections.

1. Screen Reader executables and dynamic link libraries.
2. Configuration files and utilities.
3. Installation utilities.
4. The PAL compiler and profiles.

5. On line documentation.

The keypad is used as an input device to Screen Reader/2 to request information from the display or to change settings in the speech environment. There are literally hundreds of states that effect how Screen Reader/2 responds.

Screen Reader/2 responds or speaks as the result of a user request through the keypad or because of some *autospeak*. In both cases the nature of the response is determined by the current Screen Reader/2 state and the PAL profile that is active. The important point here is that everything that Screen Reader/2 does is a consequence of some profile code written in the Profile Access Language.

2.2 The Role of Profiles

Profiles both define the basic screen reading environment, the Screen Reader/2 personality, and they are also used as add-ons to tailor that basic environment to specific applications when that is desirable or even necessary.

This aspect of Screen Reader is often misunderstood. The application specific profiles are developed for making applications easier to use. Everything that is done in application specific profiles could be done without them (probably), but with more effort (probably a lot more effort).

The simplest example of the desirability of an application specific profile is when the application has a status area in which, say, error messages are displayed. Using the keypad and standard core requests, the Screen Reader/2 user can read that area to find out about any messages without a special profile. With an application specific profile, however, that area can be monitored and automatically read. In addition, a simple application specific key sequence could be defined to read the status area.

There are about 30 applications specifically supported by the Screen Reader/2 product. These profiles are supplied for OS/2, DOS, and Windows applications.

Whether considering application profiles or the base Screen Reader profile, it is crucial also to understand that Screen Reader's behavior is completely determined by the current active profile or profiles.

2.3 Sessions

There is one base or core profile, called srd2.kpd. This profile textually includes the sub-profiles for the edit facility, mouse movement, icons, scanning, etc. These are all conceptually and syntactically part of the base profile.

The way this base profile works is that it is loaded and active when Screen Reader/2 is started. This base profile and any environment settings that might be determined by it, define what we call *default* Screen Reader/2 *session.*

Whenever a new profile is added to (we say *pushed on*) the base profile, i.e., the default session, that action creates a **new** Screen Reader session associated with the foreground process. You can save settings in that session, you can push another profiles for that session. When the process is killed the session goes away.

To illustrate this session concept, just imagine that there are two sessions, the default session and a new session created by pushing a profile for, say, the System Editor. Then you could be in *spell format* (where everything is spelled) in the System Editor session, and *text format* (normal reading) in the default session. Then any time you switched from the default session to the system editor session, you would switch from text to spell format and conversely.

2.4 The Nature of Profiles

A simple example will help to illustrate the use of PAL and the idea that any Screen Reader/2 response can occur as a result of a keypad entry or some event that triggers an autospeak.

Example 1

```
include 'srd2.inc'
Var TimeWatch: AutoHandle;

{1} Msg ( 'it is ', DateTime( DT_HOUR ), ' o clock' );

AutoSpeak TimeWatch
    Watch DateTime(  DT_HOUR ) Do
    Msg ( 'it is ', trigger , ' o clock' );
```

The file srd2.inc has definitions of constants like DT_HOUR. The third line of PAL code above gives meaning to the 1 key, {1}, of the screen Reader Keypad. When this fragment is active and the 1 key is depressed, the message 'it is 9 o clock' is spoken. This line of PAL Code is executed when the 1 key is pressed.
On the other hand, the autospeak with handle TimeWatch, watches the value returned by the function DateTime(DT_HOUR) which, the reader might have guessed, is the current hour. When that changes, then the body of the autospeak is executed. That body is also a message statement (Msg sends its arguments to the attached serial device). The value of the reserved variable, *trigger*, is the value that triggered the autospeak, in this case the current hour. So at the moment the hour changes to 9, the message 'it is 9 o clock' is announced, just like pressing key 1.

2.5 Basic Review Requests

Screen Reader has many standard review requests programmed into its base profile (os2core.kpd, included in srd2.kpd mentioned above). Each of these requests is tempered by the *Mode* and *Format* setting.
There are two reading 'modes,' called *cursor* and *pointer*. In cursor mode all reading requests are relative to the application cursor; in pointer mode all requests are relative to Screen Reader/2's own 'pointer' (called the review cursor by some screen reading packages). Because other screen readers do it differently, it is important to emphasize that there are many reading requests and all of them work in either pointer or cursor mode. The request is the same sequence on the Screen Reader keypad, like {2} for current line. Depending on the current mode, that will be the cursor line, or the line containing the Screen Reader/2 pointer.

There are five reading formats called *text*, *pronounce*, *spell*, *phonetic*, and *ASCII*. In text format Screen Reader tries to read as if reading a book. In pronounce format, punctuation is also announced. In spell format, all words are spelled and in phonetic format everything is spelled using the international phonetic alphabet. In ASCII format you hear the ASCII value of each character.

Again, I want to emphasize that all the basic requests are available and the result of each depends on the current format setting. There are shortcuts, however. For example, it is unlikely you really want ASCII format, so a key sequence is defined to give the ASCII value of the current character, and none to switch to ASCII format. Similarly, another sequence is defined to spell the current word with out changing the active format.

Here is a list of some basic review requests.

- Read the entire window or read from current position to the end.
- Read the current, previous, next sentence, line, word, or character.
- Move pointer to cursor, to top left, to bottom left, or to right edge.
- Spell the current, previous or next word.
- ASCII value of current character.
- Current character in Phonetic alphabet.
- Search for a string, search again, or search from bottom.

This list could be expanded, but such detail becomes tedious. There is a separate, though related, list which is important in portraying the Screen Reader/2 personality. As I mentioned above there are many components of the Screen Reader/2 state that can be set by the user, and in fact, can be different in different Screen Reader/2 sessions. The ways these state components are set is not so important as is the following illustrative (certainly not exhaustive) list.

- Mode cursor or pointer.
- Format: text, pronounce, spell or phonetic.
- Ignore capitalization or not.
- Announce spaces or not.
- Treat entire screen as single line (wrap) or not.
- Announce line numbers while reading or not.
- Use the dictionary or not.

The following are GUI specific.

- Hear drawing noise or not.
- Include icons or not.

Having just listed a sampling of Screen Reader/2 core requests, now I want to give a sample how some of these appear in PAL. Remember that we are not detailing the syntax and semantics of PAL. I believe that the examples will, for the most part, be self explanatory. The following examples are exactly as they appear in the profile, os2core.kpd, shipped with Screen Reader/2.

Example 2

```
{0A} 'read entire screen'
     Save( Wrap); Set( Wrap, on);
     Get(1, 1); Say(line);


{1} 'previous line'
    Get( row-1, 1); Say( Line);


{6} 'next word'
    Get( Nextword); Say( Word);


{*C} 'Ascii value of current character'
     Save( Format); Set( Format, Ascii); Say( Char);
```

A key definition in PAL is a key or key sequence in braces, followed by an optional comment used for help. After that is a sequence of PAL commands.

Only four Screen Reader/2 commands are used in the key definitions in example above. The *Save* command saves one or more components of the Screen Reader/2 state. Any saved component is restored automatically at the end of the command sequence.

The *Set* command sets a component of the state. In the statement sequence for {0A}, *Wrap* is set to *on* which means the entire view is treated as one line. The format is set to ASCII in the command sequence for {*C}.

The *Get* command positions a local pointer to a position specified by its arguments. When a command sequence is started, the local pointer is assigned the cursor position if in cursor mode, and the pointer position if in pointer mode. The local pointer's coordinates are always given in two read-only variables, *row* and *col*. The local pointer gets updated throughout a command sequence and is assigned to the Screen Reader/2 pointer at the end of the sequence if in pointer mode.

The *Say* command is used to send text from the display to the synthesizer (or other device). Depending on its argument, Say works on a line (or whole screen if wrap is on), word or character.

2.6 Editing

The Screen Reader/2 edit facility provides for feedback as the user moves the cursor and types. The edit facility is in fact another profile, and a rather complex profile at that. I don't want to explore the design of that profile here. (Neither do I want to discuss the question of whether or not the edit facility functions should or should not be built in at a lower level in Screen Reader/2.)

Instead I will mention the 5 major feedback options in the edit facility.

- **Line Browse**. With Line browse, every time the cursor moves one position to a new character, that character is spoken; when the cursor moves horizontally a word at a time, the word is spoken, and when the cursor changes row or the line changes as in a scroll, the new line is spoken.
- **Flush**. With flush turned on, each new edit facility response cuts of (stops) the previous response,

The Default edit settings are line browse and flush. In addition there are three 'echo' settings.

- **Character echo**. Echo each character as typed.
- **Word echo**. Echo changed words as the space is typed at the end of the word.
- **Line echo**. Echo changed lines when move to new line.

There other options in the edit facility that can make life easier under some circumstances, including a margin indicator and a column browser.

3. Navigating around the Graphical User Interface

Almost all OS/2 and Windows applications respond to keyboard input as a substitute for mouse input. As a simple example, the window list (the list of running programs) is displayed using the right mouse button on the desktop, or using the keyboard combination Ctrl+Esc. One can highlight an item in a list by using the mouse and single click, or by moving to the item with the arrow keys. The item is invoked (or opened) using a double click with the mouse or by highlighting the item and pressing enter.
In general, a blind user navigates around the GUI with the keyboard. During that navigation, Screen Reader/2 provides constant feedback.

3.1 The Concept of View

Under the default settings, Screen Reader/2 always restricts reading to what it determines to be the main window and all text and icons contained in that main window. Because the word 'window' is over used, we call that rectangle the Screen Reader/2 *view*. As the user switches between applications, or between main windows, Screen Reader/2 automatically adjusts the view to the current main window.
Besides adjusting the view; Screen Reader/2 automatically announces the window title when the main window or application changes.

Once the view has been established, the view is treated very much as if it were a text window. All the text in that view is sorted by baseline into a certain number (*view.rows*) of distinct rows and each row is assumed to have the same number (*view.cols*) of 'columns' namely the maximum character length of the rows.

We believe that this simplifies access to the graphical view, though it is not always an accurate reflection of what is displayed. Our experience has been, with many users of Screen Reader/2, that this filtering of the displayed data is a help for review and that no significant information is lost.

The characters in the first 'column' cannot be expected to be seen to be lined up vertically as they are in a text screen. There is no implication about vertical positioning even of different rows. One 'row' might be huge and take up most of the view, with the remaining rows crammed in a small font at the bottom of the view, like, say, footnotes. One block of text could have a baseline just above a block to its right. These would be considered distinct rows.

Two blocks of text could be the same font, size and style and have the *same* baseline and still be on different 'rows.' This happens when the text occurs in separate windows. And the reason we put text in different windows into distinct rows is because, almost certainly, one wants to be read the text of those windows separately.

All the information that is ignored by this row-column access of the view is available to the user. This includes:

- The pixel position of each character and its size (*char.left*, *char.bottom*, *char.right*, *char.top*).
- The font, pitch, style and color of each character. (*char.font*, *char.pitch*, *char.style*, *char.color*).
- The pixel size of the view and its position on the desktop. (*view.top*, *view.left*, *view.bottom*, *view.right*).
- The identity of the window containing the text, the *window handle*

By calculations in PAL (we have not found reason to use these in basic profiles).

- Whether one row or column truly lines up below another.
- The amount of white space between rows.

- The amount of white space at the beginning and end of rows.

The reason the view is set as it is by Screen Reader/2 is that conventional review requests would be utterly confusing without it. Successive lines or blocks of text might come from different main windows and make no sense at all. As with most function and default operation of Screen Reader/2, the user can have it both ways. With standard requests, the user can make the whole display (or *desktop*) the view, or only the contents of a single push button.

Views are determined by the windows. A view always includes all child windows contained within a specified window. Thus the view determined by the desktop window includes everything displayed; the view determined by a push button is just the button text or icon. Pushbuttons are windows in the technical sense (except in the case of some Windows applications written by MicroSoft).

Views are not the only way of restricting reading. Viewports are rectangular regions within a view, to which reading can be constrained.

3.2 Using Controls within a View

A *dialog* is one kind of main window that does not fit well into the row and column format described above. Dialogs typically consist of several windows (in the technical sense, again) which allow the user to enter data and make choices. The individual windows (buttons, entry fields, static text fields, check boxes) are called **controls**. The user makes choices with the buttons, entry fields and check boxes; the prompts or headings are provided with by the 'static text.'

A sample will help. In the OS/2 Enhanced editor, a search dialog appears when you want to search for and possibly replace text in a file you are editing. At the top of the dialog are two entry fields. The first has the prompt (or name) 'search' and the second has the prompt 'replace.' You type in the string you want to search for in the first entry field and press enter.

These two entry fields and their prompts actually comprise four windows in this view, and four rows as well. As you move to them, however, with the standard tab-key movement, you hear 'search entry blank' and 'replace entry blank' respectively. When the data is put in the entry field, then the editing functions of Screen Reader/2 can be used.

Because of the feedback from Screen Reader/2 to the user, there is little interest in what rows and columns are involved, or the fact that the sighted user might say that the prompt and the entry field data appear to be on the same 'row.'

Continuing with this example, below the entry fields is a box (looks like a window and is called a group box) with a heading 'Options.' That box contains five check box controls which consist of small squares (about one-quarter inch) which may or may not have check marks in them. Each has text immediately to the right of the small square. These are used to select options such as 'ignore case,' or 'change *all* occurrences.'

As the user tabs into this area the and arrows around the check boxes, Screen Reader/2 responds with

'Group box options check box ignore case checked'

or

'Group box options check box change all occurrences not checked'

It is standard in the GUI to toggle the check state of a check box with the space bar. Hitting the space bar when on the ignore case check box, results in the announcement

'Group box options check box ignore case not checked'

from Screen Reader/2.

Finally, at the bottom of this dialog is a group of six *push buttons*, things like 'Find,' 'Change,' and 'Cancel.' When the user tabs to this group of buttons, and arrows around there, the information, is presented in a form similar to that for check boxes:
'push button find'
'push button cancel'

3.3 Mouse Actions and other Tricks in Screen Reader/2.

There is no doubt that not using a mouse with the graphical user interface is a disadvantage. It is quicker to do some things with a mouse than with the keyboard equivalents (when those keyboard equivalents are known). The worst part of this is probably the fact that help panels are written with the mouse user in mind. They often do not include the the keyboard alternatives.
Screen Reader/2 has done some things to make this situation better. One of the most important features is our *switch list*.

A mouse user brings a window to the foreground by moving the mouse to any part of that window and clicking with the left mouse button. If that window is totally obscured, either other windows must be moved, resized, minimized, or closed; or the mouse user can use the keyboard equivalent if they know what it is.

From the keyboard, Ctrl+Esc brings up the list of open programs. Then the Screen Reader/2 user can scan that list one at a time, or pres the first letter of the title, until the item is found and then open the folder or start the program by pressing Enter. This is a significantly longer activity than moving the mouse and clicking.

The Screen Reader/2 user can identify up to 15 windows (in the default case) and when they want to switch to one of their favorite windows (whether obscured or not) they just enter the number of the window preceded by the simple chord 89 on the keypad. Movement between a collection of standard windows becomes easier than the same task for the (sighted) mouse user.

The OS/2 2.1 desktop covers the entire display. On it are icons for programs and program folders and other objects like printers. To open (start) one of these objects the mouse user must find it, and double click. Other windows that are open are very likely to obscure one or more of these icons, and the only option here for the mouse user is to close, move, resize, or minimize some of the open windows. This is often a problem for the sighted user, but it would be a disaster for the blind user.

Screen Reader/2 solves this problem with an alternative desktop which is a normal folder object and can be brought to the foreground just like other folders. In particular the Screen Reader/2 switch list can be used, to make the desktop the current view.

As the last subject in this section, mouse actions (left and right button, click and double click) can be preformed with Screen Reader/2 at any time.

Think of the Search Dialog for the Enhanced Editor described in the previous section. As we discussed there, the normal use of that dialog is to tab or arrow key to the required spot, enter data, or change options with the space bar, and then press enter to carry out the action. An alternative is available for any application.

Using a couple of basic Screen Reader/2 requests from the keypad, the blind user can perform a mouse single click (chord 23 then 4) or double click (chord 23 then 5). So when the Screen Reader/2 user reviews the contents of dialog and is reading, for example, the 'ignore case" checkbox, a single click (using the keypad) will change the check state of the check box from checked to non-checked or conversely.

4. Some Technical Constructs in PAL

In this last section I will describe a couple of the unique and powerful features of the Profile Access Language. Continuing in the spirit set in the first section, I will not stress the detailed syntax of PAL, but instead illustrate the ideas with PAL code and descriptive text.

4.1 Autospeaks and Procedures

In the first section, ***Example 1*** showed how an autospeak could be defined to announce the time when the hour changed. The watched expression in that case was a function (DateTime) which returns the current hour.
This idea, watching a function or procedure, is key to a very powerful concept for Screen Reader/2; but some history first.

Ten years ago, when Screen Reader was a research project called ***PC SAID***, we did have programmable autospeaks, but only to watch specified screen positions, and somewhat later, numeric expressions. In its early days, this was far ahead of anything else that was available or planned in the screen access area.

Lets review how the autospeak construct works. The syntax (roughly) is:

AutoSpeak Syntax

```
AutoSpeak <autohandle>
    watch <expression>
       do <Command Sequence>
```

The *autohandle* is a variable needed to query the state of the autospeak, and to turn it on or off. It is not important for understanding this discussion. The expression is just that, any expression in PAL. So you can watch a part of the display as in the original PC SAID autospeaks by watching, for example,

```
Display( 23, 67, 4)
```

to watch row 23, column 67 for 4 characters.

Then *Command Sequence* is any PAL sequence of commands.

The way this works is that the expression is evaluated periodically, say every tenth of a second. If that value changes (and stays changed for some specified time -- a detail that won't bother us) then the autospeak body (the Command Sequence) is executed. As indicated in the time example in Section 2.4. there is a convenient system variable, *trigger*, which contains the value of the expression that triggered the autospeak.

All of this is pretty much the same as has been available since the days of PC SAID; the syntax has been cleaned up and there are some conveniences. But what is truly new here is the fact that PAL for Screen Reader/2 has user-defined procedures that return values, and, of course, these can be parts of expressions. This is a major functional change, significantly increasing the power of the autospeak feature.

Why is this powerful? Because *in principle* it makes the autospeak construct universal, that is, one could, *in principle*, design and code an autospeak to announce whatever was desired based on the information that Screen Reader/2 knows to be on the display.

There are dozens of examples of procedures which are watched by autospeaks in the profiles shipped with Screen Reader/2. These are generally used to search for something, either in the given view, looking for some special indication, or sometimes in another view.

There are important simplifications implied by watched procedures also. It is good practice in profile development to write a procedure to obtain information from the display (from the view) and to use that procedure in a key sequence (for review) as well.

A prototypical use of this idea is checking for a highlight bar in a text based application running under the OS/2. This is a requirement for working with some DOS applications running under OS/2, but generally not for the OS/2 GUI applications.

Think first of a single row (say row 3) across the top of the display, with menu items on that row which are highlighted by changing the background color when accessed with F10 (usually) followed by arrow keys.

In this case the procedure (called *action*) to be watched checks row three and returns the column of the highlight if there is one and returns zero otherwise.

Example 3

```
proc action returning integer is
var place : integer;
   get ( 3, 1 );
   Get( diffattr,
        (char.color bitand &hff00),
        &hff00, +);
   if rc = 0 then place := col;
   else place := 0; endif;
   return place;
```

The key command in the procedure above is the Get command for finding colors (attributes). This is, in fact the most complex command in PAL. The first argument is a reserved word; you want to find the same or different attributes (*sameattr*, or *diffattr*, respectively). The second is argument is the attribute to be compared using the third argument as a mask. The forth argument tells whether to go forward to backward.

In our case we are to find the first occurrence of a color which is different from the current attribute (*char.color*) in the second byte (the background), and go in the forward (+) direction.

The return code (*RC*) will be set to zero if the search was successful, and non zero otherwise. Therefore the procedure returns the column where the different background was found it it was found, and zero otherwise.

Now if we watch this procedure with an autospeak it looks like this.

Example 4

```
var a: autohandle;

autospeak a watch action do
   if trigger <> 0 then
      get( 3, trigger); say( field);
   endif;
```

This autospeak results in the procedure *action* being evaluated periodically, and if action finds a highlighted item, the return value is non-zero and the body of the autospeak will announce the highlighted item because *say(filed)* speaks from the current position to the end of field, where a field is a contiguous sequence of character positions with the same attributes (color).

The application on which I am testing these examples uses the the same highlight for lists of elements inside boxes. When an action item is selected (with enter) the action item is no longer highlighted, and some other item is.

With an simple change in the procedure, *action*, all these highlight items can be announced, For this the *wrap* switch needs to be used (see ***Example 3***) to search the whole view for a highlight. Then only minor modifications are needed in the autospeak as well. The following presents the completed example.

Example 4

```
var r, c: integer;

proc action returning integer is
var place : integer;
   Save ( wrap ); set ( wrap, on );
   get ( 3, 1 );
   Get ( diffattr,
       ( char.color bitand &hff00),
       &hff00, +);
   if rc = 0
      Then c := col; r:=row; place := row*25+col;
      else place := 0; endif;
   return place;

var a: autohandle;

autospeak a watch action do
   if trigger <> 0 then
      get ( r, c ); say ( field );
   endif;
```

4.2 Autospeaks and Events

As I indicated in the preceeding section, the expressions that autospeaks can watch provide a great deal of power over what was available for Screen Reader/DOS. There is another addition for ScreenReader/2. Autospeaks can watch *events*. Unlike expressions, which must be evaluated periodically, events are generated by Screen Reader/2, or even another process, and those events trigger the execution of the body of the autospeak.
Here is an extremely simple example of an autospeak on EventCursorChange that does a lot of what the edit facility (Section 2.6) does. It certainly does not do all that the edit facility does.

Example 5

```
var Edit : autohandle;
var r, c : integer;

autospeak Edit
    watch EventCursorChange Do
    if r <> crow then  -- row changed
       get ( crow, 1 ); Say ( line );
    elseif if abs( c - ccol) = 1 then  -- one col change
       get ( crow, ccol );  Say ( char );
    else Say ( word );
    endif;
    r := crow; c := ccol;
```

The idea of this edit autospeak is to say the line if the cursor line changes; the character if only one column changes, and the word where the cursor lands if the change is more than one column. The variables r and c are used to save the previous values of the cursor row and column which are contained in the read-only variables, *crow*, and *ccol*, respectively.

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

EventCursorChange. A cursor being created, deleted, or moved.

EventDeviceIn. Input from the synthesizer or brialle device.

EventExternal. Another process can generate this event. Sample illustrative code is included with ScreenReader/2.

EventIndexIn. An index marker returned to Screen Reader/2 causes this event.

EventKeyPressed. Key pressed on the keyboard.

EventMouseMove. This event occurs when the mouse moves.

EventPointerChange. Whenever the Screen Reader/2 pointer changes, this event occurs.

EventProcessChange. Whenever the current process changes, this event occurs.

EventSelectorChange. A selector being created, deleted, or moved causes this event.

EventShiftChange. A shift key being pressed causes this event.

EventScroll. When window scrolling is detected.

EventSpinButton. Whenever a spin button control is used to change a value.

EventViewChange. This event occurs whenever Screen Reader/2 changes the view.

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., "How to make OS/2 Talk and Why. Access to OS/2 with Screen Reader/2," To Appear, Proceedings OS/2 World Conference, July, 1994, Santa Clara, CA.

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.