

---

---

# **JAWS For Windows MACRO Training Manual**

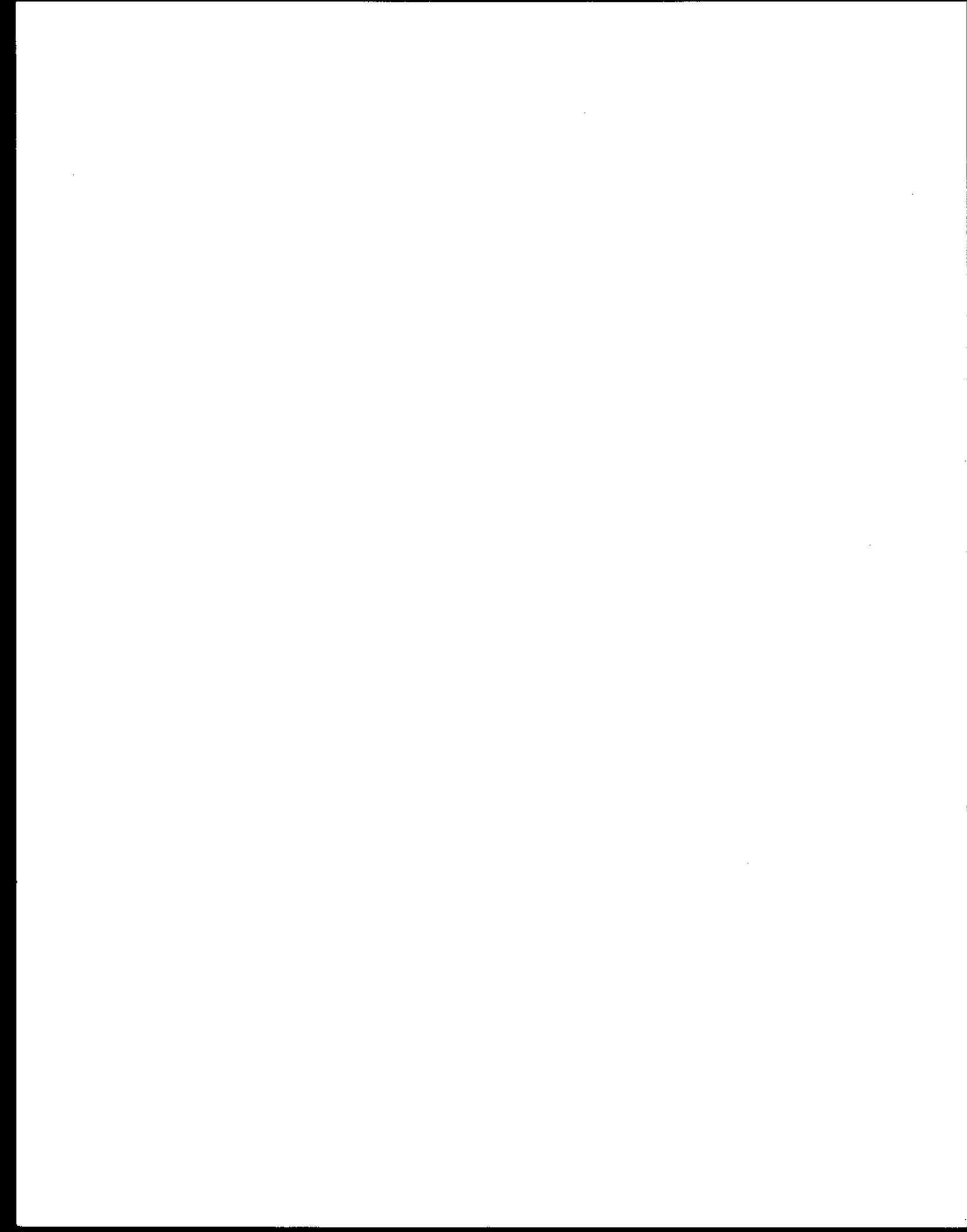
---

---

**High Tech Center Training Unit**

**Of the California Community Colleges at the  
Foothill-De Anza Community College District**

**21050 McClellan Road  
Cupertino, CA 95014  
(408) 996-4636**



---

---

# Table of Contents

---

---

<b>What is a macro? .....</b>	<b>3</b>
<b>Structure of a JAWS macro .....</b>	<b>5</b>
<b>Types of macro files .....</b>	<b>7</b>
<b>Developing new macros .....</b>	<b>8</b>
<b>Calculator Macro (calc.jms) .....</b>	<b>9</b>
<b>Practice Exercise: Phone Dialer Macro (dialer.jms) .....</b>	<b>18</b>
<b>Glossary: Jaws for Windows Macro Functions .....</b>	<b>19</b>



## WHAT IS A MACRO?

---

Before talking about how to use the Macro Editor and macros, it is important to understand the term "macro". A Macro is simply a sequence of actions that are grouped together into a single operation. Macros are created to save time by making computer use more efficient. Macros automate tasks that you can do manually. For example, a macro can reduce thirty key strokes into one key stroke. A macro can enter key strokes, activate functions, and even make simple decisions.

The concept of a macro can be demonstrated by looking at a common human activity. The human "tooth brush" macro is performed every morning by millions of people and contains concepts that are quite similar to those used in computer macros. The toothbrush macro begins as you start to look for your tooth brush, and then continues through the following steps.

- a. Pick up toothbrush.
- b. Pick up toothpaste.
- c. Remove the cap from the toothpaste tube.
- d. Squeeze toothpaste tube and deposit toothpaste on toothbrush.
- e. Replace the cap on the toothpaste tube and set the tube down.
- f. Brush teeth, as instructed by your dentist.
- g. After completing the brushing sequence, rinse mouth and toothbrush with water.
- h. Place toothbrush back in its proper location.

You can think of each of the above steps as a set of individual activities, or you can consider them to be one activity, i.e., brushing your teeth. If you put the individual steps together and give them a name, then you are doing the same thing that is done when you create a macro. The toothbrush macro is also like a computer macro in one additional way. Most of the time, when you perform the toothbrush macro, you do so without thinking about the steps. Similarly, when you use a computer macro you don't have to give much thought to the steps that are being performed, you just need to think about what the macro can do for you.

A macro can be divided into two parts: the "macro name" and the "macro routine". The Macro Name can either be a combination of keys such as (ALT-I-UP ARROW) or a descriptive phrase such as "TextEvent".



A macro name can be thought of as a label that is used to identify the macro. The Macro Routine is the various key strokes and actions that are to be performed by the macro. The macro routine is the work that is done by the macro. We may, from time to time, simply use the term "macro" in a general way to refer to both the macro name and its routine.

Almost any key on your keyboard or combination of keys can be used for a macro key. The letter keys, number keys, punctuation keys, cursor keys, and function keys can be used as macro keys. You also can use these same keys in combination with the {CONTROL}, {SHIFT}, {ALT}, and {INSERT} to create multiple macros for the same keys of the keyboard. Additionally, You can combine {SHIFT}, {ALT}, {CONTROL}, and {INSERT} with one another. For example, you can **use {ALT+INSERT+LEFT ARROW}** as a macro key.

Many of the keys that make JAWS speak as you use Windows are either macros that are connected to specific combinations of keys, or are macros that are automatically activated in response to certain events taking place on the desktop. This means the keys used to perform the functions of the standard keyboard layout can be changed. It is even possible to create a completely different keyboard layout to meet your specific needs.



## **STRUCTURE OF A JAWS MACRO**

**When you create or edit a macro, you are working with text. The JAWS Macro Editor converts the text into binary computer files when you save your macro file. The following example shows the proper Structure of a JAWS Macro. The text is displayed in the same way as it would appear in the JAWS Macro Editor. Please note; text in a macro can be either uppercase or lowercase.**

```
# This macro is used when the PAGE DOWN key is pressed
MacroBegin
  (Page Down)
  SayString("page down")
  aAWSPageDown()
MacroEnd
```

**Macros make sense when they are broken down into their component parts. To understand this macro, you need to understand the purpose and function of each part of the macro. Each line of the macro is described below.**

Line 1:

```
# This is for the PAGE DOWN key of the speech pad
  Any line in a macro file that begins with a "#" (number sign) or
  a ";" (semicolon) is a Remark Line. It contains information that
  helps to explain the purpose or use of the macro and is provided
  as a convenience to those who are editing the macro file. The
  remark line at the beginning of this macro explains the purpose
  of this macro. These optional remark statements can also be
  placed at the end of a line of text. All text that appears after
  a "#" or ";" is ignored when the macro is used. Regardless of
  whether a remark line is used, a blank line should always be
  used to separate macros from one another. Thus, a blank line
  should precede this first line of the macro.
```

Line 2:

```
MacroBegin
  This line indicates the actual Beginning Point of the Macro.
  It is required in every macro.
```

Line 3:

```
(Page Down)
  This line identifies the Macro Name. The name of the macro
  must be surrounded by braces "{ }" as is shown here. A macro
  name such as (INSERT+CONTROL+PAGE DOWN) could also be
  used. This macro name means that you must hold down both the
  (INSERT) and the (CONTROL) before pressing the (PAGE
  DOWN). You could also use a macro event as the macro name.
  Event macros are executed automatically when certain conditions
  occur on the desktop. Lines 2 and 3 are often combined on the
  same line. For example, "MacroBegin (PAGE DOWN)". Chapter
  3 in this manual lists macro keys and event macro names that
  can be used in your macros.
```

Line 4:

```
SayString("page down")
  The expression "SayString" is a Macro Function. This specific
```



macro function sends a string of text to the speech synthesizer so it can be spoken. Macro functions must be followed by a pair of parentheses "( )". The parentheses indicate the place where a Parameter is to be specified. In many cases, a parameter is not required, however, the parentheses must always be included after the macro function. In this specific case the text string must also be enclosed by quotation marks since this is text that is to be spoken.

Line 5:

JAWSPageDown()

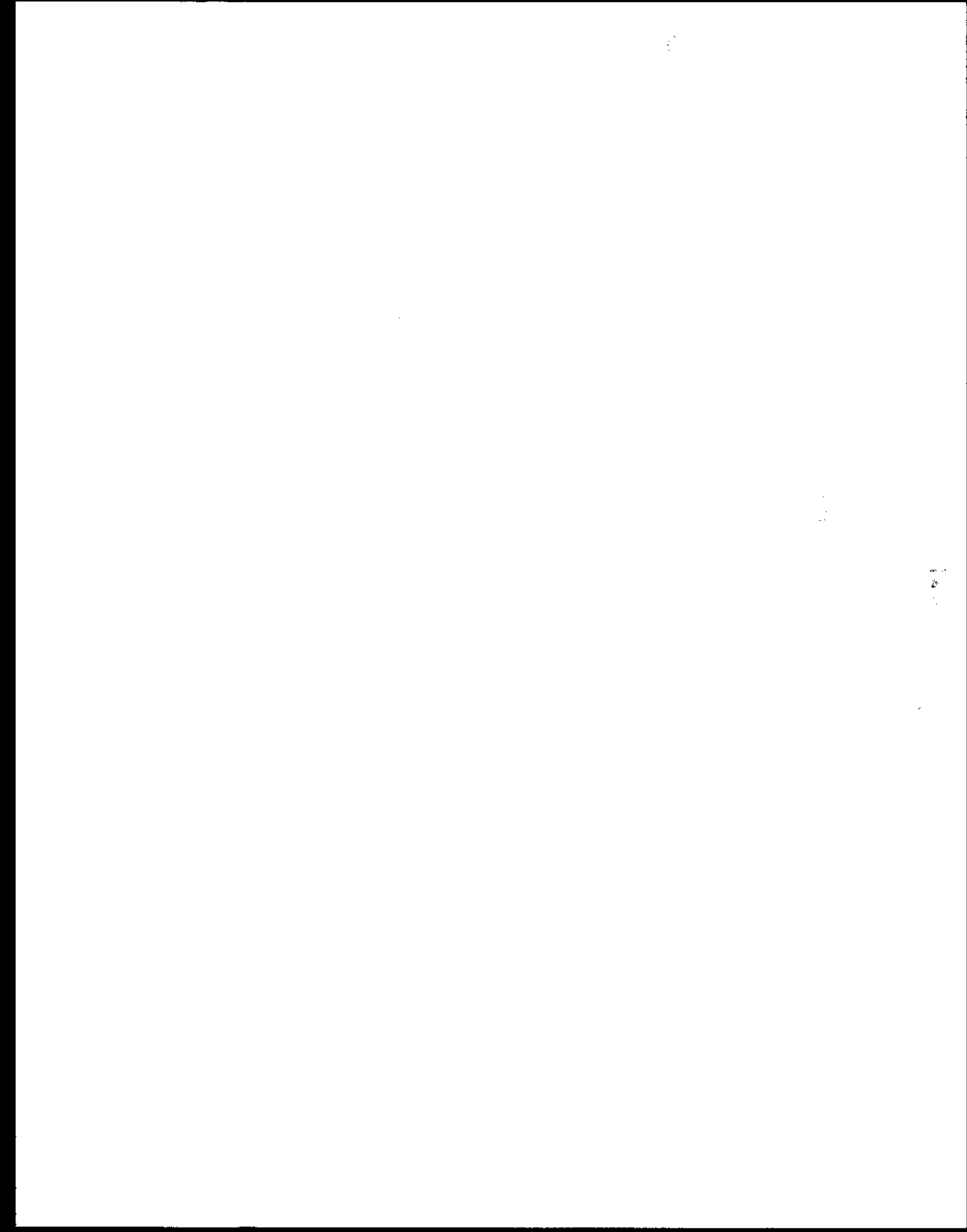
This **JAWS** macro function initiates the page down command. This *is* not a standard (PAGE DOWN) function. It performs different activities depending on the cursor that is in use. When the **JAWS** cursor is active, this function moves the **JAWS** cursor to the bottom of the active window. When the PC cursor is active, it sends a standard page down command to the Windows application.

Line 6:

MacroEnd

This required line indicates the Ending Point of the Macro. This line should be followed by a blank line to separate it from the next macro.

The concept of macros is universal. Many software programs have a facility for creating macros, however, their potential uses are not identical. For example, a **JAWS** macro can read the title of a window or announce the status of a check box in a dialog, but a WordPerfect macro cannot perform these functions. A **JAWS** macro, however, could be used to execute most any function of Microsoft Word, WordPerfect, or most any other application software program. **JAWS** macros actually offer greater versatility than other types of macros, because they can blend screen reading functions with application software **functions** to create powerful tools.



## TYPES OF MACRO FILES

---

When you edit macros, you are working with a specific type of text file called a JAWS Macro Source File. These files have the file name extension of ".JMS". When you use the macro editor to save a file with the extension of ".JMS", it saves the text file and automatically compiles it into the binary file that is use by JAWS. These JAWS Macro Binary files have filename extensions of ".JMB". If you use the macro editor with files that do not have the filename extension of ".JMS", then the files are not compiled when they are saved. You can use any text editor to edit a ".JMS" file, however the macros can only be converted into a ".JMB" file by the JAWS macro editor.

The file "DEFAULT.JMS" is the Default JAWS Macro Source file that is compiled by the macro editor to create "DEFAULT.JMB". The "DEFAULT.JMB" is always in use regardless of the Windows application you are using. Other application ".JMB" files are automatically loaded when you start specific Windows applications.

Application ".JMB" files are used concurrently with the "DEFAULT.JMB" file. When you press a key on your keyboard, JAWS first looks through the application ".JMB" file (if there is one) to find a matching macro name. If the macro name is not present, then JAWS looks in the "DEFAULT.JMB" file. If a macro is not present in the "DEFAULT.JMB" macro file, then JAWS passes the key stroke to the Windows application for processing.



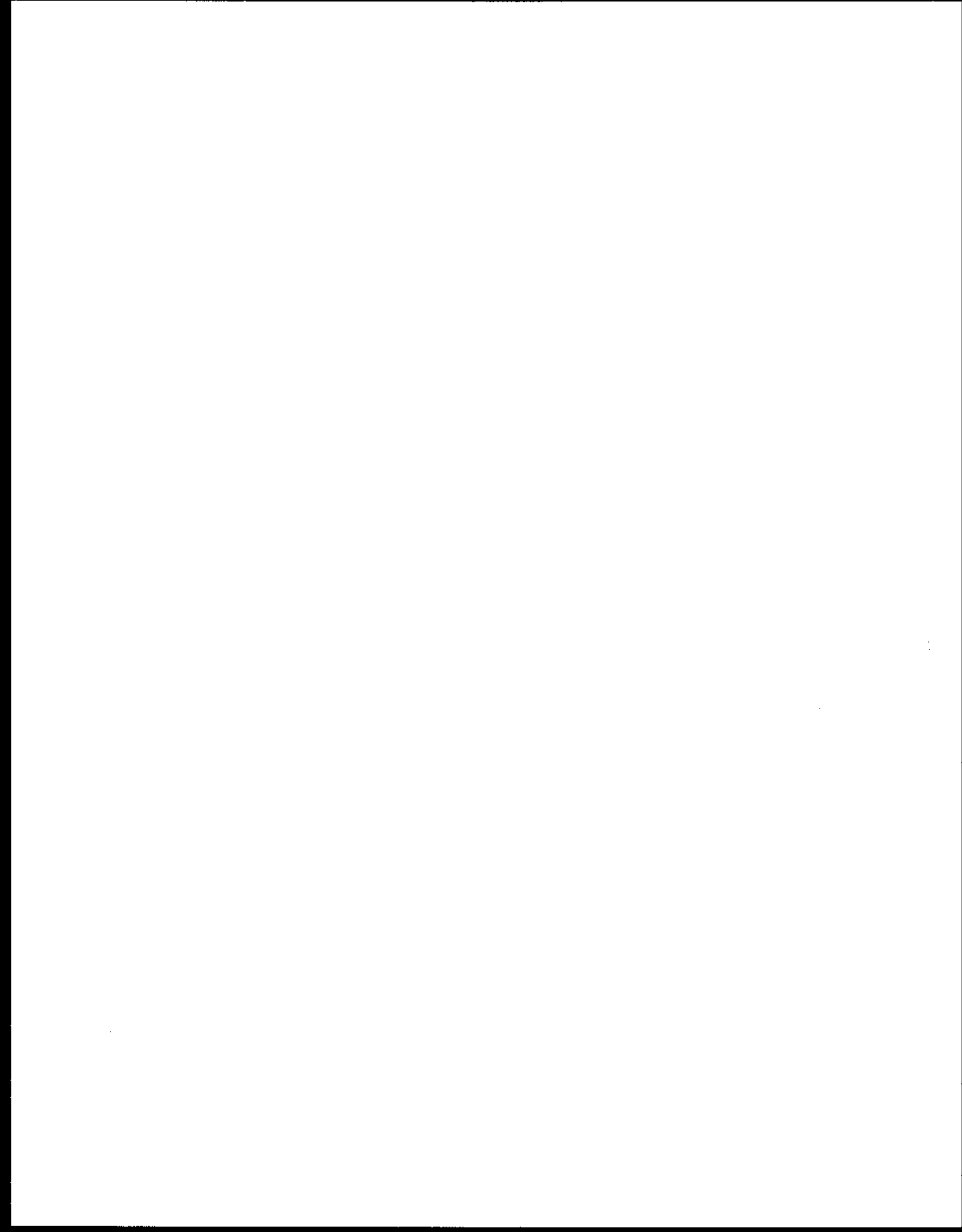
## DEVELOPING NEW MACROS

---

The best way to learn about the techniques that are used in **JAWS** macro programming is to spend some time studying the macros in existing **JAWS** macro source files. The "DEFAULT.JMS" and "DEFINCL.JMS" files located in the \JFW\SETTINGS\ENU subdirectory are a good beginning point for your study. Read the macros and follow the steps they contain from beginning to end. If you do not understand a macro function, then you can look it up in the next chapter or you can review the file "FUNCTIONS.JML" located in \JFW\SETTINGS\ENU. This file contains the most up to date list of macro functions being used in the **JAWS** Macro Language.

Once you understand the purpose of the macro functions and you understand how they can be put together to perform tasks, then the next step is to study your application programs. When you find a situation where your productivity can be enhanced with a new macro, then you need to analyze the activity. You need to breakdown the activity into its individual steps. You need to answer the types of questions presented below in order to create effective macros.

- a. What is the exact task you are trying to complete? Be as specific as possible.
- b. What specific keys do you need to press and in what order must they be pressed to accomplish the specific task?
- c. Do you want **JAWS** to speak? If so, then do you want **JAWS** to *speak* a string of text that *is* built into the macro or to read something from the desktop?
- d. If the macro must make a decision based on the status of your application program, then what information on the desktop can **JAWS** check in order to decide which response is appropriate?
- e. Have you used an application that is similar to the one that you are considering developing a macro for? If the answer is yes, then consider looking at that macro file first before trying to develop a macro from scratch.



## Calculator Macro (calc.jms)

---

---

```
;Copyright 1996 by Henter-Joyce, Inc.
;Windows calculator macros, 3/30/96, th & esd
;please note: this file contains only the standard macros.
;the scientific macros are in the file calc1.jms.
;any changes made to this file will necessitate that calc.jms be recompiled.
```

```
Include "hjconst.jmh"
Include "calc1.jmh"
Include "calc2.jmh"
```

```
MacroBegin {AutoStart}
    if GetVerbosity()    0
        then
            SayString("Press Insert plus H for Jaws Hot Keys with standard and")
            SayString("scientific calculators")
        Endif
    MacroEnd
```

```
MacroBegin {Insert+Q}
    If(GetVerbosity() =-- 0)
        then
            SayString ("The macro file currently being used is ")
        Endl f
        SayString ("Calculator macro file")
    If(GetVerbosity()    0)
        then
            SayString ("The application currently being used is the ")
        Endif
        SayString (GetAppFileName 0)
        SpellString (GetAppFileName 0)
    Macroend
```

```
MacroBegin { ' }
;this macro gets the value in the display. It is called by all other macros.
    RouteJawstoPC()
        JawsCursor()
        JawsPageUp()
        JawsHome()
    NextLine()
    NextLine()
    RouteInvisibleToJAWS0
        InvisibleCursor()
        NextWord()
```



```

        SayChunk()
    FCCursor()
MacroEnd

MacroBegin {D}
; Read the value in the display
    PerformMacroKey({'})
        if GetVerbosity() =
            then
                SayString ("Currently in Display")
            EndIf
        if GetVerbosity() = 1
            then
                SayString ("Displayed")
            EndIf
MacroEnd

MacroBegin {Escape}
; Clear Calculator
    {Escape}
        SayString ("Clear")
MacroEnd

MacroBegin {control+c}
;copy value from display to clipboard
    {control+C}
        PerformMacroKey ({' })
            SayString ("copied to clipboard from the display")
MacroEnd

MacroBegin {C}
; Clear Calculator
    {Escape}
        SayString ("Clear")
MacroEnd

MacroBegin {Delete}
; Clear the Display
    {Delete}
        SayString ("Display Cleared")
MacroEnd

MacroBegin {alt + d}
; Clear the Display
    If MenuActive () False

```



```

        then
            {Delete}
            SayString ("Display Cleared")
        Else
            {c}
    Endif
MacroEnd

MacroBegin {=}
; Perform function and put value in Display
    {=}
        SayString ("Equals")
        PerformMacroKey({' '})
MacroEnd

MacroBegin {e}
; Perform function and put value in Display
    {=}
        SayString ("Equals")
        PerformMacroKey({' '})
MacroEnd

MacroBegin {BackSpace}
; Delete the last value typed
    {BackSpace}
        SayString ("Back One")
MacroEnd

MacroBegin {R}
; Perform the Reciprocal Function
    if Menuactive()=--true
        then
            {r}
        else
            {R}
                SayString ("Reciprical")
                PerformMacroKey({' '})
    Endif
MacroEnd

MacroBegin {Control+M}
; Store the Displayed Value in Memory
    {Control+M}
        if GetVerbosity() = 0
            then

```



```

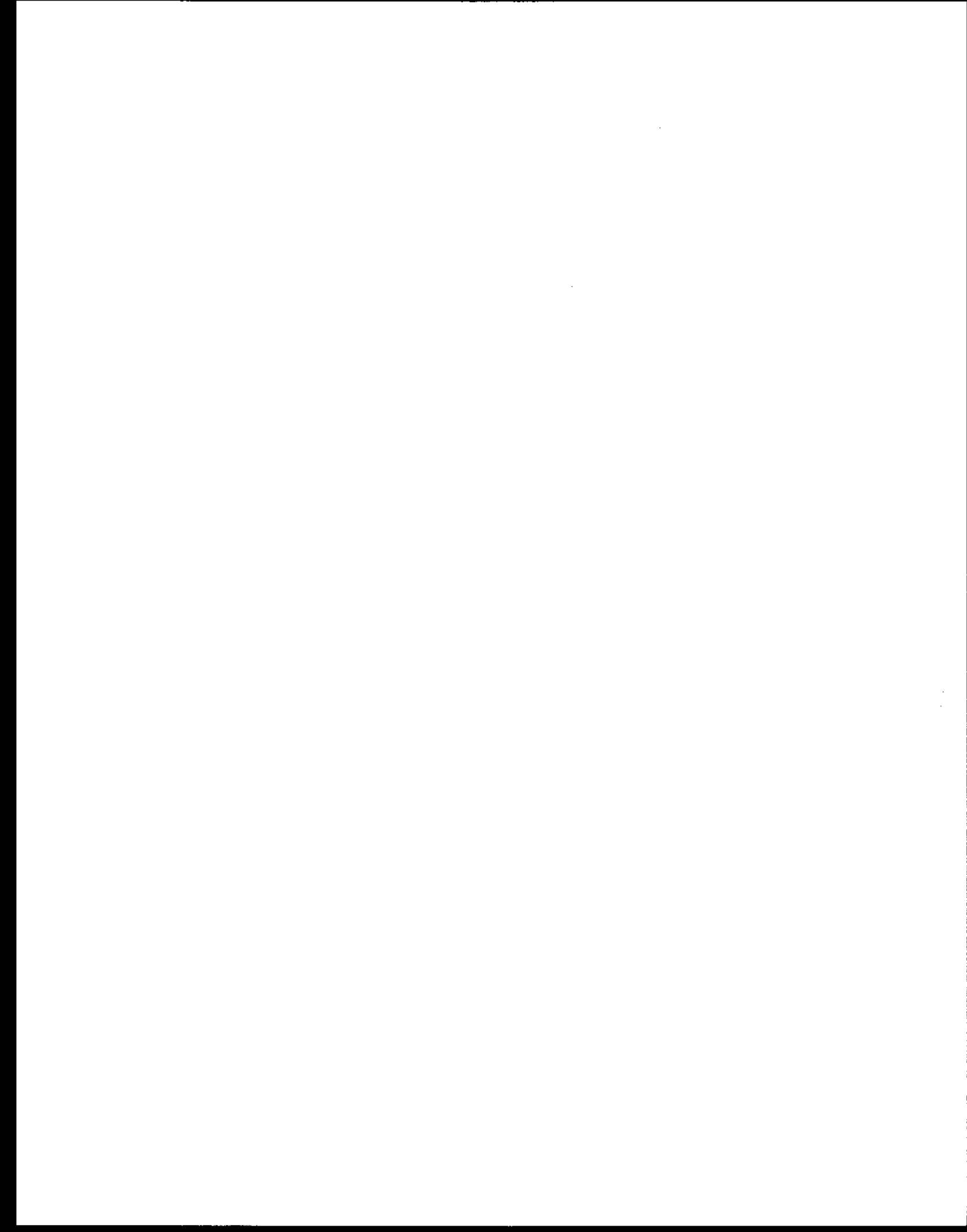
        SayString ("Stored in Memory")
    EndIf
    if GetVerbosity()    1
        then
            SayString ("Stored")
        EndIf
    PerformMacroKey({ ' })
MacroEnd

MacroBegin {Control+R}
; Recall value currently stored in Memory
{Control+R}
    if GetVerbosity() = 0
        then
            SayString ("Recalled from Memory")
        EndIf
    if GetVerbosity()    1
        then
            SayString ("Recalled")
        EndIf
    PerformMacroKey( { ' } )
MacroEnd

MacroBegin {Control+P}
; Add Displayed value to Memory value and store new value in Memory
{Control+P}
    PerformMacroKey( )
    if GetVerbosity() = 0
        then
            SayString ("Added to Value in Memory")
        EndIf
    if GetVerbosity()    1
        then
            SayString ("Added")
        EndIf
MacroEnd

MacroBegin {Control+L}
; Clear value from Memory
{Control+L}
    if GetVerbosity()    0
        then
            SayString ("Cleared Value from Memory")
        EndIf
    if GetVerbosity() = 1

```



```

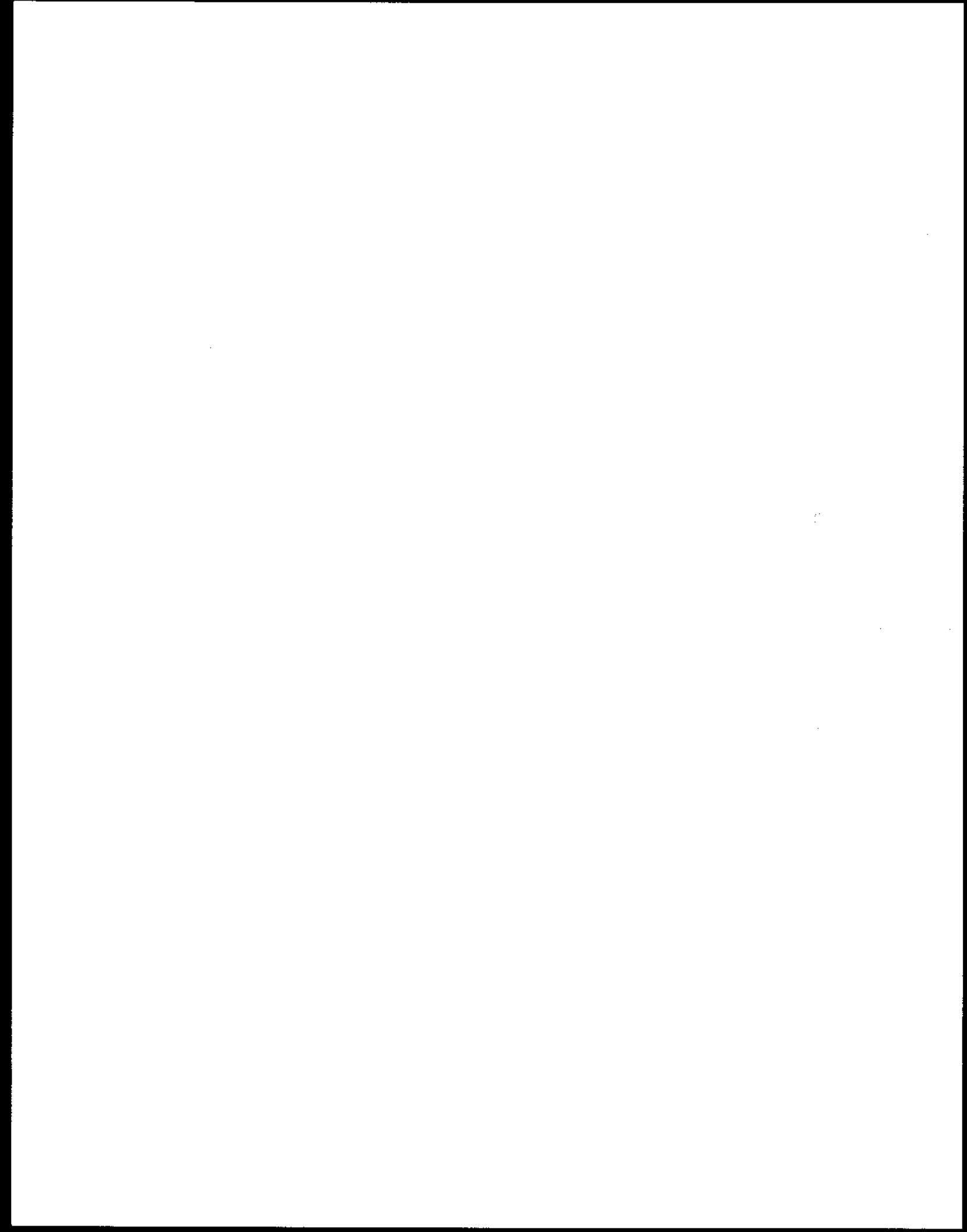
        then
            SayString ("Memory Cleared")
        Endlf
    if GetVerbosity() = 2
        then
            SayString ("Memory Cleared")
        Endlf
MacroEnd

MacroBegin {Alt+C}
    Clear value from Memory
    {Control-FL}
    if GetVerbosity()    0
        then
            SayString ("Cleared Value from Memory")
        Endlf
    if GetVerbosity()    1
        then
            SayString ("Memory Cleared")
        Endlf
    if GetVerbosity() = 2
        then
            SayString ("Memory Cleared")
        Endlf
MacroEnd

MacroBegin {F9}
; Changes "sign" of Displayed value
    {F9}
    if GetVerbosity()    0
        then
            SayString ("Sign changed")
        Endlf
    PerformMacroKey({' } )
MacroEnd

MacroBegin {Shift+2}
; Perform Square or square root operation on Displayed Value
    if (FindString(GetRealwindow(getcurrentwindow()), "inv", s_bottom,s_unrestricted))=0
        then
            SayString ("square root")
        else
            SayString ("square")
        Endif
    Shift+2 }

```



```

                PerformMacroKey({})
MacroEnd

MacroBegin {S}
; Speak and Store Displayed value in Memory
    if Menuactive()==true
        then
            {S}
        else
            {Control+M}
    if GetVerbosity() = 0
        then
            SayString ("Stored in Memory")
    Endlf
    if GetVerbosity() == 1
        then
            SayString ("Stored")
    Endlf
    PerformMacroKey({ } )
Endif
MacroEnd

```

```

MacroBegin {Alt+R}
; Recall and Speak value currently stored in Memory
    {Control+R}
    if GetVerbosity() = 0
        then
            SayString ("Recalled from Memory")
    Endlf
    if GetVerbosity() = 1
        then
            SayString ("Recalled")
    Endlf
    PerformMacroKey({ } )
MacroEnd

```

```

MacroBegin {A}
; Add displayed value to Memory
    {Control+P}
    PerformMacroKey({s })
    if GetVerbosity() = 0
        then
            SayString ("Added to Value in Memory")
    Endlf
    if GetVerbosity() = 1

```

11

20  
100

```

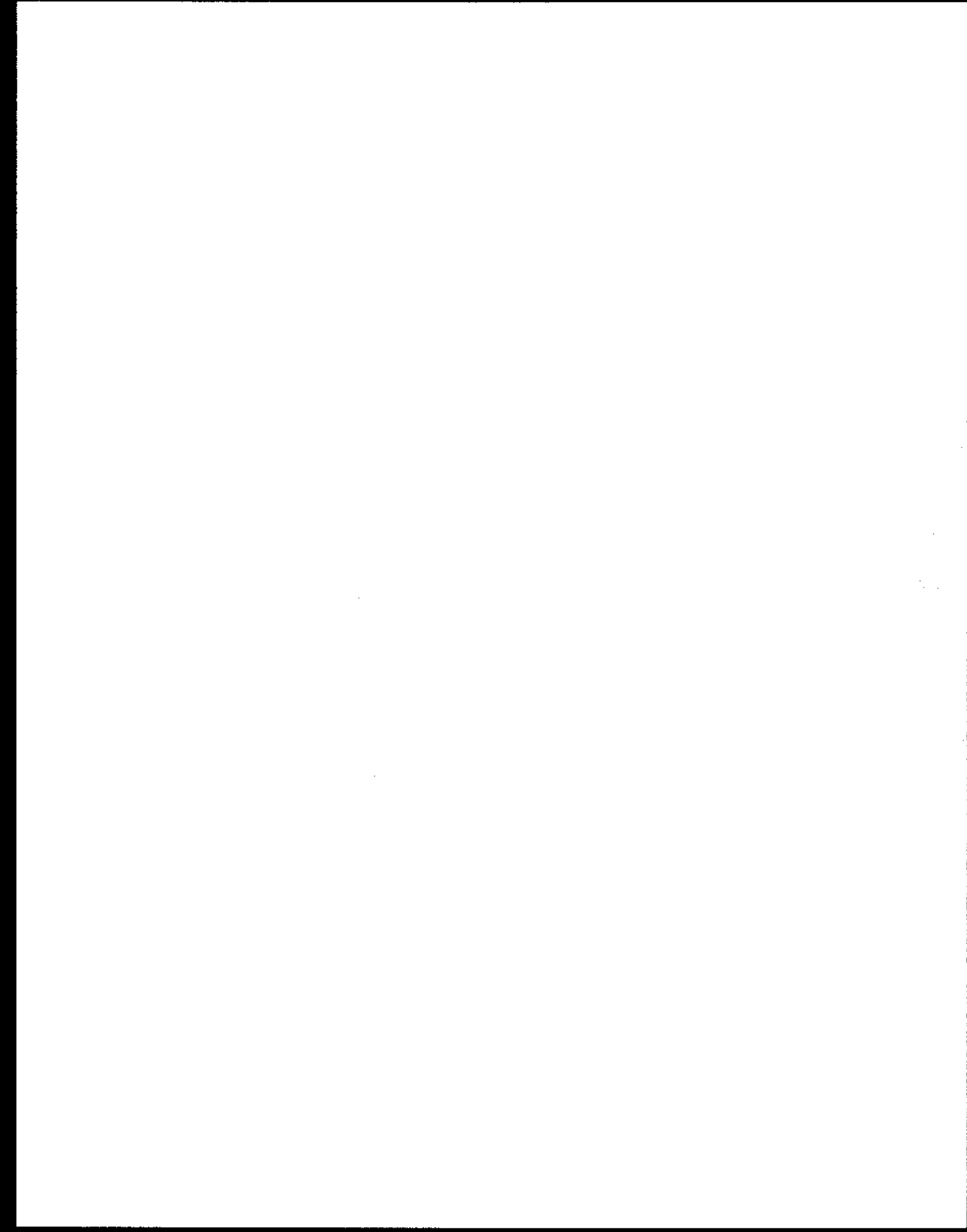
                                then
                                SayString ("Added")
                                Endlf
MacroEnd

MacroBegin {N}
;changes sign of displayed value
    if Menuactive()=--true
        then
            {n}
        else
            {F9}
    if GetVerbosity() = 0
        then
            SayString ("Sign changed")
    Endlf
    PerformMacroKey(1'1)
    Endif
MacroEnd

MacroBegin {Q}
;get the square root of the value in the display
    if (FindString(GetRealwindow(getcurrentwindow()), "inv", s_bottom,s_unrestricted))=0
        then
            {Shift-I-2}
            SayString ("Square Root")
            PerformMacroKey({'})
        else
            {i}
            {Shift+2}
            SayString ("Square Root")
            PerformMacroKey({'})
    Endif
MacroEnd

MacroBegin {insert+t}
;determine which calculator is turned on
    if (FindString(GetRealwindow(getcurrentwindow()), "inv", s_bottom, s_unrestricted))
        then
            PcCursor()
            SayString ("scientific calculator")
        else
            SayString ("standard calculator")
    Endif
    PcCursor()

```



MacroEnd

MacroBegin {z}

;toggle between scientific and standard modes

var

int echo,;

int verb

let echo=GetScreenEcho();

let verb=GetVerbosityO;

while GetScreenEcho()>0

ToggleScreenEcho()

Endwhile

while GetVerbosity()<2

ToggleVerbosity()

Endwhile

if (FindString(GetRealwindow(getcurrentwindowO), "in", s\_bottom, sunrestricted))

then

PcCursor 0

{alt+v}

{t}

while GetScreenEcho()!≠echo

ToggleScreenEcho()

Endwhile

while GetVerbosity()!≠verb

ToggleVerbosity()

Endwhile

if getverbosityO=0

then

SayString ("The standard calculator has been turned on")

else

if getverbosityO= 1

then

SayString ("Standard calculator on")

else

if getverbosity() =2

then

SayString ("standard calculator")

endif

endif

endif

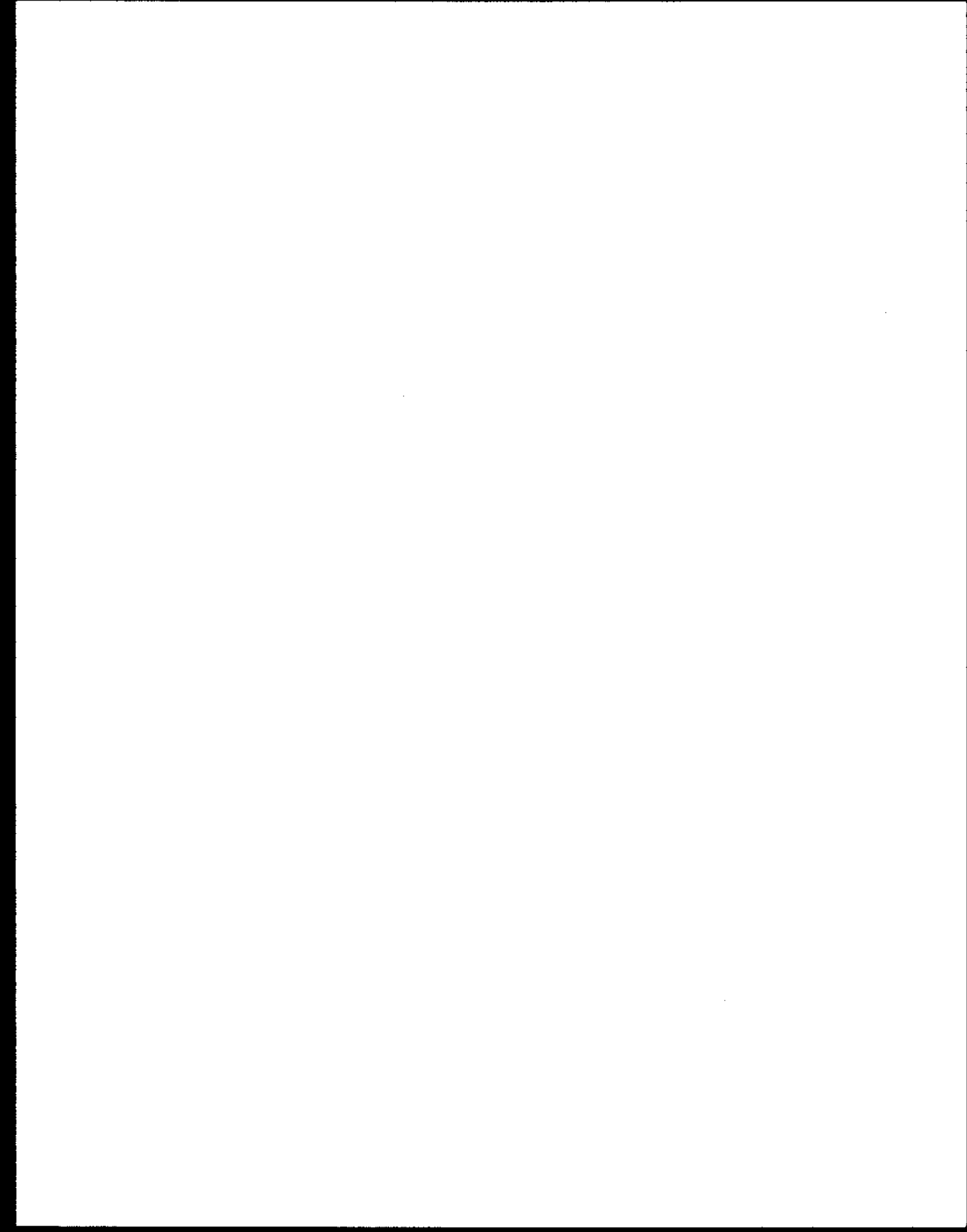
Else

{alt+v}

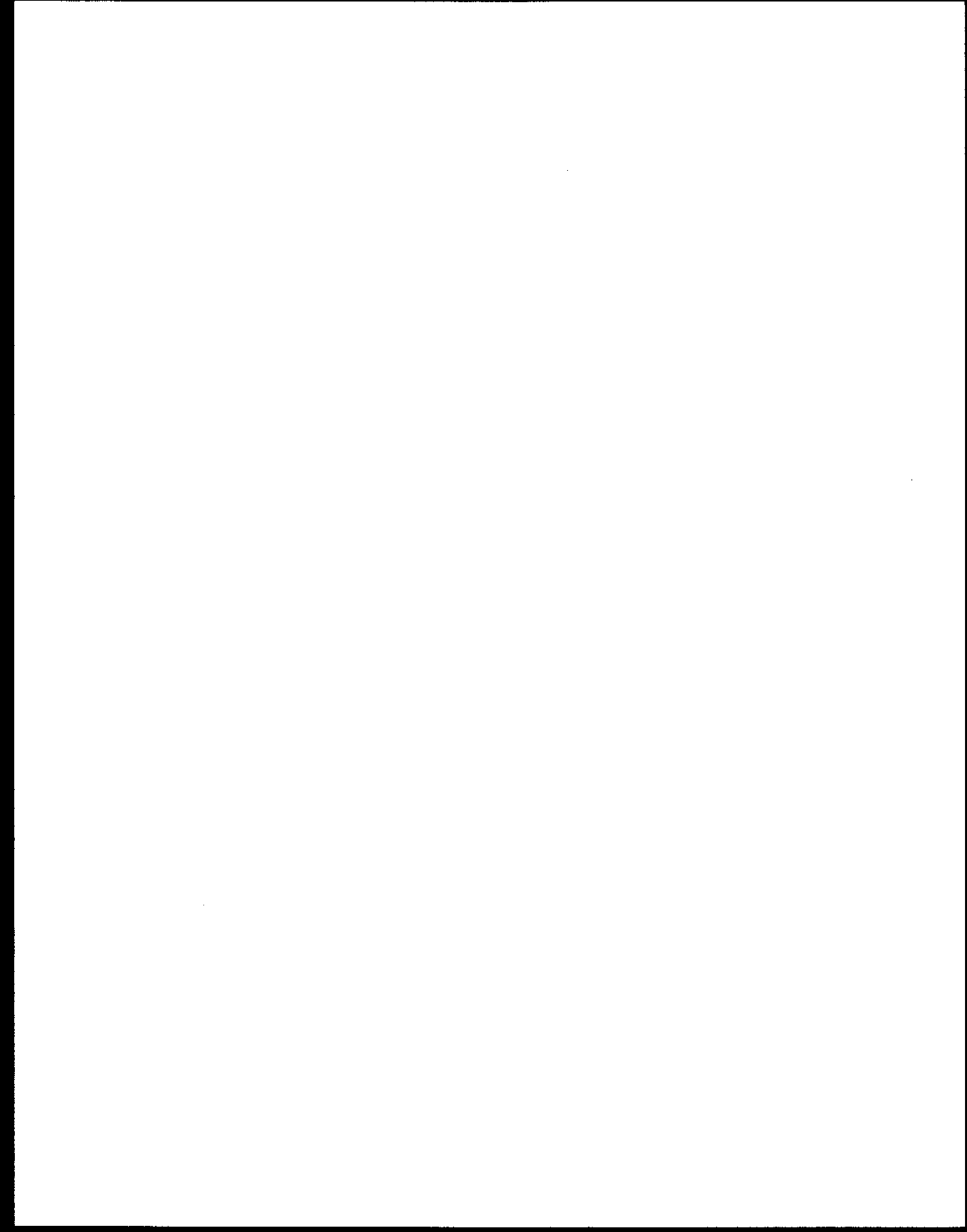
{s}

while GetScreenEchoO!≠echo

ToggleScreenEcho()



```
Endwhile
while GetVerbosity()!--verb
    ToggleVerbosity()
Endwhile
if getverbosity()=0
    then
        SayString ("The scientific calculator has been turned on")
    else
if getverbosity()= 1
    then
        SayString ("scientific calculator on")
    else
if getverbosity()=2
    then
        SayString ("scientific calculator")
endif
endif
endif
endif
MacroEnd
```



## **Practice Exercise: Phone Dialer Macro (dialer.jms)**

---

Task: Develop a macro that reads the "Speed Dial" list in the Windows 95 Phone Dialer utility.



## Glossary - JAWS for Windows Macros Functions

---

:-

Subtraction operator, used in math equations

Addition operator, used in math equations

Multiplication operator, used in math equations

· /

Division operator, used in math equations

: <

Less Than operator, used in IF statements

• <=

Less than or Equal to, used in IF statements

Is equal to, used in If statements

• >.

Greater than or equal to, used in IF statements

· >

Greater Than operator, used in IF statements

Not equal to, used in IF statements

: &&

And operator, used in IF statements

<sup>it</sup>

Or operator, used in IF statements

: AssignIntVar

Never called by the user, here for testing only

: AssignStringVar

Never called by the user, here for testing only



: Beep

Does a system beep

: BrailleLine

Sends the current line of text to a Braille display. This is automatically called ten times a second and needn't be called by a macro except when wanting to switch back to showing the current screen line after having displayed a string with BrailleString.

Returns void

BraillePanLeft

Displays text from the current line which is to the left of the first character currently shown on the Braille display

Returns int 1 if there was text to the left to be displayed, 0 otherwise

: BraillePanRight

Displays text from the current line which is to the right of the last character currently shown on the Braille display :returns int 1 if there was text to the right to be displayed, 0 otherwise

: BrailleString

Sends a string of text to the Braille display. Like SayString, but Brailles the information rather than speaks it. During the time that the string is on the Braille display, the display will not be showing the contents of the line on which the current cursor is positioned. To redisplay the line, use the BrailleLineMode.

Param 1 String the text to be displayed

Returns void

: CaretVisible

Returns int

Indicates whether or not there is an insertion point or caret visible or active on the screen. When an edit field has the focus then the caret is active, but if a menu is pulled down over the edit field then the caret is not active and JAWS will track the menu bar. Refer to the next/prior line macros for an example.

: Delay

Causes the macro to delay or suspend execution, to allow the application to do something, like move the cursor or display something new on the screen.

Param 1 int Number representing tenths of a second to delay

: DescribeFont

Speaks the font name, size, and attributes (underlined, bold, italics) at the current cursor position. Note that the size reported by this is not necessarily the same size that it will be when printed. Refer to FONTSIZE.TXT in the TECNOTES directory.



: DialogActive  
Returns int  
TRUE if a dialog box is currently active, FALSE otherwise

: DictionaryWizard  
Invokes the Dictionary Wizard for the current word

: EightDotBraille  
Switches to eight dot Braille  
Returns void

: else  
Optional part of IF statement

: endif  
Terminates the IF statement

endwhile  
End of While loop

: ExpandString  
Inserts a space between adjacent characters of a string. Can be used in conjunction with SayString to spell strings. Refer to the Insert+Numpad5 macro for an example.  
Returns string The text with spaces inserted  
Param: String The text to expand

: ExecuteKey  
Never called by the user, here for testing only  
# User callable

FindFirstAttribute  
Starting at the top of the current window, searches forward for text having the specified attributes. If successful, The active cursor is placed on the first matching character.  
:param: int the attributes to search for. These should be represented by one or more of the constants ATTRIB BOLD, ATTRIB UNDERLINE, ATTRIB ITALIC, ATTRIB HIGHLIGHT, or ATTRIB STRIKEOUT. When searching for text having multiple attributes, items should be combined using +. e.g.  
FindFirstAttribute(ATTRIB **BOLD**+ATTRIB **UNDERLINE**) would search for bold text that is also underlined.  
:returns: int 1 if the specified attribute is found, 0 otherwise

: FindGraphic  
: Searches for a graphic symbol and moves the JAWS cursor to it if it finds it. The graphic symbol must have a text label associated with it, such as assigned by the Graphics Wizard. The text label is how each graphic is identified.



:param1 handle Handle of window to search  
:param2 string graphic text label to search for  
:param3 int 0 if search should start at the top, 1 if search should start at the bottom.  
:param4 int 0 if search should extend outside of current window or control, 1 if search should be restricted to current window or control.  
:returns int non-zero on success, 0 on failure

#### Find' listAttribute

: Starting at the bottom of the current window, searches backward for text having the specified attributes. If successful, The active cursor is placed on the first matching character.

:param1 int the attributes to search for. These should be represented by one or more of the constants **ATTRIB\_BOLD**, **ATTRIB\_UNDERLINE**, **ATTRIB\_ITALIC**, **ATTRIB\_HIGHLIGHT**, or **ATTRIB\_STRIKEOUT**. When searching for text having multiple attributes, items should be combined using +. e.g.

FindLastAttribute(**ATTRIB\_BOLD**+**ATTRIB\_UNDERLINE**) would search for boled text that is also underlined.

:returns int 1 if the specified attribute is found, 0 otherwise

#### : FindNextAttribute

: Starting at the current cursor's location, searches forward for text having the specified attributes. If successful, The active cursor is placed on the first matching character.

:param1 int the attributes to search for. These should be represented by one or more of the constants **ATTRIB\_BOLD**, **ATTRIB\_UNDERLINE**, **ATTRIB\_ITALIC**, **ATTRIB\_HIGHLIGHT**, or **ATTRIB\_STRIKEOUT**. When searching for text having multiple attributes, items should be combined using +. e.g.

FindNextAttribute(**ATTRIB\_BOLD**+**ATTRIB\_UNDERLINE**) would search for boled text that is also underlined.

:returns int 1 if the specified attribute is found, 0 otherwise

#### FindPriorAttribute

: Starting at the current cursor's location, searches backward for text having the specified attributes. If successful, The active cursor is placed on the first matching character.

:param1 int the attributes to search for. These should be represented by one or more of the constants **ATTRIB\_BOLD**, **ATTRIB\_UNDERLINE**, **ATTRIB\_ITALIC**, **ATTRIB\_HIGHLIGHT**, or **ATTRIB\_STRIKEOUT**. When searching for text having multiple attributes, items should be combined using +. e.g.

FindPriorAttribute(**ATTRIB\_BOLD**+**ATTRIB\_UNDERLINE**) would search for boled text that is also underlined.

:returns int 1 if the specified attribute is found, 0 otherwise

#### : FindString

: Searches for a string of text on the screen, if found,



it moves the JAWS cursor to it.

:param1 handle Handle of window to search, such as returned by GetCurrentWindow.

:param2 string The string to search for

:param3 int 0 if search should start at the top, 1 if search should start at the bottom.

:param4 int 0 if search should extend outside of current window or control, 1 if search should be restricted to window or control associated with the passed handle.

:returns int non-zero on success, 0 on failure

: Flush

: Flushes the speech synthesizer, stops speaking

(Not currently implemented)

: For

Start of For Loop

: GetActiveCursor

:returns int

: Determines which cursor is currently on: 0=Jaws cursor, 1=PC, 2=Invisible cursor, 3=Braille cursor.

: GetAppFileName

: Returns the file name of the currently executing program.

Useful in determining what the name of a macro file should be when creating one for a specific application.

:returns string The name of the executing program

: GetAppMainWindow

: Returns the handle of the application main window associated with the passed handle. It works its way up the window hierarchy until it gets to the application's main window.

:Param1 : handle handle of window to start with.

:Returns handle The handle of the application's main window.

: GetAppTitle

:returns string The title of the application.

: Gets the title of the current application program.

: GetBrailleMode

: The Braille terminal can be in one of several modes.

GetBrailleMode allows a macro to base its behavior on which mode is active. When IN line mode (**BRL MODE LINE**), the Braille display reflects the contents of the line with the active cursor. When the cursor moves, the line contents change. When in String Mode (**BRL MODE STRING**), the Braille display shows information explicitly sent to it and does not reflect the contents of the screen. Strings sent to the display while in this mode are queued. They can be cycled through using the NextBrailleString and PriorBrailleString



:Returns int Identifier of current mode

: GetCharacter

: Retrieves the character or graphic label on which the current cursor is positioned

:returns string current character or label

: GetChunk

Retrieves the current chunk of text, that which was written to the screen in a single operation. Returns String the contents of the chunk

: GetControlID

: Returns the numeric identifier associated with a window.

If the passed handle does not refer to a child window, always returns 0. Control IDs are unique within a dialog box, except when referring to controls containing static text. In this case, several controls may have the same ID.

:param I handle Window for which information is desired

:returns int the ID of the requested control

: GetCurrentControlID

: If in a dialog box, returns the numeric identifier associated with the current control. If not in a dialog box, always returns 0. Control IDs are unique within a dialog box, except when referring to controls containing static text. In this case, several controls may have the same ID.

:returns int the ID of the current control

: GetCurrentWindow

: Returns the handle of the current window. If the PC cursor is on, then this is the window containing the caret or highlighted item. If the JAWS cursor is on, this is the window containing the mouse pointer. This is frequently used to get the handle to be used as a parameter in other macro , i.e. GetWindowType or GetWindowClass.

:Returns handle The handle of the window with the "active" cursor

: GetCursorCol

:returns int

: Gets the column or horizontal position of the active cursor, use SayInteger to speak it.

: GetCursorRow

:returns int

: Gets the row or vertical position of the active cursor, use SayInteger to speak it.

: GetDate

: Gets the system date. (Not implemented)



: GetDefaultButtonName  
: When in a dialog box, returns the label of the default button, the one which will be executed when the user presses ENTER :returns string the name of the default button

: GetDefaultJCFOption  
: Returns the default value of a numeric option. See SetDefaultJCFOption for more details.  
:Returns int Value of the specified option  
:Paraml int Identifier of the option to be returned. It is best to use one of the **OPT\_ or OPTBRL** constants in HJCONST.JMH.

: GetField  
: retrieves the contents of the current field  
:returns String the contents of the field

: GetFirstChild  
: Returns the handle of the passed window's first child or 0 if none. Useful if you want to traverse a list of windows, you would use GetParent and then GetFirstChild to get to the first window in the list.  
:Paraml: handle handle of parent window  
:Returns handle The handle of the first child window

: GetFirstWindow  
: Returns the handle of the first window at the same logical level as the passed handle. Gets the first window in a list of windows.  
:Paraml: handle handle of window at same level  
:Returns handle The handle of the first window at that level

: GetFocus  
: Returns the handle of the window with focus or 0 if none. This is independent of which cursor, JAWS or PC, is active.  
:Returns handle The handle of the window with focus

: GetJCFOption  
: Returns the current value of a numeric option. See SetJCFOption for more details.  
:Returns int Value of the specified option  
:Paraml int Identifier of the option to be returned. It is best to use one of the **OPT\_ or OPTBRL** constants in HJCONST.JMH.

: GetLine  
: Retrieves the text of the line on which the current cursor is positioned  
:returns string contents of line



: GetNextWindow  
: Returns the handle of the next window or 0 if none.

Use to traverse a list of windows.

:Param1: handle handle of window  
:Returns handle The handle of the next window

: GetParent  
: Returns the handle of the passed window's parent window or 0 if none. A parent window is the one that created this window. It is useful if you want to traverse a list of windows or controls.

:Param1: handle handle of window  
:Returns handle The handle of the parent window

: GetPriorWindow  
: Returns the handle of the prior window or 0 if none.

Use to traverse a list of windows.

:Param1: handle handle of window  
:Returns handle The handle of the prior window

: GetRealWindow  
: starting with the passed window handle, traverses upwards through the window list until a window with a title-bar is found. Then it returns the handle of that window, the one with a title or title bar. A "real" window is one that has a title.

:Param1: handle handle of window being used as starting point.  
:Returns handle The handle of the real window

: GetScreenEcho  
:returns int  
: Gets the Screen Echo settings, to determine how much of the data being written to the screen is actually spoken: 0=None, 1=Highlighted text only, 2=All. It is used to control the Text event and Highlighted Text event macros, refer to them and the Insert+S macro for examples.

: GetVerbosity  
:returns int  
: Gets the verbosity level, refer to the Insert+V macro for example.

: GetWindowClass  
:param1 handle Handle of window to check  
:returns string The class name

100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200

: Gets the class of the window. This is the name given by the application, can be anything. Every window that is created has a class name associated with it. There are several predefined window classes, like "edit", "listbox", "combobox", etc. The string returned by this will be one of these or it will be a class name unique to the application. Refer to the Insert+F 1 macro for a list of the standard class names.

: GetWindowName

: Returns the title of the passed window. See the "FocusChange" event macro for example.

:Paraml: handle handle of window

:Returns string The name or title of the window

: GetWindowType

:paraml handle Handle of window to check

:Returns string The type of the window

: returns a text string containing the type of the window whose handle was passed as a parameter. Type is less specific than Window Class, returns names like Dialog box, Button, or List box, the standard types that can be found in the Insert+F1 macro. Returns "Unknown" if it does not match one of the standard types.

: GetWindowTypeCode

:paraml handle Handle of window to check

:Returns int a number indicating The type of the window

: returns a number representing the type of the window whose handle was passed as a parameter. Type is less specific than Window Class. This is similar to GetWindowType but returns a number rather than a textual string. The number will be the same even if a nonEnglish version of JAWS is running. GetWindowType will return a string specific to the language of the current JFW session. The results of GetWindowTypeCode can be compared to WT\_ constants defined in hjconst.jmh.

GetWord

: Retrieves the text of the word on which the current cursor is positioned

:returns string current word

: GraphicWizard

: Invokes the Graphic Wizard dialog, which lets you assign a label to a graphic symbol. Be sure the JAWS cursor is on so you can be sure it is on a graphic symbol, the Graphics Wizard will always get what the JAWS cursor is on.

: HasFocusRect

: Invokes the Graphic Wizard dialog, which lets you assign a label to a graphic symbol. Be sure the JAWS cursor is on so you can be sure it is on a graphic symbol, the Graphics Wizard will always get what the JAWS cursor is on.

: HasTitleBar



: Returns True if the passed window has a title bar, such as dialog boxes or application windows.

:Paraml: handle handle of window

:Returns int

: If

: Start of If-then-Else statement

:paraml string The equation or that provides a true or false result

InvisibleCursor

: Turns on the invisible cursor and turns off the JAWS or PC cursor. The Invisible Cursor performs like the JAWS cursor except that it does not move the mouse pointer, it is strictly a reading cursor. Some applications sense the presence and location of the mouse and change the screen or make a selection automatically, even if you do not want them to. In those cases you should use the Invisible Cursor if you do not want the application to update the screen.

: IsInvisibleCursor

: Returns true if the invisible cursor is on, false otherwise.

:returns int

: IsJAWSCursor

:returns int

: TRUE if the JAWS cursor is currently on

: IsPCCursor

:returns int True or False, 1 or 0

: Returns true if the PC cursor is currently on, false otherwise.

: IsSameMacroKey

: Returns true if same macro key has been pressed twice within half a second. One key can be used for two different , like SayWord and SpellWord. See Insert+Numpad5 in the DEFAULT.JMS file for example.

:returns int True or False, 1 or 0

: JAWSCursor

: Turns on the JAWS cursor, turns off all other cursors.

: JAWSEnd

: Moves JAWS cursor to right side of window if it is on, or does the applications End if PC cursor is on.



: JAWSHome  
: Moves JAWS cursor to left side of window if it is on, or does the applications Home if PC cursor is on.

: JAWSPageDown  
: Moves JAWS cursor to bottom of window if it is on, or does the applications Page Down if PC cursor is on.

: JAWSPageUp  
: Moves JAWS cursor to top of window if it is on, or does the applications Page Up if PC cursor is on.

: JawsWindow  
: Invokes the JAWS window, gives JAWS the focus. Makes JAWS the currently running application.

: LeftMouseButton  
: Simulates pushing the left mouse button one time, press it twice quickly to double click.

: LeftMouseToggle  
: Toggles the pressed/released state of the left mouse button. returns 1 if the button is being pressed, 0 if it is being released. Press it again to unlock it Wit is locked down.

:Returns int The locked/unlocked status

: MenusActive

:returns int

: TRUE if a menu is currently active, FALSE otherwise

: MessageBox

: Pops up a message on the screen

:Param 1 String Text to be put in message box.

MoveToGraphic

:param 1 int Direction 0=First, 1=Next, 2=Prior, 3=Last

:returns int

; Moves JAWS cursor to the specified graphic element in the current window. If you use "MoveToGraphic(0)" then the JAWS cursor/mouse will be moved to the first graphic item in the current window (maybe the Application Control Menu in the top left corner).

If you want to go to the next graphic item from there then use a parameter of "1". A parameter of "2" will move to the previous graphic, and "3" will take the JAWS cursor to the last graphic item in the window. Use this in conjunction with "SayChunk" to read the label of the graphic item.

: MoveToWindow

: Moves the current cursor to the top of the specified window. If the PC cursor is on, the JAWS cursor is turned on and moved.



:paraml hwnd handle of desired window  
:returns int TRUE if move successful, FALSE otherwise

: Next  
: next

: NextBrailleString  
: Displays the next string in the Queue of items sent out to the Braille display  
:returns int TRUE if there was a next item, FALSE otherwise

: NextCharacter  
: Moves the active cursor to the next character. If the PC cursor is on then the application moves the "cursor" as it sees fit. If the JAWS cursor is on then JAWS will try to move to the next character or graphic item.

: NextChunk  
: Moves the active cursor to the next chunk of text

: NextLine  
: Moves the active cursor to the next line

: NextUnit  
: Moves the active cursor to the next unit of text  
(Not implemented)

: NextWord  
: Moves the active cursor to the next word

: Not  
: Reverses the IF statement

: PassKeyThrough  
: passes key through to the underlying application, bypassing JAWS macros. Used when a JAWS key is in conflict with a key combination normally used by the application program. (If the JAWS macro key is there it will normally take precedence and perform the macro, if PassKeyThrough is pressed first then the JAWS macro is ignored and the application will get the key.

: Pause  
: Pauses the current macro, so that other applications can get control and do whatever they need to do, then control will return to JAWS and the macro will continue. The actual amount of time is variable, depending on the number of other applications currently running and the amount of work they need to do.



PCCursor  
: Turns on the PC cursor, turns off all other cursors.

: PerformMacroKey  
: Performs the macro assigned to the specified key, just like it was physically pushed.  
When that macro is finished it returns to the macro that performed it.  
:param1 key the key with the macro to be performed

: PriorBrailleString  
: Displays the prior string in the Queue of items sent out  
to the Braille display  
:returns int TRUE if there was a prior item, FALSE otherwise

: PriorCharacter  
: Moves the active cursor to the prior character

: PriorChunk  
: Moves the active cursor to the prior chunk of text

: PriorLine  
: Moves the active cursor to the prior line

: PriorUnit  
: Moves the active cursor to the prior unit of text  
(not implemented)

: PriorWord  
: Moves the active cursor to the prior word

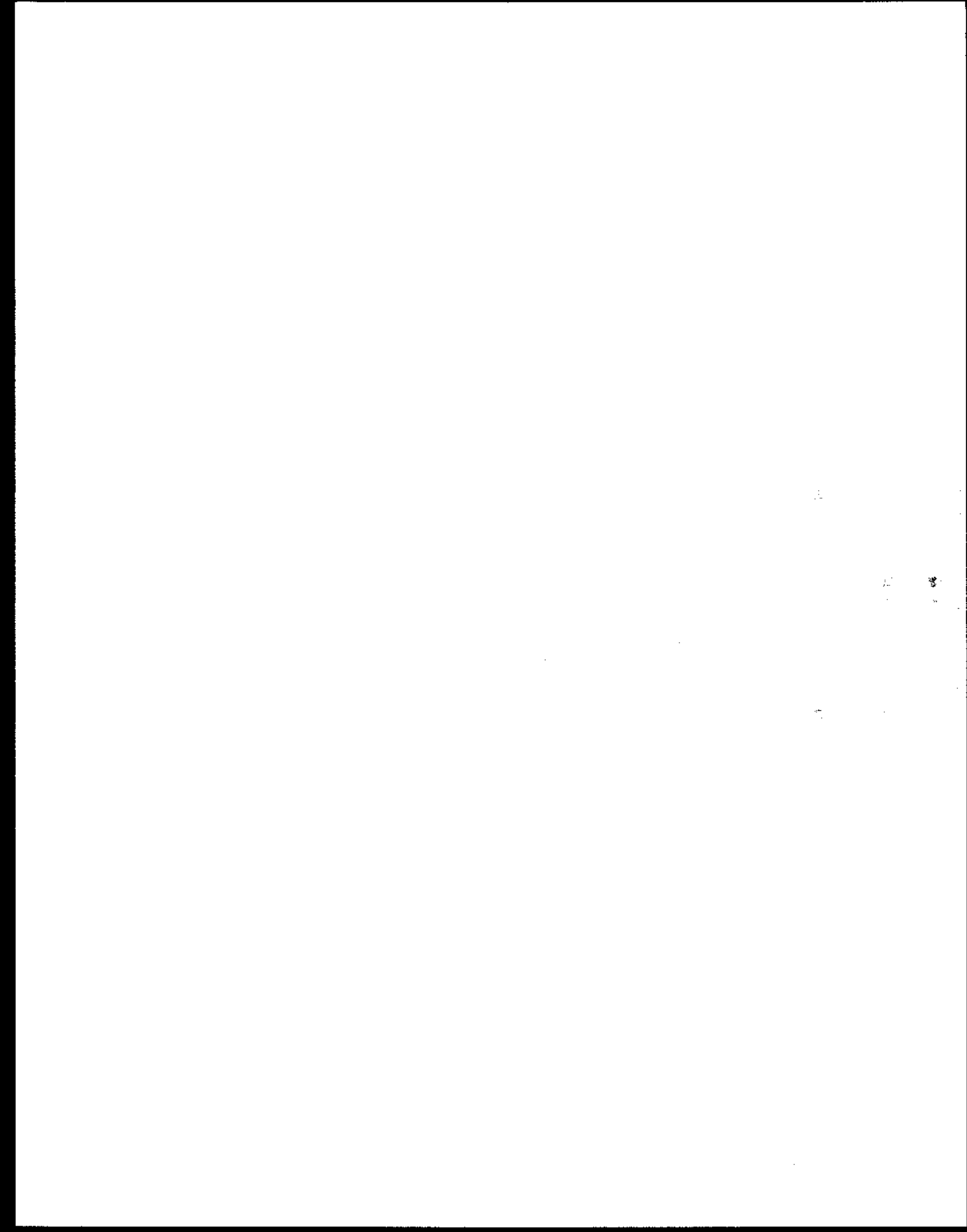
: PushInt  
: Pushes an integer on the stack

: PushIntVar  
: never called by the user, here for testing only

: PushString  
: Pushes a pointer to a string on the stack

PushStringVar  
: never called by the user, here for testing only

: Refresh  
: Refreshes the screen  
:Returns void



: RestoreCursor  
: restores the cursor saved by the most recent call to SaveCursor, i.e. it turns on the cursor that was on. Will move the JAWS cursor back to its position when it was saved.

: RightMouseButton  
: Simulates pushing the right mouse button one time, press it twice quickly to double click.

: RouteInvisibleToJaws  
: Moves the invisible cursor to the JAWS cursor.

: RouteInvisibleToPc  
: Moves the invisible cursor to the PC cursor.

: RouteJawsToInvisible  
: Moves the JAWS cursor to the invisible cursor.

: RouteJawsToPc  
: Moves the JAWS cursor to the PC cursor. Puts the mouse pointer on top of the caret (if it is there) or the highlighted text or other point that the PC cursor is tracking.

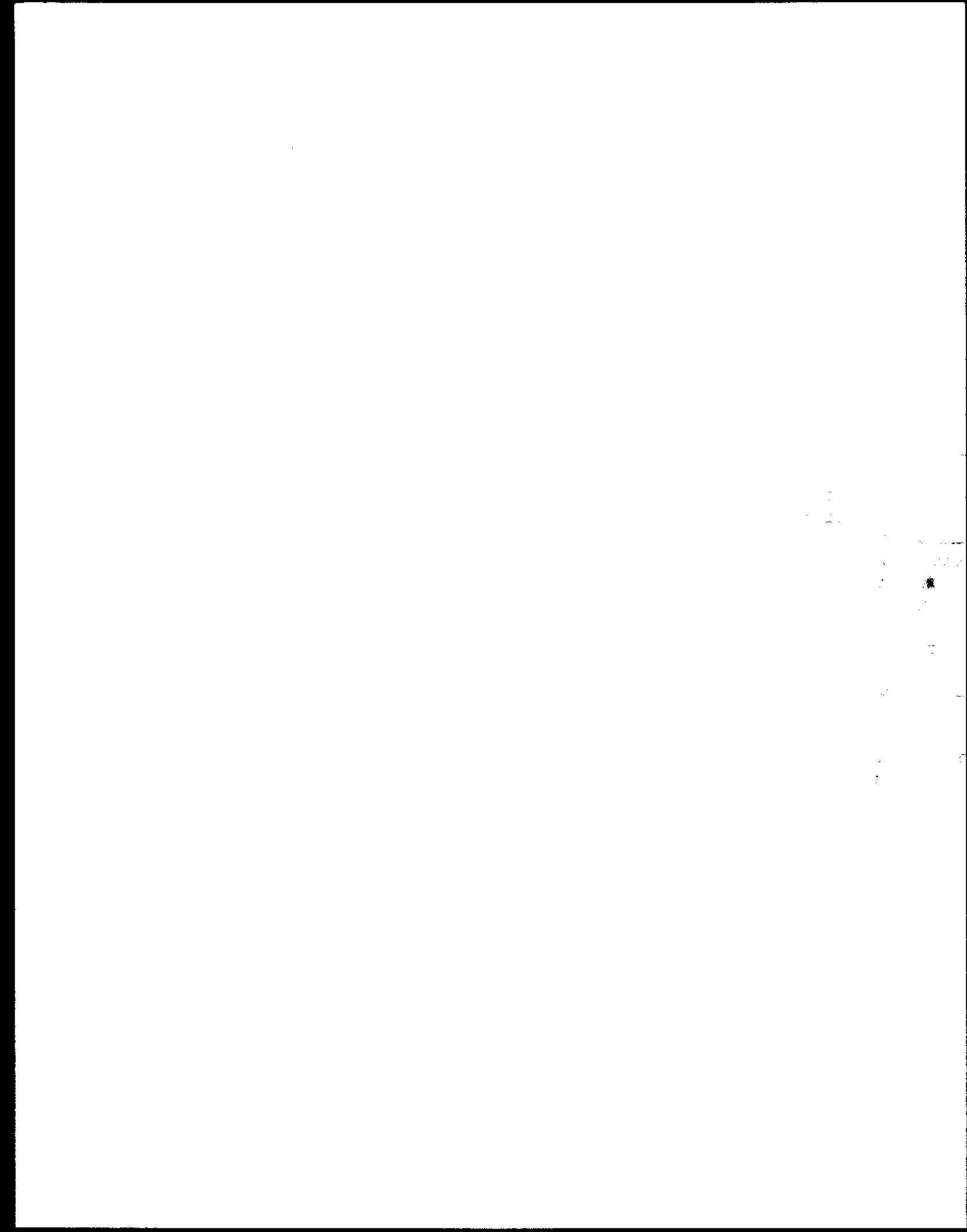
: RoutePcToJaws  
: Moves the PC cursor to the JAWS cursor when possible. This is the same as left mouse click, which asks Windows to move the insertion point to the mouse pointer. It will not work in some situations.

: Run  
:param 1 string The name of the program to run  
: Runs the program whose name and path are specified in the string that is passed.

: SaveCursor  
: saves the current cursor, i.e. it remembers which cursor is currently on. If the JAWS cursor is on, its position is also saved, so it can be returned there with the restore

: Say  
:param1 string The string to speak  
:param2 int Category (Output Type) of string being spoken. Output Types have OT\_ prefix and are defined in HJCONST.JMH.  
: Like SayString, but with greater control. The Output Type is used by JAWS to determine the appropriate voice and punctuation settings for the type of data being spoken.

: SayActiveCursor  
: Says the name of the active cursor, JAWS or PC



: SayAll  
: Reads the window, from the active cursor down to the bottom of the window. The cursor stays where it was. The control key or shift key will silence it. It will not stop and place the cursor on the last word spoken yet, that will come later.

: SayCharacter  
: Says the current character, or graphic label, where the active cursor is.

: SayChunk  
: speaks the current chunk of text, that which was written to the screen in a single operation  
:returns void

: SayColor  
: Says the foreground and background colors of the text at the current cursor position  
:returns int TRUE if text was found, FALSE otherwise

  SayControl  
:paraml handle Handle of control to speak  
: Speaks the contents of the Control along with it's prompt message. A "Control" typically is one of the fields inside a dialog box, like a list box. It is a child window, usually inside a dialog box. The parameter is the handle of the control or child window, refer to "Window handles" for more info.

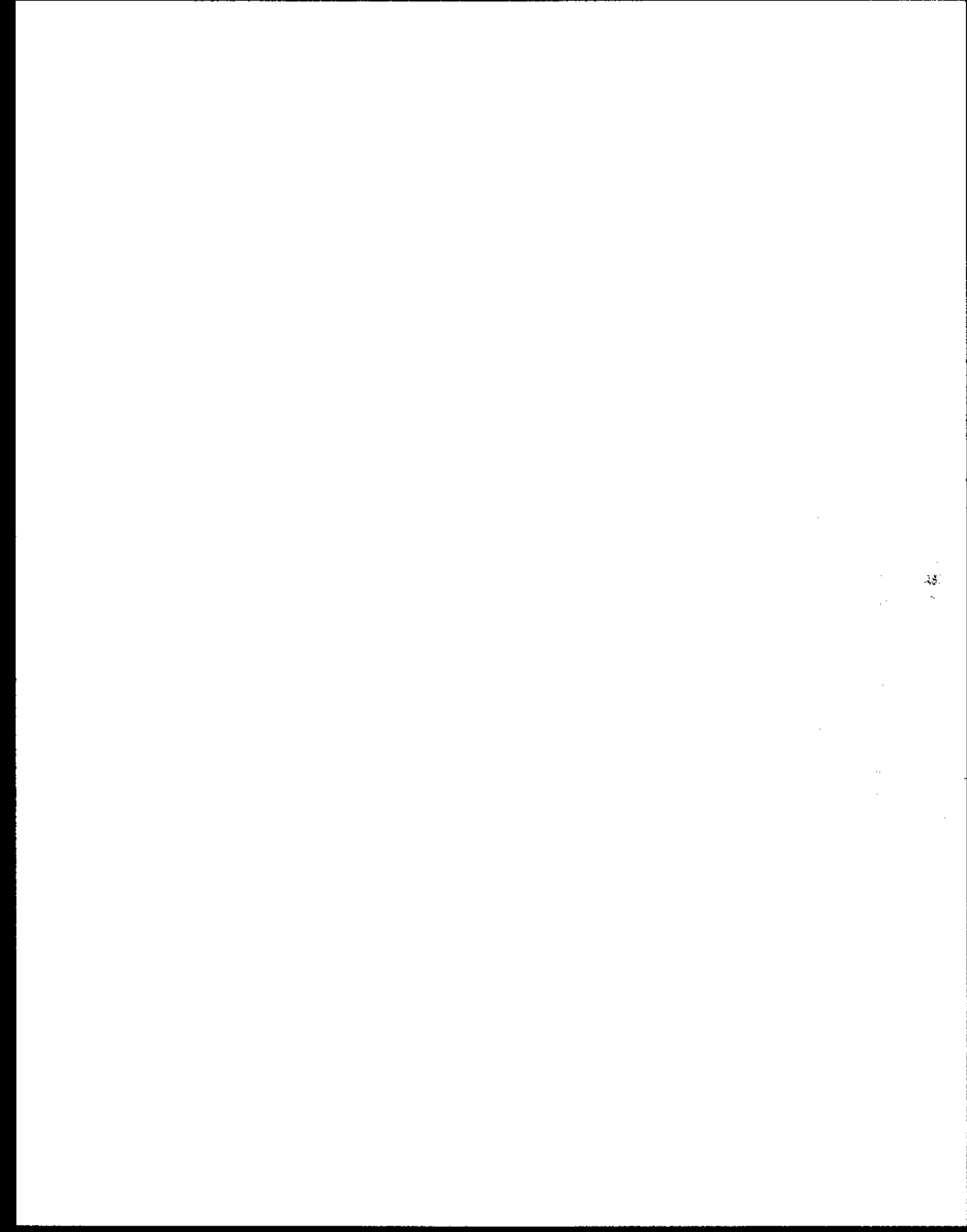
: SayCursorPos  
:paraml int The cursor to use  
: Says the position of the cursor. Refer to the DEFAULT.JMS macro file for an example.

: SayField  
: Says the current field of text. This is a block or group of text that has the same attribute, e.g. bold, underlined, italics, or strike out. The attribute must be contiguous.

: SayFocusRect  
: Says the contents of focus rectangle. Returns TRUE if any text was spoken, FALSE otherwise. (Not implemented currently)  
:paraml handle Handle of window  
:returns int True or False, 1 or 0

: SayFromCursor  
: speaks text to the right of the current cursor on the same line, and includes the character that the cursor is on.

: SayInteger  
:paraml int The number to speak  
: Says the number.



: SayLine  
: Says the current line of text, the "line" that the active cursor is on. This may include items that are slightly up or down from the current "line", or it may not include them. All items do not necessarily line up exactly, unlike DOS.

: SayString  
:param1 string The string to speak  
: Speaks the string of characters enclosed in quote marks.

: SayToCursor  
: speaks text to the left of the current cursor, on the same line. It does not read the current character, just up to it.

: SayUnit  
: Says the current unit of text  
(Not implemented currently)

: SayWindow  
:param1 handle Handle of window to speak  
:param2 int 0 = speak everything, 1 = speak highlighted text only  
: Speaks the contents of the window associated with the passed window handle (parameter 1). If parameter 2 is "0" then it speaks all the text in the window, if parameter 2 is "1" then it speaks only the highlighted text.

: SayWord  
Says the current word, or graphic label, where the active cursor is.

: SayWindowTypeAndText  
:param1 handle Handle of window to speak  
: Speaks the title, type of window, contents of window, and other important information, like the prompt or checked/unchecked status of the passed window. Also marks this text as having been spoken, so the Text and highlightedText events won't speak it again. See the FocusChange event macro for an example.

: SDMGetFirstControl  
: Returns the ID of the first control in an SDM dialog  
:Param 1 handle handle of SDM window  
:Returns int ID of item in window

: SdmGetFocus  
: Used to get the ID of the current control in an SDM dialog box, which is used in SdmSayWindowTypeAndText. Refer to the FocusChange event macro in the Word for Windows macro file, **WORD.JMS**.  
:param1 handle handle of current window, returned by GetFocus.

2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030

:returns int The ID of the current control within an SDM dialog

: SDMGetLastControl

: Returns the ID of the last control in an SDM dialog

:Param1 handle handle of SDM window

:Returns int ID of item in window

: SDMGetNextControl

: Returns the ID of the next control in an SDM dialog

:Param1 handle handle of SDM window

:Param2 hit ID of current control in window

:Returns int ID of next control in window, or 0 if there is no next control

: SDMGetPrevControl

: Returns the ID of the previous control in an SDM dialog

:Param I handle handle of SDM window

:Param2 int II) of current control in window

:Returns hit II) of previous control in window, or 0 if there is no previous control

: SdmSayControl

: Behaves the same as SayControl, only the parameters are different. An SDM dialog has only one window handle regardless of which control you are on in the dialog box. That is why the second parameter is the control ID. The current control id is gotten by SdmGetFocus. That is, GetFocus will return the handle of the SDM dialog, and SdmGetFocus will return the ID of the current control. The "Control" is the item inside the dialog box, like an edit field or a list box. See the WORD.JMS file for examples.

:Param1 handle handle of SDM window, returned by GetFocus.

:param2 int ID of item in window, returned by SdmGetFocus

: SdmSayStaticText

: Says the static text in an SDM dialog box. This is the text that is not associated with any of the input fields, i.e. not a title for any of these fields.

:param I handle handle of SDM window, returned by GetFocus.

: SdmSayWindowTypeAndText

: Behaves the same as SayWindowTypeAndText, only the parameters are different. An SDM dialog has only one window handle regardless of which control you are on in the dialog box. That is why the second parameter is the control ID. The current control id is gotten by SdmGetFocus. That is, GetFocus will return the handle of the SDM dialog, and SdmGetFocus will return the ID of the current control. The "Control" is the item inside the dialog box, like an edit field or a list box. See the **WORD.JMS** file for examples.

:Param1 handle handle of SDM window, returned by GetFocus.

:param2 int ID of item in window, returned by SdmGetFocus



: SetBrailleMode

: Sets the active mode for Braille output. (See GetBrailleMode for more details)

:param1 int identifier of desired mode

:returns int TRUE if specified mode is valid, FALSE otherwise

: SetDefaultJCFOption

: Changes the default value of a numeric option. Option identifiers are listed with an OPT\_ prefix, (OPTBRL\_ if they relate specifically to Braille) in the HJCONST.JMH file. They correspond to .JCF settings. When an option is changed by this the new value is active for the remainder of the JFW session unless an application is run which has a specific setting for this option in its .JCF file.

:Returns int TRUE on success, FALSE on failure

:Param1 int Identifier of the option to be changed. It is best to use one of the OPT\_ or OPTBRL\_ constants in HJCONST.JMH.

:Param2 int New value for the option

: SetJCFOption

: Changes the current value of a numeric option. Option identifiers are listed with an OPT\_ prefix, (OPTBRL\_ if they relate specifically to Braille) in the HJCONST.JMH file. They correspond to .JCF settings. When an option is changed by this the new value remains active until switching to another application. At that time the value changes to the one specified in the application's .JCF file, or if there is no .JCF file for the application, to that specified in DEFAULT.JCF. (See SetDefaultOption for a which changes the default value of an option for the remainder of the JFW session.

:Returns int TRUE on success, FALSE on failure

:Param1 int Identifier of the option to be changed. It is best to use one of the OPT\_ or OPTBRL\_ constants in HJCONST.JMH.

:Param2 int New value for the option

: SetVoice

: Sets a voice parameter

(Currently not implemented)

: SixDotBraille

: Switches to six dot braille

:returns void

: Spell String

: Similar to SayString, but it spells rather than says it as words.

:param1 string The item to spell.

: SpellWord

: Similar to SayWord, but it spells the current word.



: strcmp  
: never directly called by the user

: StringContains  
: Checks to see if one string is contained in the other. The search starts at the beginning of the string to search in, the first parameter. If it finds the string it is searching for, the second parameter, then it returns true, otherwise it returns false.  
:Returns int 0=False, 1=True  
:param1 string The string to look in  
:param2 string The string to search for

: SubString  
: Extracts part of a string  
:param1 string The string to be processed  
:param2 int The index of the first character to be extracted. The index of the first character of the string is 1.  
:param3 int The number of characters to extract. If this is greater than the number of characters remaining in the string, it is silently rounded down.  
:returns string The extracted part of the original string

: ToggleDebug  
:Returns int  
: Toggles Debug mode on or off

: ToggleHomeRow  
: toggles the state of home-row mode. Home-Row mode is a keyboard shift-state, another layer for defining macros. Instead of Alt+F 1 you could use HomeRow+F1 for your macro key. It is a new level or set of macro keys, especially useful for the alphabet or home-row keys and users that want to keep their hands on the home-row. In this way, the "K" could be defined to read the current line when Home Row is "on", and when Home Row is toggled "off" it could become a "K" again.  
:Returns int 0 for off, 1 for on

: ToggleRestriction  
:Returns integer 1=on, 0=off  
: Toggles the JAWS cursor between restricted and unrestricted mode. Returns the new setting of the variable. 1 if restriction is now on, 0 otherwise. When restricted, the JAWS cursor will stay within the current window. The "window" could be a list box, edit field, or some other item.

: ToggleScreenEcho  
: cycles through the possible screen echo settings, to determine how much of the data being written to the screen is actually spoken: tr None, 1=Highlighted text only, 2=All. It



is used to control the Text event and Highlighted Text event macros, refer to them and the Insert+S macro for examples.

: ToggleVerbosity

: Changes the verbosity level, refer to the Insert+V macro for example.

: UnloadJAWS

: Stops JAWS and unloads it from memory, first warning you and prompting you to confirm. If you set up a HotKey in Program Manager then it is easy to re-load JAWS.

: While

: Start of While loop

:uminus

: never called by the user, here for testing only

