**BASICS OF SCRIPTING**

**MANUAL**

**BY**

**FREEDOM SCIENTIFIC**

**INTRODUCTION**

When used with most popular applications, JAWS® screen reading software provides you with screen information for the application you are using. The information that tells JAWS how to behave is contained within a set of “default files.” These default files, created by Freedom Scientific, are provided with JAWS. Unfortunately, not all applications are the same. Therefore, information telling JAWS how to behave in these applications is contained in a set of “application-specific files.” Through the many options available in the JAWS application, changes and adjustments can be easily made to all of the commonly used features, such as Verbosity and Screen Echo. But despite this ability, you may find special situations that require more advanced or specific instructions so that JAWS can properly navigate and provide the correct screen information in the application.

A script file contains a variety of scripts and functions. You can think of scripts and functions as small computer programs. Each script or function contains commands that tell JAWS how to navigate and what to speak under different conditions. You can modify existing script files or you can write entirely new commands to make most any application accessible with JAWS. Moreover, you can create scripts that automate many of your daily tasks. In short, your use of the scripting language is endless and is only limited by your imagination and creativity. JAWS is not the only product developed by Freedom Scientific that takes advantage of the powerful scripting language. Both the MAGic® screen magnification application, through the use of the MAGic Pro Scripting edition, and PAC Mate Omni® accessible Pocket PC device take advantage of the power and flexibility offered by the scripting language. With the familiarity gained from writing JAWS scripts, you will be able to easily apply your skills when generating scripts that control magnification and speech functionality found in MAGic. For those of you who ask, "Do I need a programming background to write scripts," the answer is that while a programming background is helpful, it is not necessary. In fact, many script developers working for Freedom Scientific are self-taught. It is for this reason alone that we have compiled this step-by-step manual in hopes that you will get a good start toward becoming a successful JAWS script developer.

**BASICS OF SCRIPTING TABLE OF CONTENTS**

2.0 The Keyboard Manager

3.0 What are Scripts?

4.0 The Script Manager

5.0 Creating Scripts

6.0 Reading Information with Scripts

7.0 Using Variables and Constants

8.0 Creating and Speaking Messages

9.0 Controlling the Flow of Scripts with Decision Making

10.0 Passing Keystrokes and Typing Text with Scripts

11.0 Using the Virtual Viewer

12.0 Understanding Functions

13.0 Windows Program Structure

Copyright (c) 2011 Freedom Scientific Inc. All Rights Reserved. JAWS is a registered trademark of Freedom Scientific Inc. in the United States and other countries. The names of actual companies and products mentioned herein may be the Trademarks (tm) of their respective owners in the U.S. and/or other countries.

July 2011

**OBJECTIVES**

After you have completed this manual, you will:

Know how to use Keyboard Manager to add, change, and remove keystrokes

Know the differences between scripts and functions

Understand how JAWS processes keystrokes

Describe the file types used by the Script Manager

Use keystrokes to review scripts contained within a given script file

Use the Script Manager to create and modify scripts

Use the New Script dialog box to create scripts

Use the Insert Function dialog box to add functions to your scripts

Name the parts of a script

Manipulate various cursors using scripts

Know the different variable types used in the scripting language

Know the differences between variables and constants

Speak messages using scripts

Use decision making in scripts

Understand Windows program structure and hierarchy

Be able to identify application windows using unique identifiers

Use the script utility mode to analyze programs before scripting

**DEFINITION OF TERMS**

We will be using a variety of new terms in this manual to describe scripts and functions along with their uses. Terms are listed below along with their meaning. You will find more detailed information about these terms as they are introduced in this manual.

Built-in Function

Built-in functions are functions created by the developers at Freedom Scientific. They are used as building blocks for your scripts and functions. Although built-in functions can be used anywhere, they cannot be modified.

Call

The term 'call' is used whenever a function is performed from within a script or another function.

Compile

The term 'compile' refers to the action of converting the scripts contained within your script file into a language that can be understood by your computer.

Constant

A 'constant' is a name you give to a hard-to-remember numeric value or sequence of characters. Constants are not changed during the execution of your script or function and are typically defined in a JAWS script header file. You use constants in place of hard-to-remember values in your scripts and functions.

Event Function

Event functions have been created by developers at Freedom Scientific and are triggered by various Windows events. Examples of events that trigger event functions are when the system focus changes or when new text is displayed on the screen. Event functions can be modified or customized for your own use but you cannot create new event functions. You can find all the event functions in the Default.jss script file.

Function

A 'function' is also a series of statements that performs a given task. Functions are not activated by a keystroke. Functions are called by scripts or other functions.

Key Words

Key words' are words that are reserved for use by the scripting language. Some of the key words used in the scripting language are Script, EndScript, If, EndIf, While, and EndWhile. See Appendix A: Key Words for a complete list of the key words used in the scripting language.

Parameter

A 'parameter' is a piece of information that is passed to a function. The function uses the parameter to process its task.

Return

A 'return' is a piece of information a function sends back to a script or function from which it is called.

Script

A 'script' is a series of statements that performs a given task. The script is normally activated by a keystroke.

Script File

A 'script file' is a collection of scripts, functions, and variable declarations with the file extension of .jss for JAWS script source.

Statement

A 'statement' is a series of words used within your script that performs an action or group of actions. For example, the following block of code is considered to be a statement:

If A > B Then

User-defined Function

User-defined functions are functions created by individual script writers. User-defined functions can be modified and you can create as many as you need. You will find many examples of user-defined functions in the default JAWS script files.

Variable

A 'variable' is an entity that holds a value. Unlike a constant, the value of a variable can be modified by your script or function during its execution.

Window

A predefined rectangular area on the screen that can be identified by unique characteristics. Windows usually contain text, graphics, or various types of controls used by an application.

**2.0 THE KEYBOARD MANAGER**

You control the assignment of keystrokes to JAWS commands using the Keyboard Manager. The Keyboard Manager stores keystroke assignments in key map files, which have the extension of .jkm. You usually assign keystrokes to scripts at the time the script is created using the Script Manager. You can use the Keyboard Manager to view, add, change, or delete keystroke assignments. The Keyboard Manager has a screen layout that consists of a left hand pane and a right hand pane. Press TAB and SHIFT+TAB to move between the two panes. Press UP and DOWN ARROW to move up and down either side of the manager. For a list of all keyboard commands available in the Keyboard Manager, see Appendix B: Keyboard Manager Commands.

**TABLE OF CONTENTS**

Chapter 2.0 The Keyboard Manager, contains the following sections:

2.1 Starting the Keyboard Manager

2.2 Keyboard Manager Files

2.3 Keyboard Manager, Left Hand Pane

2.4 Keyboard Manager, Right Hand Pane

2.5 Finding Keystrokes

2.6 Reviewing Scripting Documentation

2.7 Adding, Changing, and Removing Keystrokes

2.8 Keyboard Manager Options

2.9 Chapter Exercises

**2.1 STARTING THE KEYBOARD MANAGER**

Press INSERT+F2 to open the Run JAWS Manager dialog, then press K, followed by ENTER. When you start the Keyboard Manager from within an application, the Keyboard Manager automatically opens the key map file for that application. When you start the Keyboard Manager from the Utilities menu within the JAWS interface, the Keyboard Manager opens the default key map file. After you start the Keyboard Manager, JAWS selects the appropriate key map file name in the left hand pane of the manager5. When you open the Keyboard Manager from an application, you can open the default key map file by pressing CTRL+SHIFT+D.

**2.2 KEYBOARD MANAGER FILES**

The Keyboard Manager uses two types of key map files:

Default

Application All key map files have the file extension of .jkm. The 'default' key map file contains the keystroke assignments for the default script file. JAWS loads the default key map file into memory each time you start JAWS. The default set of keystroke assignments are always available for your use. The 'application' key map files contain keystroke assignments for scripts in an application script file. All application key map files have the same name as the application's script file. For example, the script file name for Microsoft® Word 2007 is Microsoft Word 2007.jss. Thus, the key map file for Word is Microsoft Word.jkm. See Naming JAWS Settings Files for more information on how JAWS derives the names of key map files. When you have an application plus the JAWS program open on your computer, JAWS employs both the associated application key map file and the default key map file so that both sets of keystroke assignments are available. For those applications where the application and default key map files share the same keystroke assignment, the keystroke assignment in the application key map file always takes precedence over the keystroke assignment in the default key map file.

**2.3 KEYBOARD MANAGER, LEFT HAND PANE**

The left hand pane of the Keyboard Manager contains an alphabetical list of all the current application key map files plus the default key map file. You can use your UP and DOWN ARROW keys to move through the list of files. You can also type a letter to select the first file in the list that begins with that letter. Each time you type the letter, the next file beginning with that letter is selected. You can continue this process until no file names match the letter you are typing. The Keyboard Manager displays the contents of the currently selected file in the right hand pane of the manager. When you find the desired key map file, press TAB to move to the right hand pane and review the information contained within that file.

**2.4 KEYBOARD MANAGER, RIGHT HAND PANE**

The right hand pane of the Keyboard Manager presents current key map file information in four columns. The columns of information are:

Script name

Keystroke

Key map file name

Key map section

You can use your UP and DOWN ARROW keys to move through the columns of information. As in the left hand pane, you can also use first letter navigation to move to the first script name that begins with the letter you have typed.

Script Name The first column of information contains the script name. When you press the associated keystroke, this is the script that is activated. The actual script is contained within the corresponding script file with the same name as the selected key map file whose contents you are reviewing. Examples of script names include AdjustJAWSOptions, SaySystemTime, and SayCharacter.

Keystroke. The second column of information contains the keystroke assignment that activates the script when the command is pressed. As you move up or down the list of information, you may notice some script names that do not have keystroke assignments. In these cases, the Keyboard Manager displays "none." This indicates that either a keystroke assignment was never made, the script is used in conjunction with a Braille display and you do not have a Braille display, or the script was left over from a previous version of JAWS and is no longer used. Examples of keystrokes include: JAWS Key+V, JAWS Key+F12, and JAWS Key+NUM PAD 5.

Key Map File Name.

The third column of information contains the name of the key map file in which the keystroke assignment is stored. When a keystroke assignment is located in the default key map file, then the Keyboard Manager displays "Default" in this column. The Keyboard Manager displays the application key map file name if the keystroke assignment is specific to that application.

Key Map Section Name. The fourth and final column of information contains the key map section. This is the section of the key map file in which the keystroke assignment is stored. When this column displays "common," then the keystroke assignment is available in both the desktop and laptop keyboard layouts. In cases where this column displays "desktop," then the keystroke assignment is only available when you are using the desktop keyboard layout. When this column displays "laptop," then the keystroke is only available when you are using the laptop keyboard layout. See 2.7 Adding Keystrokes for more information on assigning keystrokes to specific keyboard layouts.

**2.5 FINDING KEYSTROKES**

You can use the Keyboard Manager to search for existing keystrokes. Before you write a script, you can search the application key map file to determine if the keystroke you have chosen for the new script is already assigned. If the application for which you are customizing JAWS does not have a key map file, then you can be sure the keystroke does not exist in the application key map file. However, the keystroke may exist in the default file so it is a good idea to check there as well. The right hand pane of the manager must be active to search for a specific keystroke in the current key map file. You can either choose Find Keystroke… in the Action menu or press CTRL+F to display the Find Keystroke dialog. When the Keyboard Manager displays the Find Keystroke dialog, the Search for Key edit box is active. Press the keystroke for which you are searching. JAWS echoes the keystroke after you press it. Once you have entered the keystroke, press TAB to move to the group of radio buttons that allows you to search in the active or default key map file. If you do not change this option, the Keyboard Manager carries out the search in the active file. Press ENTER on the set of radio buttons or activate the Ok button by pressing SPACEBAR to begin the search. When the Keyboard Manager finds the keystroke, the script name associated with the keystroke is selected. JAWS automatically speaks the highlighted information. The Keyboard Manager displays the Keystroke Not Found dialog when the keystroke is not found.

**2.6 REVIEWING SCRIPT DOCUMENTATION**

As you browse the keystroke assignments in a key map file, you can view the script documentation for the script assigned to a given keystroke. The script documentation consists of the Synopsis (brief description of the script's purpose) and the Description (additional information about the script). See 5.2 The New Script Dialog for more information on adding script documentation. You can view the documentation for the current script by choosing Documentation from the Action menu or by pressing CTRL+D. When the Script Documentation dialog opens, the script name edit box is active. Press TAB and SHIFT+TAB to move among the edit boxes contained within this dialog. You can press ESC to close the dialog when you are finished reviewing the script information.

**2.7 ADDING, CHANGING, AND REMOVING KEYSTROKES**

After you locate the desired script, you can use the Action menu to Add, Change, or Remove the keystroke assigned to this script. Freedom Scientific does not recommend changing keystroke assignments within key map files provided with JAWS. You should give careful consideration to the keystroke assignment before you change the keystroke. You should try to add a new keystroke instead of changing an existing keystroke assignment. You can add a new keystroke from the Action menu. When you add a keystroke, the Keyboard Manager does not replace the existing keystroke assignment. Rather, the Keyboard Manager creates an additional keystroke assignment for the same script. If more than one keystroke is assigned to a script, the script is listed once for each keystroke in the key map file. When you delete a keystroke assignment from the key map file, the Keyboard Manager does not delete the associated script from the corresponding script file. Only the keystroke assignment is deleted.

Adding Keystrokes

You can add a keystroke to the key map file in one of two ways. You can choose Add Keystroke… from the Action menu or press CTRL+A to display the add Keystroke dialog. When the Keyboard Manager displays the Add Keystroke dialog, the Assign To edit box is active. This edit box behaves exactly as the edit box used in the Find Keystroke dialog. Press the desired keystroke and the Keyboard Manager interprets this as the new keystroke you want to add. If you press a keystroke that is already in use, the Keyboard Manager displays a warning dialog box advising you that the keystroke is already in use. The warning dialog box also displays the key map file in which the keystroke is assigned. At this point, you should choose cancel and try another keystroke. After you have pressed the new keystroke, press TAB to move to the assign Key To checkbox. The Keyboard Manager determines the label for this checkbox based on the keyboard layout you are using. This check box is checked by default. If you leave this checkbox checked, then the keystroke assignment is active for the current keyboard layout only. If you want to allow the keystroke assignment to be used within all keyboard layouts, then it is best to press SPACEBAR to clear this checkbox. For example, when you are using the desktop keyboard layout the Keyboard Manager displays the label of Assign Key to DESKTOP Keys only. When you leave this checkbox checked, the keystroke is only available when the desktop keyboard layout is in use. When you assign a keystroke to a script using a keystroke combination that uses a key from the number pad, then the new keystroke is only available when the desktop keyboard layout is in use. Laptops typically do not have number pads. If you change to the laptop keyboard layout, the keystroke is no longer available. When you clear this checkbox, then the keystroke assignment is available in all keyboard layouts. This checkbox determines what is displayed in the Key Map Section column of information shown in the right hand pane of the Keyboard Manager. After you have determined the keyboard layout for which the keystroke will be available, press TAB to move to the Ok button. When you press SPACEBAR to activate the button, the Keyboard Manager displays the Confirm Add Keystroke dialog. When you press SPACEBAR to activate the Yes button, the Keyboard Manager adds your keystroke to the key map file. After the Keyboard Manager adds the keystroke information to the current key map file, the right hand pane of the manager is active.

Note: After you have added a keystroke, you do not need to save the current key map file. The Keyboard Manager saves the file after the new keystroke is written to the key map file.

**Changing Keystrokes**

As stated previously, you should use care when changing existing keystrokes. The only exception to the rule of changing keystroke assignments involves scripts that you have written. In this case, your scripts are not provided with JAWS and changing the assigned keystroke has no impact on the operation of JAWS. To change a keystroke assignment, first locate the desired script. You can then choose Change Keystroke… from the Action menu or press CTRL+H to display the Change Keystroke dialog. Like the add Keystroke dialog discussed above, the Assign To edit box is active when the Keyboard Manager displays the change Keystroke dialog. Like the Assign To edit field in the Find Keystroke and Add Keystroke dialogs, this edit box accepts any keystroke you press. So do not try to use the say line command, or INSERT+UP ARROW, to reread the contents of this field. JAWS interprets any keystroke you press as the keystroke you want to add. When you press a keystroke that is already assigned, the Keyboard Manager displays the Keystroke in the Use dialog. You can choose to add the keystroke anyway, or press SPACEBAR to activate the Cancel button and clear the dialog.

After you have pressed the desired keystroke, press TAB to move to the Assign Key To checkbox. Again, the Keyboard Manager determines the label for this checkbox based on the keyboard layout currently in use. If you leave this checkbox checked, then the keystroke assignment is active for the current keyboard layout only. If you want to allow the keystroke assignment to be used within all keyboard layouts, then it is best to uncheck this checkbox. When you press TAB from the Assign Key To checkbox, the OK button becomes active. You can activate this button with SPACEBAR. The Keyboard Manager displays the Confirm Change Keystroke dialog. The default button in this dialog is the Yes button. Press SPACEBAR to activate the button and confirm the keystroke change.

**Removing Keystrokes**

If you have previously added a new keystroke to a key map file, you can remove that keystroke as well. To remove a keystroke, select the keystroke and choose the Remove Keystroke option from the Action menu or press DELETE. When you initiate the remove keystroke action, the Keyboard Manager displays the Confirm Remove Keystroke dialog. The default button in this dialog is the Yes button. If you want to remove the keystroke, press SPACEBAR to activate the Yes button, and confirm the removal of the keystroke. Important: You only remove the keystroke assignment from the key map file, not the attached script. If you are removing a keystroke assignment you have added previously, then the entire entry is removed from the key map file. If the keystroke assignment you are removing is an existing keystroke assignment that came with JAWS, then you will see the word "none" placed in the keystroke column in the right hand pane of the Keyboard Manager.

**2.8 KEYBOARD MANAGER OPTIONS**

You can use the Options menu to change the way information in the right hand pane of the Keyboard Manager is displayed. You can sort the information based on a different column, filter out different types of keystrokes, turn off the confirmation messages, and assign the TAB, SHIFT+TAB, ENTER, and ESC keys to scripts. When you access any of the items found within the Options menu, the Keyboard Manager displays the Options multi-page dialog. The active page corresponds to the option you selected. Press CTRL+TAB to move among the pages. The five pages contained within the dialog going from left to right are:

Key Filter

Messages

Sort

Hot Key

File Filter

Key Filter Options

You can use the Key Filter page to determine what keystrokes are displayed within the right hand pane of the Keyboard Manager. You can choose to display all, active, or application keystrokes only. You can also choose to display only those scripts that have keystrokes assigned, or those that do not. This page contains two sets of radio buttons. You can use the first set of radio buttons to filter keystroke information based on the actual keystroke. You can use the second group of radio buttons to filter out information based on the script to which the keystroke is assigned. The following sections discuss these options.

All Keystrokes. Use this option when you want to display all keystrokes from both the application and default key map files. When you choose this option, the contents of the key map file name column will contain either the name of the application-specific or default key map file name. This option allows you to see all keystrokes available within a given application.

Active Keystrokes. Use this option when you want to display only active keystroke assignments in both the application specific and the default key map files. If a keystroke is assigned in both the application and default key map files, only the application keystroke is active. JAWS always acts on the first keystroke it finds and it looks in the application key map file first. This setting filters out duplicate keystrokes in the default key map file. For example, when you have a keystroke such as CTRL+G defined in both the application-specific and default key map files, the Keyboard Manager displays only the keystroke assignment in the application-specific file. For more information on application-specific keystrokes versus default keystrokes, see 3.2 Processing Keystrokes.

Application Keystrokes. Use this option when you want to display only keystrokes contained within the selected key map file. This is the default key filter option. For example, when you open the Keyboard Manager from within Microsoft Word 2007, the Keyboard Manager opens the Microsoft Word 2007.jkm key map file. This is the key map file for Word. The Keyboard Manager displays only those keystrokes that are specific to Microsoft Word in the right hand pane.

All Scripts. Use this option when you want to display all scripts contained within the selected key map file. This is the default setting. This option lists all scripts regardless of whether they have a keystroke assigned to them.

Scripts Assigned to Keys.

Use this option when you want to display only scripts that are assigned to keystrokes. This option keeps any scripts with the word "none" shown in the keystroke column from being displayed. For example, when you are viewing the default key map file and you have not installed a Braille display, choosing this option keeps the keystrokes used in conjunction with a Braille display from being displayed. Scripts without keystroke assignments are also not displayed.

Messages Options. You can use the Messages page of the Options multipage dialog to turn off the notification messages displayed each time you make a change to the selected key map file. By default, the Keyboard Manager displays these messages each time you add, change, or remove a keystroke from the selected key map file. This page contains only a single checkbox. You can clear this check box by pressing SPACEBAR.

Sorting Options. You can use the Sort page of the multi-page Options dialog to determine the column for which information is to be sorted. By default, the Keyboard Manager displays all keystroke information sorted by the script name. This page contains a single group of radio buttons that you can use to change the sort option. Use your arrow keys to select the column that you would like to sort by.

**HotKey Options**

You can use the HotKey page of the Options multipage dialog to assign the TAB, SHIFT+TAB, ESC, and ENTER keys to scripts contained within the selected key map file. Each hot key has a check box, that when checked, allows you to assign the key to a script. Once you have assigned one of these hot keys to a script, the keystroke no longer performs its normal function. For example, when you check the TAB check box in the hot key page of the Options multipage dialog and assign it to a script, you can no longer use the TAB key to move from control to control within a dialog box. You should only use these keys as a last resort when all options for key assignments have been exhausted. o assign one of these hot keys to a script, you must first check the appropriate check box for the desired key. Next, you can close the dialog by either pressing SPACEBAR on the OK button or by pressing ENTER on the check box for the desired key. After you have closed the Keyboard Manager Options multi page dialog, select the script or scripts for which the hot key will be assigned. Choose either the Add Keystroke or Change Keystroke item from the Action menu. Press the hot key when prompted by the dialog. After you confirm the modification to the selected key map file, your keystroke becomes active.

**File Filter Options**

You can use the File Filter page of the Options multipage dialog to indicate which set of key map files JAWS should display in the Keyboard Manager. This page contains a single group of radio buttons that you can use to change the file filter option. Use your arrow keys to select the type of key map files you want to display in the Keyboard Manager. You can choose from active, user only or shared files. The following sections discuss these options

Active Files. Use this option to display all key map files currently in use. If both a shared and user version of the same key map file exist, only the user-specific file is displayed. This is the default file filter option.

User Files. Use this option to display only those key map files that reside in the user folder on your system. These keystrokes override those in the shared key map files.

Shared Files. Use this option to display only those key map files that reside in the shared folder on your system.

**2.9 CHAPTER EXERCISES**

The following exercises give you practice in using many of the functions of the Keyboard Manager. The objective of the exercise is listed first.

**Exercise 2.1: Finding a Keystroke**

The objective of this exercise is to find a specific keystroke within the default key map file and view the corresponding script documentation.

Press INSERT+F2 to display the Run JAWS Managers dialog.

Type K to select the Keyboard Manager option followed by ENTER. This action starts the Keyboard Manager.

Press INSERT+UP ARROW to read the current line. This is the current key map file. Did the selected key map file match the active application?

Press CTRL+SHIFT+D to select the default key map file. The contents of the file are shown in the right hand pane of the Keyboard Manager.

Press TAB to move to the right hand pane of the manager. How many items does JAWS tell you are contained in the list?

Press CTRL+F to activate the Find Keystroke dialog.

Press INSERT+F12. JAWS speaks the keystroke after you press it as JAWS KEY+F12. The keystroke is also shown in the Search for Key edit box.

Press ENTER to begin the search. Did JAWS find the keystroke? Did JAWS read the script name, keystroke, key map file name and section? If you did not hear this information, press INSERT+UP ARROW to read the current line and repeat the information.

Press CTRL+D to activate and display the Script Information dialog. Press TAB and SHIFT+TAB to move to and read the edit boxes contained in the dialog.

Press ESCAPE to close the dialog after you have reviewed the script information.

**Exercise 2.2: Adding a Keystroke**

The objective of this exercise is to locate a specific keystroke in the default key map file and add a second keystroke to the script attached to that keystroke. If you already have the Keyboard Manager running, you can move to it by pressing ALT+TAB. If the Keyboard Manager is not running, then follow steps 1 through 3 in exercise 2.1 to start the Keyboard Manager and open the default key map file.

Press CTRL+F to activate the Find Keystroke dialog.

Press INSERT+F11. This is the keystroke for which we want to search. JAWS echoes the keystroke as you press it and JAWS KEY+F111 is placed in the Search for Key edit box.

Press ENTER to begin the search. When JAWS finds the keystroke, the information is selected and spoken automatically by JAWS.

Note: If the keystroke was not found, then the Keystroke Not Found dialog is displayed.

Press CTRL+A to activate the Add Keystroke dialog. The Assign To edit box is active.

Press CTRL+F11. JAWS echoes the keystroke and CTRL+F11 is placed in the Assign To edit box.

Press TAB to move to the Assign To X Keys Only check box where X represents the keyboard layout in use. If you are using the desktop layout then this is the Assign Key To Desktop Keys Only checkbox.

Press SPACEBAR to uncheck the Assign Keys to X check box.

Press ENTER to activate the Ok button in the Add Keystroke dialog. Since the default button is the Ok button, the dialog is closed. Next, the Confirm Add Keystroke dialog is displayed.

Press SPACEBAR to activate the Yes button. The Confirm Add Keystroke dialog is closed and you are returned to the Keyboard Manager.

Press INSERT+UP ARROW to read the current line. Did you hear the original keystroke information for INSERT+F11 spoken by JAWS?

Press DOWN ARROW to move to the next entry in the list. Did you hear the new keystroke echoed by JAWS? If you want to repeat the information, press INSERT+UP ARROW to read the current line. After you have completed the steps above, be sure to test both new and old keystrokes. Do they both work?

**Exercise 2.3: Changing an Existing Keystroke**

The objective of this exercise is to locate the keystroke we just added in the previous exercise and change it. You should still have the Keyboard Manager running with the default key map file open. If you closed the Keyboard Manager, follow steps 1 through 3 from exercise 2.1 to activate the Keyboard Manager and open the default key map file.

Press CTRL+F to activate the Find Keystroke dialog. The focus is in the Search for Key edit box.

Press CTRL+F11. This is the keystroke for which the search is carried out. JAWS speaks the keystroke and CTRL+F11 is displayed in the Search for Key edit box.

Press ENTER to activate the Ok button. The find keystroke action is performed. When the keystroke is found, the keystroke information is spoken automatically by JAWS.

Press CTRL+H to activate the Change Keystroke dialog. The Assign To edit box is active in this dialog.

Press SHIFT+F11. This is the new keystroke. JAWS echoes the keystroke and SHIFT+F11 is placed in the assign To edit box.

Press TAB to move to the Assign to X Keys Only check box, where X represents the keyboard layout in use. . If you are using the desktop layout, then this check box is labeled as Assign to Desktop Keys Only.

Press SPACEBAR to uncheck the Assign to Desktop Keys Only checkbox.

Press ENTER to activate the Ok button. The Change Keystroke dialog closes. The Confirm Change Keystroke dialog box is then displayed and the Yes button is active.

Press SPACEBAR to activate the Yes button. The Confirm Change Keystroke dialog closes and you are returned to the Keyboard Manager.

Press INSERT+UP ARROW to read the current line. What keystroke information was spoken by JAWS?

Since the current keystroke entry is not the changed entry, press UP ARROW to move to the prior entry. Is this your new keystroke? After you have followed the steps above, be sure to test both the old and new keystrokes. Do they both work as expected?

**Exercise 2.4: Removing a Keystroke**

The objective of this exercise is to find the keystroke we changed in the previous exercise and remove it. This will leave only the original keystroke to list the items in the system tray. If the Keyboard Manager is not running already refer to steps 1 through 3 in exercise 2.1 to start the Keyboard Manager and open the default key map file.

Press CTRL+F to activate the Find Keystroke dialog. The Search for key edit box is active.

Press SHIFT+F11. This is the keystroke for which the search is carried out. JAWS echoes the keystroke and SHIFT+F11 is placed in the Search for Key edit box.

Press ENTER to perform the Find Keystroke action. When the keystroke is found, the information is highlighted and spoken automatically by JAWS.

Press DELETE to remove the keystroke. The Remove Keystroke dialog is displayed and the Yes button is active.

Press SPACEBAR to activate the Yes button and close the Confirm Remove Keystroke dialog. The keystroke is removed from the default key map file.

Press INSERT+UP ARROW to read the current line. Is the SHIFT+F11 keystroke gone? Is the default keystroke, INSERT+F11, the only keystroke displayed?

**3.0 WHAT ARE SCRIPTS?**

You can think of a script as a mini computer program that contains sequences of individual steps used to activate and control a wide variety of computer processes. The action a script performs ranges from tasks as simple as entering repetitive strings of data to many of the standard routines your computer does on an ongoing basis as part of its operation. Scripts or combinations of scripts and functions control most JAWS behavior. Scripts analyze what actions are taking place in an application and determine how to handle the screen reading for you.

Script files also contain functions. Like a script, a function is also a sequence of statements that perform a given task. However, functions differ from scripts in that you cannot activate them with a keystroke.

You create scripts and functions with the Script Manager. You use a standard set of programming codes, formats, and rules to create your scripts and functions. After you have written your script or function, you compile your script or function into machine-readable language, so that the computer can understand the instructions. Once you compile your script or function, they are ready for your use.

**Table of Contents**

Chapter 3.0 What are Scripts?, contains the following sections:

3.1 JAWS Scripts and Script Files

3.2 Processing Keystrokes

3.3 Performing Functions

3.4 Scripts vs. Functions

3.5 Chapter Exercises

**3.1 JAWS SCRIPTS AND SCRIPT FILES**

A script file is a file that contains multiple scripts and functions. You can find a number of script files in the JAWS shared settings folder on your system. These script files all have the extension of .jss for JAWS script source. For example, the Default.jss script file contains all the scripts and functions JAWS uses to interface with Windows based applications. JAWS also uses application-specific script files. These files contain all the scripts and functions needed to access information in a given application. JAWS loads the application-specific script file on top of the default script file when you start the corresponding application.

The Default Script File. JAWS loads the default script file each time it starts. The scripts and functions contained in the default file are active in all sessions. JAWS receives all the information it needs to provide proper voice output from the scripts contained in the default script file. The scripts within this file tell JAWS what to speak and when to speak it in most circumstances. The default script file is named Default.jss.

Application-specific Script Files.

the second type of script file JAWS uses is specific to a Windows based application. You create an application script file when you encounter situations that require customized reading of the screen. JAWS loads the application-specific script file when the application is started or becomes active. For example, an application named SPREADSHEET.EXE has a script file called SPREADSHEET.JSS. The application script file must exist in either the JAWS shared or user settings folder in order to be loaded with the application at run-time. When the spreadsheet application is no longer active, JAWS unloads the spreadsheet's script file. All of the default scripts then become active until JAWS loads another application's script file.

**3.2 PROCESSING KEYSTROKES**

Each time you press a keystroke, JAWS goes through a series of steps to determine what action to perform. Even though your scripts may have different names, it is the keystroke binding that determines what script is performed. JAWS must first determine where the keystroke is assigned.

**Application Key Map**

JAWS first looks for the keystroke assignment within the application-specific keymap file. If JAWS finds the keystroke assignment in this file, then the name of the script associated with the keystroke is remembered.

Application Script File. JAWS then looks for the name of the script in the Application-specific Script file. If JAWS finds the script, then JAWS performs the script and no further processing is done.

**Default Key Map**

If JAWS does not find the keystroke assignment in the application-specific key map file, then the default key map file is searched. If JAWS finds the keystroke assignment in this file, JAWS notes the name of the script activated by the keystroke. Before JAWS executes the script, JAWS must determine if the script is contained in the application-specific script file. If JAWS does not find the keystroke in the application or default key map files, then the keystroke is passed through to the application just as if JAWS were not running.

Application-specific Script File. JAWS searches the application-specific script file for the script name found in the default key map file. If JAWS finds the script, then JAWS performs the script and no further processing is necessary. If JAWS does not find the script within the application-specific script file then JAWS searches the default file. If JAWS finds the script in the default file, then JAWS performs the script and no further processing is required.

Unknown Script Call.

When JAWS does not find the name of the script attached to the keystroke in either the application-specific or default script files, then an unknown script call error message occurs. JAWS speaks “unknown script call to” followed by the name of the script. An unknown script call can occur when either the script has been deleted from the application-specific or default script file or when the name of the script is misspelled in the key map or script file. To keep these types of errors from occurring, you should use the Script Manager to remove a script and its associated key map and documentation entries.

**Pass to Application**

If JAWS does not find the script in either file, then JAWS passes the keystroke through to the application. The application performs the keystroke just as if JAWS were not running. JAWS always performs a script in the application script file rather than a script found in the default script file even when you make the key assignment in the default key map file.

Assigning Keystrokes. You can assign a keystroke to a customized script in an application-specific script file even though it is assigned in the default key map file. In doing so, you will not encounter any adverse consequences. When you use a keystroke that is already defined in the default key map file, the default functionality is lost for only that application. For example, when you assign CTRL+INSERT+V to a script contained in the Notepad script file, the default script, SayAppVersion, is no longer performed. JAWS recognizes the fact that you have used the default keystroke for an application-specific script. So, JAWS does not perform the SayAppVersion script when you are using Notepad. If you give the script in the application-specific file the same name as the script in the default script file, you do not need to assign a keystroke to the application-specific script. JAWS performs the script in the application-specific script file instead of the script found in the default file if they are both bound to the same keystroke. For example, you can find the AdjustJAWSOptions script in both the default script file and the Internet Explorer script file. The keystroke is assigned only in the default key map file. When you press INSERT+V while you are in Internet Explorer, JAWS finds and performs the application-specific version of AdjustJAWSOptions. However, when you are using Notepad and press INSERT+V, JAWS performs the AdjustJAWSOptions script from the default script file.

Note: If you attempt to assign a script to a keystroke which is already in use by the default file, then JAWS displays a warning dialog. This warning dialog gives you the opportunity to continue with the assignment or change it to another key. This is to prevent you from accidentally disabling a script in the default file without realizing you are doing so.

**3.3 PERFORMING FUNCTIONS**

Unlike scripts, JAWS performs functions based on the name and location of the function. Each time a function is called from within a script or another function, JAWS searches the application-specific script file first. If the function is contained within the application-specific script file, then JAWS performs the function and no further processing is done. If JAWS does not find the function in the application-specific file, then JAWS searches the default file. If JAWS finds the function in the default file, then JAWS performs the function and no further processing is done. When JAWS does not find the function in the default file, then JAWS searches the list of built-in functions. If JAWS finds the function, then JAWS performs the function and no further processing is done. When JAWS does not find the function, then you will hear JAWS speak "unknown function call to" followed by the name of the function. JAWS will then spell the name of the function. This message tells you that the function JAWS tried to call may have a name that is misspelled, the function was previously deleted or has not been written. An example of how JAWS processes functions follows:

You create a custom version of the SayHighlightedText function in an application-specific script file. You created this custom version of the SayHighlightedText function because processing of highlighted text is handled differently in the application. The SayHighlightedText function also exists in the default script file. JAWS performs the SayHighlightedText function found in the application-specific script file rather than the SayHighlightedText function found in the default script file.

**3.4 Scripts vs. Functions**

To this point, we have been mainly discussing scripts. However, there is a second block of code found in a script file. This second block of code is called a function. Functions differ from key-bound scripts in that they can return information after they finish executing. A function can also accept data from the calling script or function. The data passed to a function is called a parameter and the data returned to the calling script or function is called a return. There are three types of functions:

Event

Built-in

User-Defined

**Event Functions**

JAWS performs event functions automatically when windows events occur. Without these event functions, JAWS does not speak system changes automatically. You cannot create new event functions. However, you can modify existing event functions. You can also create application-specific versions of the event functions found in the default script file.

Some of the event functions within JAWS include:

AutoStartEvent

BottomEdgeEvent

FocusChangedEvent

NewTextEvent.

JAWS performs the AutoStartEvent function each time an application is started or becomes active. JAWS performs the BottomEdgeEvent each time the active cursor reaches the bottom edge of a window. JAWS performs the FocusChangedEvent function when focus moves between applications, dialogs, and dialog controls. JAWS performs the NewTextEvent function each time any new text is written to the screen.

**Built-in Functions**

There are over 1300 built-in functions that are available for use in creating new or modifying existing JAWS scripts. You can think of the built-in functions as the building blocks of your scripts and functions. Some built-in functions include: SayLine, JawsCursor, NextWord, and PriorWord. You cannot modify existing built-in functions. Built-in functions are performed when they are called by another script or function. These functions are performed by placing the name of the function within the coded lines of another script or function. When the script gets to the line that includes the name of the function, the function is performed. This process is known as "calling" a function.

**User-defined Functions**

You can find user-defined functions in script files. These functions are developed by either Freedom Scientific script developers or by a script developer outside of Freedom Scientific. These user-defined functions have been created for some special purpose and can also be modified. Like built-in functions, user-defined functions are called from a script or another function.

**3.5 CHAPTER EXERCISES**

Exercise 3.1: Performing Scripts

The objective of this exercise is to show how a single keystroke can activate either an application-specific script, or a script found in the default script file.

Press WINDOWS KEY+M to minimize all of your running applications.

Start Notepad from the start menu. You can find the Notepad application in your Programs | Accessories group within your Start Menu.

After Notepad starts make sure it is the active application by pressing INSERT+T. You should hear JAWS speak "Untitled - Notepad".

Press INSERT+V to activate the Adjust JAWS Options dialog. Note how many items are found within this dialog. How many are there? What was the first item in the tree view?

Press ESC to close the Adjust JAWS Options dialog and return to Notepad.

Press WINDOWS KEY+M to minimize all of your running applications and make the Desktop active.

Press I until you locate the Internet Explorer icon on your desktop. When you find the icon, press ENTER to start Internet Explorer.

Press INSERT+T to read the title of the active window. Internet Explorer should be the active application.

Press INSERT+V to display the Adjust JAWS Options dialog. How many items are in this tree view? Is the tree view different from the tree view you saw while in Notepad? Why?

**4.0 THE SCRIPT MANAGER**

The Script Manager provides you with all the tools you need to write, modify, and use scripts. The Script Manager is a full-featured text editor. In addition to the usual features contained in most text editors, it has built in capabilities to help you create scripts. You can insert functions into your new scripts, display reference information about functions and scripts that have already been written, check for errors that violate the rules of correct script writing, and save and compile your script files. In addition to providing full scripting capabilities for JAWS, the Script Manager can also be used to create scripts for the MAGic Pro Scripting Edition. The Script Manager provides hot keys you can use to read through the scripts in your script file more efficiently. For example, a script file can contain one or more scripts. You can use the function key F2 to move forward from script to script, or SHIFT+F2 to move backward from script to script. You can use the hot key combination, CTRL+L, to display a list of all scripts in the current file in alphabetic order. When you save your script file in the Script Manager with the keystroke, CTRL+S, the Script Manager automatically performs an additional step not found in other text editors and compiles your script file. Compiling a script converts the file from text to machine-readable information.

The Script Manager saves the original text file with a .jss (JAWS Script Source) file extension, and the compiled or binary version (machine-readable language) is saved with a .jsb (JAWS Script Binary) file extension. This is why it is always important to create and save your script files using the Script Manager. If you edit and save your files using some other text editor such as Notepad, they are not compiled when you save them, and JAWS will be unaware of the changes you have made. For a list of all keyboard commands available in the Script Manager, see Appendix C: Script Manager Commands.

**Table of Contents**

Chapter 4.0 The Script Manager, contains the following sections:

4.1 Script Manager Files

4.2 Starting the Script Manager

4.3 Navigating Script Files

4.4 Finding and Replacing Text

4.5 Viewing Information

4.6 Chapter Exercises

**4.1 SCRIPT MANAGER FILES**

The Script Manager uses five different files, some of which are created automatically. All five files are found within your JAWS settings folder. See Naming JAWS Settings Files for more information on how JAWS derives the names of files created automatically.

JSS JAWS Script Source File

JSD JAWS Script Documentation File

JSH JAWS Script Header File

JSM JAWS Script Message File

JSB JAWS Script Binary File

**The JAWS Script Source File**

The first file the Script Manager uses is the JAWS script source file. The script source file is the file that contains all the text of the scripts and functions you create. This file is essentially a text file with the file extension of jss. When you start the Script Manager from an application that is not supported by Freedom Scientific, the Script Manager automatically creates this file for the active application. For example when you run the Script Manager from within Notepad, the Script Manager creates a new script file. The manager creates the file because Notepad is not supported by Freedom Scientific. In other words, the script developers at Freedom Scientific have not written any scripts or user-defined functions for Notepad. See section 4.2 for more information on starting the Script Manager.

**The JAWS Script Documentation File**

The second file the Script Manager uses is the JAWS script documentation file. This file contains the name, synopsis, description, return value, and parameters for all of the scripts and user-defined functions in the corresponding script file. Like the script source file, this file is also a text file with an extension of jsd. When you activate keyboard help, the information JAWS speaks as you press keystrokes is retrieved from this file. When you press a keystroke once, JAWS speaks the synopsis of the script. When you press the same keystroke twice in succession, JAWS speaks the description.

The JAWS Script Header and JAWS Script Message Files. The next files the Script Manager uses contain header and message information. You can create and modify both of these files using the Script Manager. The first of these two files is the JAWS script header file and has an extension of jsh. You use header files to store things such as constant and global variable definitions. The second of these 2 files is the JAWS script message file and has a file extension of jsm. You use message files to store all of the spoken messages used within your script file. Through the use of the Include statement, you can add the content of both of these files to as many script files as you want. Storing your constant definitions, global variable definitions, and messages in header and message files makes for easier maintenance. For example, when you store all of your messages in a message file, it is much easier to translate those messages from English to another language. Instead of searching your entire script file for all spoken messages, you need only to look at the appropriate message file. See 5.1 The Include Statement for more information on including header and message files.

**The JAWS Script Binary File**

The last file used by the Script Manager is the JAWS script binary file. This file matches the script source file with one exception. The file contains the machine language used by your computer to determine what action to take when a script is performed or a user-defined function is called. The file has an extension of jsb and is created each time you compile your script source file. You cannot view the contents of this file with the Script Manager or any other text editor.

**4.2 STARTING THE SCRIPT MANAGER**

The Script Manager presents a different set of starting options depending on whether you are running an application when you start the Script Manager or just running JAWS. When running an application:

Press INSERT+F2 to open the Run JAWS Manager dialog

Press S followed by ENTER. When you start the Script Manager from within an application, the Script Manager automatically opens the script file for that application. If no file exists, the Script Manager creates the script file. When running the script manager from within the JAWS window:

Press ALT+U to open the Utilities Menu.

Press S. OR

Press INSERT+F2 to open the Run JAWS managers dialog.

Press S followed by ENTER. When you start the Script Manager from the Utilities menu, the Script Manager opens the default script file. When you press INSERT+F2 followed by S and ENTER, then the Script Manager opens the jaws window.jss script file. While the Script Manager is open, you can press INSERT+T and JAWS will announce the script file that is currently open.

Note: You cannot have more than one instance of the Script Manager opened at a time. However, you can open multiple files within the Script Manager. Press CTRL+TAB to cycle between open files in the Script Manager.

**4.3 NAVIGATING SCRIPT FILES**

You can use a number of different ways to navigate through your script file. You can navigate line by line, by script, or display a list of all scripts and user-defined functions within the current script file.

Navigating by Script. You can use the Next Script and Prior Script options from within the Script menu to navigate by script. You can also press F2 to move to the next script and SHIFT+F2 to move to the prior script. Each time you press either keystroke, the insertion point is moved to the beginning line of the appropriate script and JAWS speaks it automatically. When you press F2 with the insertion point on the last script within the current script file, a system beep sounds informing you that you have reached the bottom of the file and have now wrapped around back to the top of the file. Likewise, when you press SHIFT+F2 with the insertion point on the first script in the current script file, a system beep sounds informing you that you have wrapped around to the bottom of the file.

Displaying a List of Scripts.

You can display a list of all of the scripts and user-defined functions contained within your script file. Select the Script List option from the Script menu or press the keystroke CTRL+L. The Script Manager displays the list in alphabetic order. Use your arrow keys to move up and down through the list. You can also use first letter navigation to move directly to a script. After you have located the desired script, press ENTER to move directly to the script or function within the script file. You should exercise care when using this method. Once you press ENTER on the desired script or function name, the Script Manager moves the insertion point directly to the beginning line of the script or function. The Script Manager also highlights the name of the script or function. Since the Script Manager highlights the name, typing any characters will replace the name of the script or function. You can press any of the navigation keystrokes to remove the highlight.

Using Line Numbers. You can also move directly to any line within your script file. Choose Go To Line… from the Script menu or press CTRL+G to display the Go To Line dialog. You can type the line number in the Line Number edit box and press ENTER to move the insertion point to the line. The Script Manager displays the line number along with the total number of lines in the script file in the status bar at the bottom of the application window.

**4.4 FINDING AND REPLACING TEXT**

Like any text editor, you can find specific strings of text. You can also search for and replace specific strings of text.

Finding Specific Text. To find specific strings of text choose Find…from the Edit menu or press CTRL+F to display the Find dialog. When the Script Manager displays the Find dialog, the Find What edit box is active. Type the desired text and press TAB to move to the Match Whole Word only check box. By default, this check box is unchecked. Press SPACEBAR to check this check box and search for text that matches the entire search string of text.

You can then press TAB to move to the Match Case check box. This check box is unchecked by default. Press SPACEBAR to check this check box and cause the search to be case sensitive. Press TAB to move to the Direction set of radio buttons. By default, the Down radio button is selected. Use your arrow keys to change this option.

After you have entered your text and chosen the desired search options, press ENTER to activate the Find Next button. The Script Manager moves the insertion point to the first occurrence of the text. If the text is not found, then the Script Manager plays an appropriate Windows error sound. This Windows sound indicates the string of text was not found. To find the next occurrence of text, select the Find Next option from the Edit menu or press F3. The Find Next action moves the insertion point to the next occurrence of text within the script file. You can continue to press F3 to move through the file until you reach the end of the file. At that point, you will hear a Windows error sound advising you that the search string was not found. To find the prior string of text, choose the Prior Text option from the Edit menu or press SHIFT+F3. The Prior Text action moves the insertion point to the prior string of text within the script file. You can continue to press SHIFT+F3 to move to the prior string of text until you hear a warning sound. This warning sound indicates you have reached the top of the file and the search string was not found.

Incremental Searches.

You can use incremental searches to find a specific string of text. An incremental search is very similar to selecting an option in a combo box or list box. To initiate the incremental search choose Incremental Search from the Edit menu or press ALT+CTRL+I. After you have initiated the search, start the search by typing one character at a time. As you type, the Script Manager matches the characters you have just typed with the corresponding word within the script file. For example, you want to search for the SayMessage function. First, initiate the incremental search. Next, start typing SayMessage until a match is found. After the search has matched the first occurrence of the string of characters you have typed, press F3 to find the next occurrence of text or SHIFT+F3 to find the prior occurrence of text.

Note: To start editing after the incremental search finds the first string of text, press any of the navigation keys to remove the highlight. If you do not remove the highlight before you start editing, the Script Manager interprets your keystrokes as additions to the incremental search. You can also view the text you have typed in the status line at the bottom of the screen.

Replacing Text. Like most text editors and word processors, you can find a specific string of text and replace it with another string of text. Choose the Replace option from the Edit menu or press CTRL+H to display the Replace dialog. When the Script Manager displays the Replace dialog, the Find What edit box is active. Type the string of text you want to find in this edit box. Press TAB to move to the Replace With edit box and type the text you would like to use as a replacement. Press TAB after you have entered the replacement text to move to the Match Whole Word only check box. By default, this check box is unchecked. Press SPACEBAR to check this check box and force the replace text action to only replace text that matches the entire search string.

Press TAB to move to the Match Case check box. This check box is unchecked by default. Press SPACEBAR to check this check box and cause the replace text action to be case sensitive.

A series of command buttons follows the Match Case check box. You can find the next occurrence of text, replace the first occurrence of text, or replace all occurrences of text. Press TAB to move to the appropriate button. Press the SPACEBAR to activate the button and carry out the replace text action. After the Script Manager finishes the find and replace action, you may hear a Windows system sound. This sound indicates the find and replace action is complete. Press ESCAPE to close the Replace dialog and return to the Script Manager.

**4.5 VIEWING INFORMATION**

You can view two types of information within the current script file. You can view script and function information as well as display key word help for any script or function within the current script file.

Viewing Script Information. You can view documentation for any script or function within the current script file. The Script Information dialog is identical to the New Script dialog discussed in 5.2 The New Script Dialog. First, move the insertion point to an individual script or function. You can use the Next Script, Prior Script, or Script List actions to move to the desired script or function. Any of the three navigation methods moves the insertion point to the beginning line of the script or function. Press DOWN ARROW twice to move into the body of the script or function. This also removes the highlight from the name of the script or function. To view the documentation, choose the Documentation option from the View menu or press CTRL+D. Using either action displays the Script Information multipage dialog. The first page of this dialog, General, displays general information about the script or function. The information shown in the Script Information dialog changes depending on whether you are viewing script or function information. When you move to a script, the following information is shown:

Script Name

Can be Attached to Key check box

Synopsis

Description

Category

Keystroke Assignment.

When you select a function, the Script Information dialog displays the following information:

Function Name

Can be Attached to Key check box

Synopsis

Description

Category

Return Type

Return Description. The Parameters page displays any parameters used by the selected function. You can view a list of existing parameters on this page along with their descriptions. You can find more information about both pages of the Script Information dialog in 5.2 The New Script Dialog. After you are finished viewing the script information, press ESC to close the dialog.

Key Word Help. Within the Script Manager, you can get key word help. Key word help is available for scripts and all three types of functions contained within the current script file. To view key word help, move the insertion point to the desired script or function name and press SHIFT+F1. The Script Manager displays the Key Word Help dialog containing specific information on the script or function. When you view key word help for a script or user-defined function, the information shown is taken from the appropriate documentation file. For example when you view key word help from within the script file for Microsoft Word, Microsoft Word.jss, The Script Manager retrieves the documentation for all scripts and user-defined functions from the Microsoft Word.jsd file.

If you are viewing key word help for a built-in function, the Script Manager retrieves that information from the builtin.jsd documentation file. The builtin.jsd file contains information for all built-in functions within the JAWS scripting language. After you have reviewed the key word help information, press ESC to close the Key Word Help dialog.

**4.6 CHAPTER EXERCISES**

The following exercises give you practice in using many of the functions of the Script Manager. The objective of the exercise is listed first.

**Exercise 4.1: Navigating Scripts. The objective of this exercise is to gain practice using the Next and Prior** script movement commands.

Press INSERT+F2 to display the Run JAWS Managers Dialog.

Press S followed by ENTER. These keystrokes select and run the Script Manager.

After the Script Manager opens, press CTRL+SHIFT+D to open the default script file.

Press F2 to move through the file a script at a time. Does JAWS automatically speak the name of the script or function?

Press SHIFT+F2 to move to the prior script. Does JAWS continue to speak the name of the script or function automatically?

**Exercise 4.2: Listing Script and Function Names**

The objective of this exercise is to give you experience activating and using the Script List dialog. If the Script Manager is already running, make sure it is the active application. If the Script Manager is not running, follow steps 1 through 3 in the previous exercise to start the Script Manager and open the default script file.

Press CTRL+L to display the Scripts List dialog.

Press DOWN ARROW to move down through the list of scripts and functions.

Continue pressing DOWN ARROW until you reach the script titled "AdjustJAWSOptions".

Press ENTER. Did JAWS move to and speak the name of the script?

Press CTRL+L a second time.

Type SAY. Did the Script Manager select the first script beginning with say? Press INSERT+UP ARROW to verify the name of the script.

Press ESC to close the Scripts List dialog.

**Exercise 4.3: Viewing Script Documentation**

The objective of this exercise is to give you practice in using the Script Information dialog to view script information. If you have Script Manager running, make sure it is the active application. Press INSERT+T to read the title of the active application. If you do not have Script Manager running, follow steps 1 through 3 in exercise 4.1 to start the Script Manager and open the default script file.

Press CTRL+L to display the Script List dialog.

Type SAY to move to the first script beginning with that text.

Press DOWN ARROW until you reach the script named "SaySystemTime".

Press ENTER to select the script and move to the script within the default script file

Press DOWN ARROW twice to move into the body of the script. This also removes the highlight from the script name.

Press CTRL+D to display the Script Information dialog.

Review the documentation for the SaySystemTime script. Press TAB and SHIFT+TAB to move through the fields in the Script Information dialog.

Press ESC to close the dialog after you have finished reviewing the information.

**5.0 CREATING SCRIPTS**

In Chapter 4, you read about many of the features available for your use in the Script Manager. This chapter discusses using the Script Manager to create your scripts and functions. You can use the New Script dialog to create new scripts and user-defined functions. You use this dialog to enter the script or function name, synopsis, description, and more. You can open the New script dialog from the Script menu or by pressing the keystroke shortcut, CTRL+E. After you have created the new script or function, you can use the Insert Function dialog to add a variety of functions to the body of the script or function. You can access the Insert Function dialog from the Script menu or by pressing the keystroke shortcut, CTRL+I.

**Table of Contents**

Chapter 5.0, Creating Scripts, contains the following sections:

5.1 Before You Start

5.2 The New Script Dialog

5.3 Parts of a Script

5.4 Saving and Compiling Script Files

5.5 Inserting Functions

5.6 Chapter Exercises

**5.1 BEFORE YOU START**

Before you start writing scripts, there are a few things you should add to your new script file. You can add comments at the beginning of the file to document the script file. Comments make it easier to maintain the scripts and user-defined functions contained in your script files. You can also use include statements. An include statement tells JAWS to take the contents of one file and add it to your script file. You usually include script header and message files.

**Adding Comments**

You can add comments anywhere within your script file. You begin a comment with a semicolon (;). When JAWS encounters a semicolon, the compiler ignores anything to the right on the line on which the semicolon appears. You can use comments to give information about the script file, describe complex script statements, and more. Some examples of comments follow:

; script file for Notepad

SayString ("Hello world!"); Speak a message

; SayString ("Hello world!") tell the compiler to ignore this entire line

When you use multiple lines for comments, you must place a semicolon at the beginning of each line. Otherwise, you will encounter syntax errors when you try to compile your script file. When you begin with a blank file, it is a good idea to start the file off with a comment section. You can use this comment section to document any information about the application being scripted, version of JAWS, and date you created the file. An example of a comment block follows:

; Script file for Notepad in Windows Vista

; written for JAWS version 12.0.XXX

; Written by the Freedom Scientific Training Department December 2010

The Include Statement

After you have created a comment block at the top of your new script file, you can add one or more include statements. The Include statement tells JAWS to include the entire contents of a file in the script file at the time you compile the file. You can think of an include statement as a way of putting a large amount of information into your script file without having to actually type it. By using the include statement, you can store all of your messages used in your script files in one location. Then you can use the include statement in each script file that will use those messages. If you ever need to make a change to a message, you need only make the change in the message file. Since it is included in each script file, you make the change in only one place. After you have made the change, you can compile each script file to include your changes. You can also store all of your global variable and constant declarations in a script header file. You can then use the include statement in multiple script files and use the global variables and constants in each of those files. When you need to make a change to a constant value, for example, you need only make it one place. After you make the change, you just compile each of the script files that use the particular script header file. This makes changing a constant value much easier because you need not make the change in several places.

You begin the include statement with the word "include," without the quotation marks. Following the word "include" is the file name to be included. The file name must be enclosed in quotation marks. The file name must also include the file extension. You can include both script header and script message files. Each included file must be located in your JAWS settings directory. Examples of the include statement follow:

include "hjconst.jsh"; default constant file

include "hjglobal.jsh"; default global file

include "common.jsm"; message file

**5.2 THE NEW SCRIPT DIALOG**

To create a new script within the current script file, choose New Script from the Script menu or press CTRL+E. After you have initiated the new script action, the Script Manager displays the New Script dialog and the Script Name edit box is active. The New Script dialog is a multipage dialog. The first page, General, is identical to the General page in the Script Information dialog discussed previously in 4.5 Viewing Script Information. You can enter the script name, check the attach to check box, enter the synopsis and description, and assign a keystroke within this page. The second page, Parameters, contains parameter information for functions you create. You can press CTRL+TAB to move between the 2 pages. The Script Manager writes all the information you enter in this dialog to the documentation file that corresponds to the current script source file. For example, the spreadsheet.exe application mentioned earlier has a script file name of Spreadsheet.jss. The corresponding documentation file, or .jsd file, has the name of Spreadsheet.jsd.

General Page. This is the first page of the New Script multipage dialog. You use this page to enter all the basic information about your new script or function. You can use TAB and SHIFT+TAB to move between the controls on this page.

Script Name. When you first initiate the new script action, this edit box is active. You enter the name of your script here. It is helpful to use a name that is descriptive of the action the script performs. You may use several words concatenated together. For example, in MySampleScript, start each word with a capital letter so JAWS will pronounce the name as separate words. The script name cannot contain spaces or punctuation, only alpha numeric characters. If you try to enter spaces or punctuation in the script name, you will hear an audible beep from the computer.

Can be Attached to Key. Press the SPACEBAR to check this item when you want to create a script. If you leave this check box unchecked, you will create a function instead of a script. Remember that scripts can be attached to keys but functions cannot.

Synopsis.

Type a brief statement of what the script does in this field. JAWS speaks this information when you or other users access Keyboard help mode (INSERT+1) or Key Word help (SHIFT+F1). To hear synopsis help, press the INSERT+1 key combination using the 1 on the number row, not the number pad 1. JAWS says "keyboard help on." Next, press the keystroke combination that activates your script. JAWS should say the information placed in this edit box. You can leave the keyboard help mode by pressing INSERT+1 a second time.

Description.

Enter a more detailed explanation of the action the script performs in this edit box. This description is used if you enter the Keyboard help mode (INSERT+1) or Key Word help (SHIFT+F1) in the Script Manager. The description is spoken by quickly pressing the key combination (that activates your script) twice after turning Keyboard Help on.

Category. You can type in a category name or choose one from the drop down combo box. This feature is not in use so you do not need to select a category at this time.

Assign To. This edit box is only available if you checked the Can Be Attached to Key check box. Type the keystroke combination you wish to use for your script. For example, to enter CTRL+ALT+Z, press and hold the CTRL and ALT keys while pressing the Z key. If the choice you make is already assigned to another script, the Script Manager displays an error dialog. You can continue with the assignment or choose a different keystroke combination. You cannot use standard reading commands to review the keystroke you have entered. For example, if you press INSERT+UP ARROW to try and read the keystroke you have entered, JAWS will think you are wanting to use INSERT+UP ARROW as the keystroke to activate the script. To review your keystroke selection, press TAB to move away from the edit box and then SHIFT+TAB to return. JAWS will automatically read the keystroke you entered.

**Function Returns**

This choice is only available if you did not check the Can Be Attached to Key check box. You can type in a Return Name or choose one from the drop-down combo box. Among the choices are Handle, Int, Object, String, or Void. Select the type of return that your function is designed to return to the calling script. Select Void if you do not need to use any value returned by the function. The return type appears on the first line of the function before the word Function.

Return Description. This choice is only available if you did not check the Can Be Attached to Key check box. Enter a brief description of the information that is being returned by the function and how the information is to be used.

Parameters Page.

This is the second page of the New Script multipage dialog. You use this page to enter all parameters that are to be used by your new function. You will not use this page of the new script dialog unless you are creating a function. You can use TAB and SHIFT+TAB to move between the controls on this page. The Parameters page will be discussed in greater detail later in this manual.

**5.3 PARTS OF A SCRIPT**

When you use the New Script dialog to create a new script or function, the Script Manager automatically places 3 pieces of information into the current script file. These three parts make up your script or function.

Beginning Line. Each new script or function you create has a beginning line that is created when you enter the script or function information into the New Script dialog. The contents of the beginning line vary depending on whether you created a script or function. When you create a new script, the following line begins the script definition:

Script ScriptName ()

On the other hand, when you create a function the following line is added to your script file:

Void Function FunctionName (parameter list)

Remember that a function always has a return type. This return type can be a handle, integer (Int), object, or string. The function can also return nothing. In cases where the function returns nothing to the calling script or function, the word void is shown as the return type. If the function uses any parameters, they are shown between the parentheses following the function name. If you decide to change the name of your script or function in the beginning line, it is best to change the name through the Script Information dialog. When you make changes using the General page of the Script Information dialog, the script or function name is changed automatically for you in the associated documentation file. Likewise if you forgot to add a parameter to your new function, you can use the Parameters page of the Script Information dialog to add it. See section 4.5 for more information on displaying the Script Information dialog.

**The Body**

The body immediately follows the beginning line of the script or function. When you create a new script or function, the body is the three blank lines immediately following the script or function beginning line. You add all the instructions your script needs to perform its given task here. These instructions are also referred to as statements. Each type of script statement has a specific job to perform, and each follows certain formatting and usage rules known as syntax.

**Ending Line**

Each new script or function you create also has an ending line. This line ends the individual script or function. This is always the last line of your script or function. When you create a new script, the ending line looks like the following:

EndScript

On the other hand, the ending line of a user-defined function looks like the following:

EndFunction

**5.4 SAVING AND COMPILING SCRIPT FILES**

You have two options when it comes time to save your script file. You can choose to compile and save the file with one keystroke. You can also choose to save the file without compiling. Both save options are described below.

**Saving and Compiling Script Files**

After you have created a new script or function, you can choose the Compile option from the File menu or press CTRL+S to save and compile your script file. This action causes the script compiler to examine each of the scripts and functions contained in the script file. The compiler checks each statement in those scripts and functions making sure the syntax is correct. When the compiler finds an error, the compilation process is stopped and the Compile Error message dialog is displayed. This dialog advises you what type of error the compiler encountered with the Ok button being the active control. To clear the dialog after you have reviewed the cause of the error, press SPACEBAR to activate the Ok button or ESCAPE to close the dialog. After you have cleared the dialog, the insertion point is placed in the line containing the error. At this point, you should review the line making sure the statement is correct. You may also find the line containing the insertion point does not contain the error. Many times a previous statement with incorrect syntax can cause an error in a line following it. In cases such as this, you will need to review both the line containing the insertion point and any preceding lines. Although the compiler checks the syntax of script statements, it does not check spelling or logic. As you add function calls to the body of the script, you must make sure the name of the function is spelled correctly. Failure to spell the names of your functions correctly will result in run-time errors. A run-time error occurs when you begin testing your script and a problem occurs during the execution of the script. When you hear "unknown function call to Function ABC" where ABC represents the name of a user-defined or built-in function, you can be assured you have a run-time error. You should go back to your script file and make sure all function names are spelled correctly.

**Saving Script Files Without Compiling**

You may not want to compile your script file immediately. You may find you need to save the file before you complete the script or function you are currently writing. If you try to save the file using CTRL+S, the script file is compiled and then saved. If your script is not complete, then the compiler could encounter errors causing the file not to be saved. You can select the Save Without Compiling option from the File menu or press CTRL+W. When you choose this option, the script file is not compiled before it is saved. Instead, the file is saved much like you would save a document in a word processor or text editor. You can then come back to the file at a later date and continue working on the script or function.

**5.5 INSERTING FUNCTIONS**

You can insert both built-in and user-defined functions into the body of your scripts and functions. You can eliminate any spelling and syntax errors when you use this method to add these functions.

Activating the Insert Function Dialog

You can choose the Insert Function option from the Script menu or press CTRL+I to activate the Insert

Function dialog.

Note: You cannot activate the Insert Function dialog unless the insertion point is within the body of a script or function. When you activate the Insert Function dialog, the Function Name edit box is active. The dialog box also contains a Function Name list box, Description edit box, Function Returns edit box, and a number of different command buttons.

**Selecting the Correct Function**

You can type the name of the function you want to add to your script in the Function Name edit box. As you type the letters of the function name, JAWS moves the highlight to the matching name in the Function Name list box. When a matching function name is found, JAWS speaks the description of the function automatically. If you have typed several letters in the Function Name edit box, but have not quite found the correct function, you can press TAB to move to the Function Name list box. This list box contains all built-in functions along with any user-defined functions from both the default and application script files. You can use your arrow keys to move up and down the list. You can also use first letter navigation to move immediately to a function that begins with a specific letter. For example typing S in this list box will move the highlight to the first function that begins with the letter S. As in the Function Name edit box, JAWS speaks the description automatically as you move through this list.

Reviewing the Function Description. If you did not hear the entire description of the function, or would like to review the description of the function, you can press TAB until you reach the Description edit box. JAWS retrieves the description for functions built into JAWS from the documentation file for built-in functions, Builtin.jsd. For user-defined functions, JAWS retrieves the description from the appropriate application specific documentation file. You can use your arrow keys along with any of the other cursor navigation keys to read the information contained in this edit box.

Accepting the Function. After you have chosen the desired function, you can activate the Next button to continue inserting the function. You can activate this button by either pressing TAB until you reach the button and then press SPACEBAR or press ENTER while either the Function Name edit box or Function Name list box is active. If you move to and activate the Finish button, then the function is inserted directly into your script or function. However, you will not be prompted for any parameters that the function may require.

**Adding Parameters**

After you activate the Next button, the Script Manager prompts you for any parameters required by the function. The Parameter 1 edit box is active and the Script Manager reads the description of the parameter automatically. Like the description of the function, the Script Manager retrieves the parameter description from the appropriate documentation file. You can choose to type in the value of the parameter, an appropriate variable type such as a string or integer, or insert a function whose return value will act as the parameter. If you choose to insert a function in the Parameter edit box, then you can activate the Insert Function button by pressing ALT+I. This action displays a second Insert Function dialog. You can select the function in the same manner as you did for the initial function. You may find that some functions only require one parameter while others may require more. You can add the subsequent parameters in the same manner as you did the first. When you enter the last parameter, activate the Finish button to complete the insertion of the function. You need only to press ENTER from the last parameter edit box to activate this button. You can also press TAB to move to the button followed by SPACEBAR to activate it. In other words, the Insert Function dialog contains both Next and Finish buttons. You use the Next button to add the necessary parameters to the function, and you use the Finish button to complete the insertion of the function into your script.

**5.6 CHAPTER EXERCISES**

These exercises will give you practice in creating, inserting functions, compiling, and testing your scripts.

**Exercise 5.1: Adding Comments and Include Statements**

The objective of this exercise is to give you some practice adding comments and include statements to your script file. Before you begin this exercise, make sure you only have JAWS and Notepad running. You can press INSERT+F10 to list all of your running applications.

Press INSERT+F2 followed by S and ENTER while Notepad is the active application. This opens The Script Manager with the Notepad.jss file in the edit window.

Note: Unless you have written any scripts in the Notepad script source file, this file will be empty.

Press CTRL+HOME to move to the top of the file. This ensures the insertion point is at the top of the script file.

Note: If the insertion point is already at the top of the file, you may hear a windows system sound.

Type a semicolon (;). This starts a comment line in the script file.

Type the text "JAWS [ version ] Script file for Notepad in Windows X" where X denotes the operating system you are using and version denotes the version of JAWS you are writing scripts for. At the end of this line, press ENTER to move the insertion point to the next line.

Type a semicolon (;) to begin a second comment line in your script file.

Type the text of "Written By" followed by your name and the current date. At the end of this line press ENTER to complete the line and move the insertion point to a new blank line.

Press ENTER twice. This will create a blank line and then move the insertion point to a second blank line after your comment section. This will help separate the comment section from the area where include statements are placed.

Type include "HJConst.jsh" This adds an include statement telling JAWS to include the contents of the HJConst.jsh file within the Notepad script file. Press ENTER to complete the line and move the insertion point to a new blank line.

Note: When you type the include statement, the file name must be surrounded by quotation marks.

Press CTRL+W to save your script file without compiling. After completing the steps above, your Notepad script file should start with the following lines:

; JAWS [ version ] Script file for Notepad in Windows X

; Written by your name on mm/dd/yy

include "hjconst.jsh"

Note: The text of your name should be replaced with your actual name. The date should also be filled in for the date you are doing this exercise.

**Exercise 5.2: The HelloWorld Script**

The objective of this exercise is to use the New Script dialog to create your first script. If you have just completed the previous exercise, make sure that Script Manager is the active application. If you have closed the Script Manager, make sure Notepad is still running, switch to Notepad, and follow step 1 in the previous exercise to start the manager.

Press CTRL+END to move to the bottom of the Notepad script file. This will ensure your new script is created at the bottom of the file below the comments you added in the previous exercise.

Press ENTER twice to create a blank line between the Include statement and your new script.

Press CTRL+E to display the New Script dialog. The Script Name edit box is active.

Type HelloWorld In the Script Name edit box.

Note: Script names cannot contain spaces or any punctuation marks such as dashes or underlines. The Script Manager will cause a system beep to occur any time you attempt to add any of these characters to your new script name. You should also capitalize the first letter of each word in a multiple word script name.

Press TAB to move to the Can be Attached to Key check box and press SPACEBAR. This step tells JAWS you want to create a script and not a function.

Press TAB to move to the Synopsis edit box. Type "Speaks a message in Notepad" into this edit box without the quotation marks. This text is spoken by JAWS when you enter Keyboard Help and press the keystroke that activates this script.

Press TAB to move to the Description edit box. Type "Speaks a hello world message in Notepad." without the quotation marks. The description is heard when you enter keyboard help and press the keystroke that activates this script twice in succession.

Press TAB to move to the Category list box. Since this is not used, you do not need to select an item from this list.

Press TAB to move to the Assign To edit box. Be careful as any keystroke you press while in this edit box will be used as the keystroke that activates this script. Press CTRL simultaneously with 1 in the numbers row, not the number pad. Since the Ok button is the default button in this dialog, you can press ENTER to close the New Script dialog and return to the Notepad script file.

Press UP ARROW once to move the insertion point to the first blank line in the body of the script. Type the following text:

SayString ("Hello World!")

Note: The SayString function is no longer used to speak messages with JAWS. This function is obsolete and should only be used for debugging scripts and functions.

After you have typed the line of text above, the insertion point is at the end of the line. Press DOWN ARROW once to move to the next line. You can press DELETE twice to remove the remaining blank lines. After you have followed the steps above, you should have the following script in your script file:

Script HelloWorld ()

SayString ("Hello World!")

EndScript

If your script does not look like the one above, make corrections as needed. You can edit the text of any script just as you would in a word processor or text editor.

**Exercise 5.3: Compiling and testing the HelloWorld Script**

The objective of this exercise is to compile and test the HelloWorld script created in exercise 5.2.

Press CTRL+S to save and compile the Notepad.jss script file. Did you hear the "Compile Complete" message? If not, go back to the script and check it against the script shown in the previous exercise.

After you have successfully compiled your script file, you are ready to test your new script. Press ALT+TAB to move back to Notepad.

Press CTRL+1. Did you hear the message spoken by JAWS?

Press INSERT+1 to turn on Keyboard Help. You should hear JAWS speak "Keyboard help on" when you are using beginner verbosity. Otherwise, you should hear , "On."

Press the keystroke that activates your script, CTRL+1. Did JAWS speak the synopsis of your script?

Press the keystroke, CTRL+1, twice in succession. Did JAWS speak the description of your script?

Press INSERT+1 a second time. You will hear JAWS say "Keyboard help off" when you are using beginner verbosity. Otherwise, you should hear , "Off."

**Exercise 5.4: Using the Insert Function dialog.**

The objective of this exercise is to create a new script and insert a function in the body of the script using the Insert Function dialog. If you are still in Notepad, press ALT+TAB to move back to the Script Manager.

Press CTRL+END to move to the bottom of the file.

Press ENTER twice to create a new blank line. This blank line separates the new script from the previous HelloWorld script.

Press CTRL+E to activate the New Script dialog. Your cursor should be located in the Script Name edit box.

Type SayGreeting without spaces. Capitalize the S in say and the G in Greeting.

Press TAB to move to the Can be Attached to Key check box and press SPACEBAR to check it. This tells JAWS you are creating a script not a function.

Press TAB to move to the synopsis edit box. Type "Speaks a greeting in Notepad".

Press TAB to move to the description edit box. Type "Speaks a greeting in Notepad using the Say function."

Press TAB to move to the category combo box. As the category is not used, you do not need to make a selection.

Press TAB to move to the Assign To edit box. Press CTRL simultaneously with the 2 on your numbers row, not the num pad. You should hear JAWS speak "CTRL+2." Since the Ok button is the default button in this dialog, press ENTER to close the New Script dialog.

Press UP ARROW once to move to the first blank line in the body of your new script.

Press CTRL+I to activate the Insert Function dialog box. The cursor should be located in the Function Name edit box.

Type "Say." JAWS moves the highlight in the Function Name list box of this dialog to the location of the Say function and speaks its description.

Press ENTER to tell JAWS this is the function you want to add to your script. You will then be prompted for the first parameter of the function. This parameter is the string of text that is spoken by JAWS each time you press CTRL+2.

Note: A parameter is information a function uses to perform its task.

Type "Hello world, this is my second script." including the quotation marks. This is the actual message JAWS speaks each time you press CTRL+2.

Press ENTER to add the first parameter. You will then be prompted for the second parameter of the Say function.

Type "OT\_MESSAGE" in all capital letters. This is a constant value contained within the HJConst.jsh script header file. This parameter tells JAWS what type of message this is and when to speak it.

Press ENTER to accept the second parameter. Press ENTER again to leave the third parameter empty and close the Insert Function dialog box. You are returned to the script file and the insertion point is at the end of the line containing the function you just inserted.

Press the LEFT ARROW KEY to move the cursor back into the text of the function and you will find a blank space and a comma in between the OT\_MESSAGE and the right parenthesis at the end of the function. This happened because we chose to leave the third parameter of the Say function blank. In order for the script to compile properly, you must delete the blank space and the comma in between the OT\_MESSAGE and the right parenthesis.

Press DOWN ARROW once to move to the next blank line. Press DELETE twice to remove the two remaining blank lines. After following the steps above, your script should look like the following:

Script SayGreeting ()

Say ("Hello world, this is my second script", OT\_MESSAGE)

EndScript

Now you are ready to compile the script. Press CTRL+S to compile the script file. If you don't hear "compile complete", then you may have an error in your script. Go back and review the script. You should make sure your script matches the script shown above. After you have corrected any errors, try compiling the script again.

Once you save and compile your script without any errors, return to Notepad and test your script. Press CTRL+2. Did JAWS speak the message?

**6.0 READING INFORMATION WITH SCRIPTS**

You learned about the Script Manager and used it to create your first two scripts in Chapter 4 and Chapter 5. You created scripts that simply spoke messages when you pressed the appropriate keystroke. In this chapter, you will learn how to use the scripting language to read areas of the screen by moving various cursors. As you create scripts that read parts of a window, you should understand the differences between the various cursors and how they can be manipulated through a script or function.

JAWS has three cursors you can use to read areas of the screen:

PC cursor

JAWS cursor

Invisible cursor

Manipulating cursors to read information is not the most efficient method of accessing that information. However, as a beginning scripter, you can use cursor manipulation as a quick way to write scripts to read information within a given window. You do not need to know the relationship between windows or the types of windows your scripts are accessing. As your knowledge of the scripting language grows, you will learn how to create scripts that read information more efficiently using advanced techniques.

**Table of Contents**

Chapter 6.0 Reading Information with Scripts, contains the following sections:

6.1 Before You Write the Script

6.2 PC Cursor

6.3 JAWS Cursor

6.4 Invisible Cursor

6.5 Before You Move a Cursor

6.6 Moving the Cursor

6.7 Speaking the Information

6.8 Chapter Exercises

**6.1 BEFORE YOU WRITE THE SCRIPT**

You may find a variety of reasons for moving a cursor around a given window. Some applications contain buttons that are only accessible with the mouse or JAWS cursor. These buttons usually cannot be reached by pressing TAB or through the menu system. You may also encounter applications that place blocks of text in areas of the screen that can only be read using the JAWS or invisible cursors. Before you create any script that moves a cursor to read information, you should determine the outcome of the script. If you are trying to activate a button within an application window, use cursor movement keystrokes to move the JAWS or invisible cursors to the button. You should keep track of each keystroke you press as you move the appropriate cursor. Each cursor movement command you use has an equivalent built-in function. For example, the built-in function, NextLine, acts just like pressing the DOWN ARROW from your keyboard. It moves the active cursor down one line. Instead of using INSERT+PAGE DOWN to read the bottom line of the screen, use cursor movement commands instead. You can use the JAWS cursor to accomplish this task by following the steps below:

Press NUM PAD MINUS to activate the JAWS cursor.

Press INSERT+NUM PAD MINUS to route the JAWS cursor to the location of the PC cursor.

Press PAGE DOWN to move the JAWS cursor to the bottom of the window.

Press INSERT+UP ARROW to read the current line.

Now you know the steps needed to move the JAWS cursor to and read the contents of the bottom line of the window. You can take these steps and use built-in functions to accomplish the same task. You also need to think about which cursor you will use to read or access the information. Are you going to need mouse commands? Or will you only need to read the information? Use the JAWS cursor when you will need to perform mouse commands such as left or right mouse clicking. Use the invisible cursor when you need to read information but do not need to perform mouse commands. You also need to remember not to leave the wrong cursor active when your script finishes. If you are using the invisible cursor to read information on the screen, you don't want to leave that cursor active when your script finishes. Instead, you want to make sure the cursor, which was active before the script is executed, is again activated when the script finishes.

**6.2 PC CURSOR**

You can think of the PC cursor as the system focus. This cursor is linked to the keyboard functions of the Windows operating system and Windows based applications. This is the cursor that is used when typing information, moving through options in dialog boxes, and selecting options or icons. You can move the PC cursor using a variety of navigation keystrokes such as PAGE UP, PAGE DOWN, HOME, and END. You can also use the arrow keys to move the cursor. However, the PC cursor can only go where your application allows it to go. You will have to use another cursor when the text is not within the range of movement of the PC cursor. You can use many of the built-in functions to move the PC cursor within the areas to which it is limited.

**Activating the PC Cursor**

You do most of the reading in any window using the PC cursor. You can use the built-in function, PCCursor, to activate the cursor from within a script or user-defined function. When you use the PCCursor function, the PC cursor is activated just as if you pressed NUM PAD PLUS from your keyboard.

**6.3 JAWS CURSOR**

The JAWS cursor acts just like the mouse pointer. It can go anywhere the mouse pointer can go. In fact, the mouse pointer accompanies the JAWS cursor as you move it around the screen. The JAWS cursor is not limited to any specific boundaries and you can use the same navigation keystrokes as you can when the PC cursor is active. There are some things you should know before you start moving the JAWS cursor. First, since the JAWS cursor acts as the mouse pointer, you may find that tool tips are opened as the JAWS cursor moves across certain areas of the screen. You may also find that other types of pop up information appear as the JAWS cursor is moved. The Windows operating system automatically performs these actions or events. You have no control over when these actions or events occur. You may find that you are using the JAWS cursor to activate a specific button on the screen. You may not want to move the JAWS cursor from that location to read another area of the screen. Moving the JAWS cursor to read another area of the screen forces you to move the JAWS cursor back to that button after your script finishes. Before you use the JAWS cursor, you can determine if it is the best cursor for the job by answering the following question:

Do you need to perform any mouse actions i.e. left click, on the item?

If you answered "Yes" to the question above, then you should use the JAWS cursor. If you answered "No" to the question, then the JAWS cursor is not the best cursor to use.

**Activating the JAWS Cursor**

You activate the JAWS cursor using the built-in function, JAWSCursor. This function activates the JAWS cursor regardless of its location within your system. When you use the JAWSCursor function, the JAWS cursor is activated just as if you pressed NUM PAD MINUS from your keyboard.

**6.4 INVISIBLE CURSOR**

The Invisible cursor was created for the sole purpose of accessing information on the screen. You have probably not used the Invisible cursor during your daily use of JAWS. The Invisible cursor is identical to the JAWS cursor. You can move it anywhere on the screen, but you cannot perform any mouse actions such as a left mouse click. You use the same navigation keystrokes as you do when other cursors are active. You can activate the Invisible cursor from your keyboard by pressing NUM PAD MINUS twice in succession. When you only need to read information on the screen, then the Invisible cursor is the best cursor to use.

**Activating the Invisible Cursor**

**You can use the built-in function, Inv**isibleCursor to activate the Invisible cursor regardless of its location on your screen. The InvisibleCursor function activates the cursor just like using NUM PAD MINUS twice from your keyboard.

**6.5 Before You Move a Cursor**

After you activate a cursor you should save its location. This ensures that the cursor is restored to the correct position on the screen after your script is performed. For example, when you are using the JAWS cursor to access a specific button within an application, you want it to stay there because you need to perform a left mouse click. If you write a script that moves the JAWS cursor without saving it first, then you must move the JAWS cursor back to the button after your script is finished. The end result is that you are left with the wrong cursor activated and in the wrong place.

You also need to route the cursor to the active application window. This action ensures the cursor you are using to read specific information is in the correct window. This is important as the JAWS and invisible cursors can be anywhere on your screen. So it is always a good idea to move these cursors to the active application. Otherwise, moving the cursor may yield unexpected results.

You can use the SaveCursor built-in function to save the cursor and its location. The SaveCursor function notes the active cursor and its present location. You can call this function after activating any of the cursors such as the JAWS cursor. The following block of code illustrates the use of the SaveCursor function:

JAWSCursor (); activate the JAWS cursor

SaveCursor (); save the JAWS cursor and its location

You can call the SaveCursor function multiple times in a script, after each cursor you activate. This is called "stacking saved cursors." The following code example illustrates the use of multiple SaveCursor function calls:

JAWSCursor () ; activate the JAWS cursor

SaveCursor () ; save the JAWS cursor and its location

InvisibleCursor () ; activate the invisible cursor

SaveCursor () ; save the invisible cursor and its location

Restoring Saved Cursors

After you perform your script and JAWS finds one or more saved cursors, JAWS restores them automatically. When you save multiple cursors, JAWS restores the cursors in reverse order. You can also use the RestoreCursor built-in function when you need to restore a specific cursor before your script finishes. This function is useful when you need to save a cursor first, perform an action with that cursor, and restore it before the script finishes.

Example 1: Using the RestoreCursor function

PCCursor () ; activate the PC cursor

SaveCursor () ; save the pc cursor

JAWSCursor () ; activate the JAWS cursor

SaveCursor () ; save the JAWS cursor

; reading statements go here

RestoreCursor (); restore the JAWS cursor

In the above example, the RestoreCursor function restores the last saved cursor. In this case, the RestoreCursor function restores the JAWS cursor since it was the last cursor saved.

**Routing the Cursors**

The PC cursor is generally the active cursor in most applications. When you need to access information outside of the boundaries of the PC cursor, you need to route either the JAWS or Invisible cursor to the location of the PC cursor. When you route either cursor, you have a starting point that you know exists within the active application.

**Routing the JAWS Cursor**

You can use the built-in function, RouteJAWSToPC to move the JAWS cursor to the location of the PC cursor. This function repositions the JAWS cursor to the exact position of the PC cursor. This function also moves the mouse pointer to the location of the insertion point or selection highlight. However, this function does not activate the JAWS cursor. Any cursor movement functions used in your script will move the active cursor instead of the JAWS cursor. If you want the JAWS cursor to move you must activate it first.

Example 2: Using the RouteJAWSToPC function

JAWSCursor (); activate the JAWS cursor

SaveCursor (); save the JAWS cursor and its location

RouteJAWSToPC (); move the JAWS cursor to the location of the PC cursor

In the above example, the JAWS cursor is first activated and then saved. The RouteJAWSToPC function moves the JAWS cursor from its current location to the location of the PC cursor. At this point, you can use some of the cursor movement functions to move the JAWS cursor. See 6.6 Moving the Cursor for more information on cursor movement functions.

**Routing the Invisible Cursor**

You can use the built-in function, RouteInvisibleToPC, to move the invisible cursor to the location of the PC cursor. This function repositions the Invisible cursor to the exact position of the PC cursor. Unlike the JAWS cursor, the Invisible cursor is not connected to the mouse pointer, so the function does not move the mouse pointer. If you want to move the Invisible cursor you must also activate it first.

Example 3: Using the RouteInvisibleToPC function

InvisibleCursor (); activate the Invisible cursor

SaveCursor (); save the location of the invisible cursor

RouteInvisibleToPC (); move the Invisible cursor to the PC cursor

In the above example, the Invisible cursor is activated then saved. The RouteInvisibleToPC function then moves the Invisible cursor to the exact location of the PC cursor. Now that the Invisible cursor is in the active window, you can use cursor movement commands to read information contained within the window.

**6.6 moving the cursor**

After you have activated, saved, and moved the JAWS or invisible cursor to the active application window, you can move the cursor. Nearly all of the keyboard commands you use to move the various cursors within JAWS have corresponding built-in functions. For example, pressing DOWN ARROW to move the cursor to the next line has a comparable built-in function of NextLine. Likewise, pressing RIGHT ARROW has a comparable built-in function of NextCharacter.

Note: When you press DOWN ARROW to read the next line, behind the scenes JAWS activates the SayNextLine script. The SayNextLine script does more than just move the cursor down to the next line. The script determines what type of window is active and then reads the information accordingly.

**JAWSPageDown and JAWSPageUp**

The JAWSPageDown and JAWSPageUp built-in functions perform special versions of their respective keyboard commands. You can use these functions when any of the cursors are active. However, the outcome of the function is based on the active cursor. When the PC cursor is active, each function performs the standard PAGE UP and PAGE DOWN commands for the application. For example, when you press PAGE DOWN from within a word processor, the PC cursor is moved down a screen of information. Likewise, when you press PAGE UP from within a word processor, the PC cursor is moved up one screen of information.

Example 1: Using the JAWSPageDown function with the PC cursor

PCCursor (); activate the PC cursor

SaveCursor (); save the location of the PC cursor

JAWSPageDown (); performs a page down command using the PC cursor

In the above example, the PC cursor is activated and then saved before it is moved. Next, the JAWSPageDown function moves the PC cursor down one screen of information. When the JAWS or Invisible cursor is active, the JAWSPageDown function moves the cursor to the bottom of the active window based on the cursor restriction setting. Likewise, the JAWSPageUp function moves the JAWS or Invisible cursor to the top of the active window based on the restriction setting.

Example 2: Using the JAWSPageDown function with the Invisible cursor

InvisibleCursor (); activate the Invisible cursor

SaveCursor (); save the location of the Invisible cursor

RouteInvisibleToPC (); move the Invisible cursor to the location of the PC cursor

JAWSPageDown (); move the Invisible cursor to the bottom of the screen

In the above example, the Invisible cursor is activated and then saved. Next, the RouteInvisibleToPC function routes the Invisible cursor to the location of the PC cursor in the active window. Finally, the JAWSPageDown function moves the Invisible cursor to the bottom of the active window.

Note: The JAWS restriction setting is set to application by default. This setting keeps the JAWS and Invisible cursors within the boundaries of the active application window. You can cycle through the various restriction settings by pressing INSERT+R. See the JAWS help system for more information on cursor restriction settings.

**JAWSEnd and JAWSHome**

The JAWSEnd and JAWSHome built-in functions perform special versions of their respective keyboard commands. Like the JAWSPageDown and JAWSPageUp functions, you can use these functions when the PC, JAWS, or Invisible cursor is active. However, the outcome of each function is based on the active cursor. When the PC cursor is active, each function performs the standard END and HOME commands for the application. For example, when you press END from within a word processor, the PC cursor is moved to the end of the current line. Likewise, when you press HOME from within a word processor, the PC cursor is moved to the beginning of the current line.

Example 3: Using the JAWSHome function with the PC cursor

PCCursor (); activate the PC cursor

SaveCursor (); save the location of the cursor

JAWSPageDown (); move the PC cursor down one screen

JAWSHome (); move the PC cursor to the beginning of the line

In the above example, the PC cursor is activated, saved, and moved down one screen. The JAWSHome function then moves the cursor to the beginning of the line. When you execute this block of code, the PC cursor is limited to the active window. When either the JAWS or Invisible cursor is active, the JAWSEnd function moves the cursor to the last text character or graphic on the line within the active window. Likewise, the JAWSHome function moves the cursor to the first text character or graphic on the current line within the active window. The active window is based on the cursor restriction setting.

Example 4: Using the JAWSHome function with the Invisible cursor

InvisibleCursor (); activate the Invisible cursor

SaveCursor (); save the location of the invisible cursor

RouteInvisibleToPC (); move the Invisible cursor to the location of the PC cursor

JAWSPageDown (); move the Invisible cursor to the bottom of the screen

JAWSHome (); move the Invisible cursor to the beginning of the current line

In the above example, the Invisible cursor is activated, saved, moved to the location of the PC cursor and moved to the bottom of the screen. Next, the JAWSHome function moves the Invisible cursor to the beginning of the current line. By moving the Invisible cursor to the beginning of the current line, you now have a distinct starting point. You can then use a number of other built-in movement functions to move the Invisible cursor across the line to read a specific piece of information such as a line number.

**Next and Prior Cursor Movement**

You can also use built-in functions to move to the next or prior piece of information. Many of these functions also have equivalent keyboard commands like the JAWSEnd, JAWSHome, JAWSPageDown, and JAWSPageUp functions. You can use these functions with any of the three cursors. However, the outcome of the function is determined by the active cursor. For example, if you move the JAWS cursor to the title bar of the active application, you can continually press INSERT+RIGHT ARROW to move across the title bar a word or graphic at a time. When you continually press the keystroke, the JAWS cursor does not stop at the end of the line. Rather, it continues to move on to the next line until you reach the bottom of the window. On the other hand, you can only move the PC cursor within the document edit area of a word processor when you continually press INSERT+RIGHT ARROW. Once you reach the bottom of the document edit window, the PC cursor can move no further.

Note: When you continually press INSERT+RIGHT ARROW, JAWS activates the SayNextWord script. The script must determine how to read the next word based on the active window. When you use any of the built-in functions to move to the next item, the active cursor is only moved to that item, but, does not speak it. The following list of functions moves the active cursor based on the name of the function. For example, the NextWord function moves the active cursor to the next word. The NextSentence function moves the active cursor to the next sentence and so on.

NextCharacter ()

NextLine ()

NextParagraph ()

NextSentence ()

NextWord ()

PriorCharacter ()

PriorLine ()

PriorParagraph ()

PriorSentence ()

PriorWord ()

Example 5: Using the NextWord function

InvisibleCursor (); activate the invisible cursor

SaveCursor (); save the location of the Invisible cursor

RouteInvisibleToPC (); move the invisible cursor to the location of the PC cursor

JAWSPageDown (); move the Invisible cursor to the bottom of the screen

JAWSHome (); move the Invisible cursor to the beginning of the current line

NextWord (); move the Invisible cursor to the next word or graphic In the above example, the Invisible cursor is activated, saved, and then routed to the location of the PC cursor. Next, the invisible cursor is moved to the bottom of the screen and then to the beginning of the current line. The NextWord function then moves the Invisible cursor to the next word or graphic on that line.

**6.7 SPEAKING THE INFORMATION**

As you learned in the previous section, the movement functions built into JAWS only move the cursor, but do not speak any information. You must use another built-in function to actually speak the information found at the active cursor location. In section 6.1 you used a number of keystrokes to move the JAWS cursor to read the bottom line of the active window. You used INSERT+UP ARROW to read the contents of the bottom line of the window as the last step. If you were to look at the actual code of the SayLine script, you would see the script uses a built-in function, SayLine, to read the current line. You can use built-in functions to read the current character, word, line, and more. You can also use built-in functions to read information from the cursor location to either the beginning of the line or the end of the line. You can use the functions listed below to read a specific piece of information. For example, the SayCharacter function speaks the current character while the SayWord function speaks the current word. The list of functions follows:

SayCharacter ()

SayChunk ()

SayField ()

SayLine ()

SayParagraph ()

SaySentence ()

SayWord ()

The functions listed below all speak information beginning at the location of the active cursor:

SayFromCursor ()

SayToCursor ()

SayToBottom ()

SayToPunctuation ()

Note: You can find more information for any of the functions listed above by reviewing the documentation file for functions built into JAWS, builtin.jsd. You can view this file in the Script Manager or in any text editor. You can find this file in the JAWS shared settings folder. Navigate to the JAWS program group in the Start menu and choose Explore JAWS > Explore Shared Settings. This opens a Windows Explorer screen where you can find all of the original JAWS script files.

**6.8 CHAPTER EXERCISES**

These exercises give you some practice in creating scripts that read pieces of information.

Exercise 6.1: Reading the Script Manager Status Line

The objective of this exercise is to use built-in cursor movement and reading functions to read the status line in the Script Manager.

Press INSERT+F2 to activate the Run JAWS Managers dialog.

Press S to select the Script Manager followed by ENTER.

Since this script should work in Script Manager, you need to open the script file for the manager. Choose Open Shared File from the File menu or press ALT+F followed by the letter E.

Type the file name Script Manager and press ENTER. You don’t need to include the file extension for script source files, .jss, because this is the default file type.

Press CTRL+END to move the insertion point to the bottom of the file. Press ENTER to create a blank line between the last script and your new script.

Press CTRL+E to activate the New Script dialog.

Type ReadBottomLine in the Script Name edit box. Be sure to capitalize the first letter of each word in the name of the script.

Press TAB to move to the Can be Attached to Key check box. Press SPACEBAR to check the check box.

Press TAB to move to the Synopsis edit box. Type "Reads the status line in the Script Manager" without the quotation marks.

Press TAB to move to the Description edit box. Type "Reads the status line in Script Manager using various cursor movement and reading functions" without the quotation marks.

Press TAB to move to the Category edit combo box. You do not need to enter any information in this combo box as it is not used.

Press TAB to the Assign to Key edit box. Press CTRL+SHIFT+L. This is the keystroke that activates the script.

Press ENTER to close the New Script dialog and return to the Script Manager window.

Press UP ARROW once to move to the first blank line of the ReadBottomLine script.

Press CTRL+I to activate the Insert Function dialog. You are looking for the function to save the current cursor. Type Save in the Function Name edit box. Did JAWS find the SaveCursor function? If so, then press ENTER to insert the function into your script and close the Insert Function dialog.

After you have inserted the SaveCursor function into your script, the insertion point is at the end of the line. Press ENTER to move to the next line.

Press CTRL+Ito activate the Insert Function dialog. You now need to locate the InvisibleCursor function. This will activate the Invisible cursor which you will use to read the status line of the window. Find the InvisibleCursor function and press ENTER to insert it.

Press ENTER to complete the line and move to a new blank line.

Follow the same procedure found in step 17 to insert the following functions into your script:

RouteInvisibleToPC

JAWSPageDown

SayLine

After following these steps, the script should look like the following:

Script ReadBottomLine ()

SaveCursor ()

InvisibleCursor ()

SaveCursor ()

RouteInvisibleToPC ()

JAWSPageDown ()

SayLine ()

EndScript

If your script looks like the one above, press CTRL+S to save and compile the script file. Did you hear the compile complete message? If so, then you are ready to test the new script. If not, then go back to the script and make sure it looks like the one above then compile your script file a second time.

Press CTRL+SHIFT+L to test your script. Did you hear the status line of the Script Manager spoken by JAWS?

Exercise 6.2: Read the Line Number and Total Lines in the Script Manager Status Line

The objective of this exercise is to create a script that reads only the line number and total number of lines within the status line. In the previous exercise, you created a script that read the entire Script Manager status line. Create a script that only speaks the line number and total number of lines in the script file. On the script manager status line there is more information than you necessarily want to hear, for example, "For help, F1". Your script should only say what you want to hear, which is the word "line" followed by the total number of actual lines in the script file. If the Script Manager is not the active application, press ALT+TAB until the Script Manager becomes the active window. The Script Manager.jss script file should also be opened within the manager. You can verify the file name by reading the title of the window with INSERT+T. Before you create the script, follow the steps below to determine what functions are needed:

Press NUM PAD MINUS twice in succession to activate the Invisible cursor.

Press INSERT+NUM PAD MINUS to route the Invisible cursor to the PC cursor location.

Press PAGE DOWN to move the Invisible cursor to the bottom of the window.

Press HOME to make sure the Invisible cursor is at the beginning of the status line.

Press INSERT+RIGHT ARROW until JAWS speaks "Line:".

Press INSERT+PAGE UP to read from the location of the Invisible cursor to the end of the line.

After you followed the steps above, did JAWS speak the line number and total number of lines? Your new script should also read only the line number and total number of lines within the current script file. The steps needed to create the script follow:

Press CTRL+END to move to the bottom of the script file.

Press ENTER twice to insert a blank line between the last script and your new script.

Press CTRL+E to activate the New Script dialog.

Fill in the script information fields as follows:

Script Name: ReadLineNumber

Can be Attached to Key: checked

Synopsis: Reads the line number and total number of lines.

Description: Reads the line number and total number of lines from the status line.

Category: not used

Assign To: CTRL+SHIFT+S

After you have entered all the script information, press ENTER to close the New Script dialog and return to the Script Manager edit area.

Press UP ARROW once to move to the first blank line in the body of the script.

Use the Insert Function dialog, activated by CTRL+I, to insert the following functions into your script:

SaveCursor ()

InvisibleCursor ()

SaveCursor ()

RouteInvisibleToPC ()

JAWSPageDown ()

JAWSHome ()

NextWord ()

NextWord ()

NextWord ()

NextWord ()

NextWord ()

SayFromCursor ()

Note: If you need to review how to select a function from the Insert Function dialog, see 5.5 Inserting Functions for more information. You should press ENTER after each function is inserted into your script. Failure to do so, causes each of the functions to be inserted on the same line.

After you have followed all the steps above, your script should look like the following:

Script ReadLineNumber ()

SaveCursor ()

InvisibleCursor ()

SaveCursor ()

RouteInvisibleToPC ()

JAWSPageDown ()

JAWSHome ()

NextWord ()

NextWord ()

NextWord ()

NextWord ()

NextWord ()

SayFromCursor ()

EndScript

After you have completed the steps above, Compare your script to the script shown above. Make any corrections to your script as necessary.

Note: When correcting your script, you can use normal editing techniques just as you would in a word processor or text editor.

Now that you have reviewed your script, press CTRL+S to save and compile your script. Did you hear the compile complete message? If so, then you are ready to test your script. If not, then note the error shown in the Error dialog. Next, press ESCAPE to close the Error dialog and return to your script. Review your script for the error and correct it. You can refer back to the script shown above, if necessary.

Once you have a complete compile, press CTRL+SHIFT+S to activate your script. Did you only hear the line number and total number of lines spoken by JAWS?

**7.0 USING VARIABLES AND CONSTANTS**

In previous chapters, you created scripts that spoke messages using hard coded strings of text. You used the SayString and Say built-in functions to speak those strings of text by typing the text directly into the parentheses of the function. You had to surround the text with quotation marks because each function expected a string of text as its parameter. Instead of using numeric values to represent the output type in the Say function, you used constants to represent those values. You can find all of the output type constants in the default constant file, HJConst.jsh. In this chapter, you will learn how to declare and use variables. You will learn the differences between local and global variables as well. You will also learn how to declare your own constant values and then use them in your scripts.

**TABLE OF CONTENTS**

Chapter 7.0 Using Variables and Constants, contains the following sections:

7.1 Variables

7.2 Constants

**7.1 VARIABLES**

You can think of a variable in terms of a bucket you may have at home. You can put a number of different things into the bucket. For example, you may use the bucket to carry water, sand, or gravel. As you pour out the contents of the bucket, you are changing the amount of sand, water, or gravel in the bucket. You can also put varying amounts of each of those items in the bucket. Sometimes you may only need to fill the bucket half full of water. Another time, you may need to completely fill the bucket with water. Like a bucket, a variable is an entity that holds a value. You can change the value the variable holds during the execution of your script. Conditions that occur in your script can also change the value stored in your variables. Each variable you use in your script or function is of a specific type and can only store one type of information.

You must declare all variables with a name and a type before you can use them within your script or function. You must give each variable a distinct name to distinguish it from other functions and names used in the script or script file. You cannot use duplicate variable names as this can lead to errors when you compile your script file.

Variable Types. You can use one of four variable types within your scripts and functions. You store only one specific type of information in each of these variable types. Variable types include:

integer

string

handle

object

Integer Variables. You can use the integer variable type to store numeric values. The numeric value you store in an integer must be a whole number such as 0, 10, or 100. You cannot store numbers with decimal points in integer variables. JAWS gives all integer variables an initial value of zero each time you activate the script in which the variables are declared. You must declare an integer variable before you can use it within your script or function. You can declare an integer variable as follows:

Int MyIntegerVariable

String Variables.

You can use the string variable type to store a string of characters. You can think of a string as a group of characters including letters, numbers, punctuation marks, and spaces. When you assign a value to a string variable, you must enclose the characters in quotation marks. JAWS gives all string variables an initial value of null or no value each time you activate the script in which the variables are declared. You can represent null values as a pair of quotation marks with no spaces between them or with the constant, cscNull, found in the Common.jsm default message file. You must declare a string variable before you can use it in your script or function. You can declare a string variable as follows:

String MyStringVariable

Handle Variables

You can use the handle variable type to store a window handle value. The operating system automatically assigns a window handle to each window within any running application. The window handle is a unique number that changes each time you close and then reopen an application. For example, when you start Microsoft Word, the document edit window may have a window handle value of 1000. When you close Word and then reopen it, the window handle value is no longer 1000. The new value may be 1100, 3000 or any other arbitrary number. A handle is also a whole number like an integer and can be manipulated like other integer variables. However, you can only use handle variables solely for the identification of window handles. In other words, you cannot store a handle value in an integer variable. JAWS gives all handle integer variables an initial value of zero each time you activate the script in which the variables are declared. You must declare a handle variable before you can use it in your script or function. You declare a handle variable as follows:

Handle MyHandleVariable

Object Variables

The forth and final variable type you can use is the object variable type. You use the object variable type to store an object. An object refers to the types of objects used within certain Microsoft applications such as the Office suite. JAWS gives all object variables an initial value of null or no value each time you activate the script in which the variable is declared. You must declare an object variable before you can use it in your script or function. You declare an object variable as follows:

Object MyObjectVariable

Naming Variables

As you begin to use variables within your scripts, there are a few ways you can name them to easily identify their contents. When you use multiple words for variable names, you should capitalize the initial letter in each word of the variable name. When you follow this rule of thumb, JAWS uses mixed case processing to pronounce each word beginning with a capital letter as a new word. This makes it much easier to understand the name of the variable. When the first letter of each word in a variable name is not capitalized, JAWS pronounces the variable as a single word. The following examples illustrate the use of variables with multiple word names. The first example uses initial capital letters in each word while the second example does not.

MyFirstVariable

Myfirstvariable

As you used JAWS to read the previous examples, did you notice a difference in the way the variable names were pronounced? You should also give your variables meaningful names. If you give a string variable the name of "i", you can't tell what type of information the variable stores. However, when you use a name of address or message for the name of a string variable the contents are easy to identify. You can also use Hungarian notation to name your variables. Hungarian notation uses lower-case letters at the beginning of variable names to indicate the variable type. For example, you could name a string variable that contains a first name sFirstName. The lower case "s" indicates the variable is of type string. The "FirstName" portion of the variable name indicates the variable contains first name information. Other types of Hungarian notation include "I" to indicate integer variables, "h" indicating handle variables and "o" for indicating object variables.

**Declaring Variables**

You can declare variables in one of two ways, locally or globally. When you declare variables locally, you can only use those variables in the script where you declared them. When you declare a group of variables globally you can use those variables in any script within the script file where you declare them.

**Declaring Variables Locally**

You declare local variables within a script immediately following the beginning line of the script. You start the beginning of a local variable declaration with the key word Var. You should declare each variable on a separate line. When you declare more than one variable, you must place a comma immediately after each variable name, except for the last variable. A good idea is to add a comment at the end of the declaration stating how the variable is used. As stated previously, you can only use local variables within the script in which you declare them. An example of a local variable declaration is as follows:

Var

string sFirstName, ; describe variable here

string sLastName ; describe variable here

**Declaring Global Variables**

You declare global variables within a script file immediately following any include statements. You start the beginning of a global variable declaration with the key word Globals. You should declare each variable on a separate line. When you declare more than one variable, you must place a comma immediately after each variable name, except for the last variable. A good idea is to add a comment at the end of the declaration stating how the variable is used. After you have created a global variable and assigned it a value, the global variable retains the value even after you close the application that uses it. If you open the application during the same computer session, the global variables from that application's script file will have the same values they had prior to when you closed the application. You can only clear the values stored in global variables by unloading and restarting JAWS. An example of a global variable declaration is as follows:

Globals

string sFirstName, ; describe variable here

string sLastName ; describe variable here

You can also declare global variables in a script header file. When you declare global variables in a script header file, you must include the file in any script file in which you will use those variables. You can find an example of a header file that contains global variable declarations by viewing the default global file, HJGlobal.jsh. You can find this file in your JAWS shared settings folder. The HJGlobal.jsh file is included in the default script file, default.jss. Including the HJGlobal.jsh header file in a default script file allows the global variables to be used in any of the default scripts and user-defined functions. When you include the HJGlobal.jsh file within an application-specific script file, you can use these variables to get specific information as needed. However, you should exercise caution when using any global variables from the HJGlobal.jsh file. The default scripts and user-defined functions often modify the values contained in these variables. When you modify the values of the variables within your own scripts, you may encounter unwanted results in your script file.

**Assigning Values**

You can use the let statement to assign values to any one of the 4 variable types. For example, let iControl = 1500 assigns the value of 1500 to the variable, iControl. Beginning in JAWS version 11 the let statement is now optional and is not required. You can assign a value yourself or you can assign the return value from a built-in function to a variable. When you assign a return value from a function to a variable, you must make sure the variable types are the same. For example, the GetFocus built-in function returns a window handle. You need to use a handle variable to store this return value. Otherwise, the script compiler will generate a syntax error when you attempt to compile your script. The statement you would use to assign the return value from the GetFocus function to a handle variable is:

let hWnd = GetFocus ()

Other examples of assigning values to variables using the let statement follow:

let sCompanyName = "Freedom Scientific"

sMessage = "Hello world!"

let iControl= GetControlID (GetFocus ())

**7.2 CONSTANTS**

Constants differ from variables in that they do not change value. You can use constants as a way of using easily-remembered names to store hard to remember strings of letters or numbers. You can think of a constant as a number or group of letters or words that you give or assign a new name. You create constants purely for mnemonic reasons since it is easier to remember a name than a number or long group of letters or words. JAWS has no restrictions to how many constant names you can assign to a number. For example, in the default JAWS constants file, HJConst.jsh, the script developers at Freedom Scientific have assigned both of the constants True and On the same value of 1. Since both True and On represent the value of 1, you can write a script statement to check a condition for values of True, On, or 1. JAWS considers all of these constants as having the same value once you declare the constants. There are several advantages to using constants in your scripts. First, you may find it more difficult to remember that when something has a value of 1 then it is also true. It is also easier to understand the difference between true and false. Second, when someone else is reading your script file, it is much easier for them to understand what you were trying to do if they see well-named constants instead of numbers. Finally, if you need to change the value of a number that is used in several places, you can change all of them at once if you have used a constant to represent that number. You need only change the value of the constant declaration.

**Declaring Constants**

You can declare constants at the beginning of a script file. You begin the constant declaration with the key word, Const. You follow each constant declaration with a comma except for the last entry. An example of a constant declaration follows:

Const

True = 1,

False = 0,

On = 1,

Off = 0

You can also declare constants in a script header file. You can create your own script header file to store your constant definitions. When you create a file to store your constant declarations, don't forget to include the file in any script file in which you want to use those constants. Since many of the built-in functions accept constant values as parameters, you can also use the constants defined in the default constant file, HJConst.jsh. This file is also included in the default script file, default.jss. Although you can use the constants declared in this file in your own script files, you should not add your own constant declarations to this file.

**8.0 CREATING AND SPEAKING MESSAGES**

In this chapter, you will learn how to create messages using the JAWS message format. You will learn how to create your messages within a JAWS script message, (.jsm) file. You will also learn how to use long and short messages to honor your verbosity settings. In the previous chapters, you created scripts using built-in speaking functions that used hard-coded messages. A hard-coded message is a string of text typed directly in a function as a parameter. Each time you need to change the contents of the message, you must search the entire file for the message. Once you have located the message, you have to modify the message being careful not to forget the quotation marks surrounding the message. You have to do this with each message contained in your script file. Rather than using hard-coded messages, you can store all of your messages in a JAWS script message file (.jsm). This makes it easier to find all of your messages as they are all in one place. If you ever need to modify one or more of the messages, you need only to go to the message file and make the change once. You can include your message in any number of script files using the include statement. When you make changes to any of the messages in your message file, you must recompile each script file that includes the message file. You must take this extra step so that JAWS sees the changes to your messages.

**Table of Contents**

Chapter 8.0 Creating and Speaking Messages, contains the following sections:

8.1 JAWS Message Constants

8.2 The JAWS Message Format

8.3 Formatting Your Messages

8.4 Speaking Your Messages

8.5 Chapter Exercises

**8.1 JAWS MESSAGE CONSTANTS**

In prior versions of JAWS, you had to create all of your messages as constants. A constant is a mnemonic way of representing a group of hard-to-remember values. Some examples of message constants follow:

Const

msgSettingSaved\_S = "Setting Saved"

msgNoFollowPC\_L = "The Jaws cursor will not follow the PC Cursor"

msgNoFollowPC\_S = "Jaws will not follow PC"

msgFollowPC\_L = "The Jaws cursor will follow the PC Cursor"

You can find the above examples in the default.jsm message file located in your JAWS settings folder. Each message ends with either an "S" or "L". Messages ending in "S" indicate the message is a short message. By default, JAWS speaks short messages when you use intermediate or advanced verbosity settings. The messages ending in "L" indicate long messages. JAWS speaks these messages when you use beginner verbosity setting. If you needed to make changes in any of these message constants, you would have to make sure the quotation marks are in place after the modifications are complete. Failure to do so causes compilation errors when you try to compile any script file that includes the default.jsm message file. Each time you wanted JAWS to speak multiple messages; you had to concatenate the messages together to produce one message. If you remember using hot key help in JAWS 3.7, the information spoken by JAWS was lengthy. The keystrokes spoken by JAWS were a series of message constants concatenated together to make the entire list of keystrokes. An example of string concatenation follows:

"This is the first part of my message." + " I am using concatenation to create a single message " + "out of 2 or 3 messages."

The result of concatenating the 3 strings of text follows:

"This is the first part of my message. I am using concatenation to create a single message out of 2 or 3 messages."

When you use string concatenation to create a single message, you also have to remember to use proper spacing. Otherwise, the words of your message may run together to create an unintelligible message.

**8.2 The JAWS Message Format**

The JAWS message format was first introduced in JAWS 4.0. This new message format replaced the need for concatenating message constants together to produce a single message. The message format also makes modifying your messages much easier. You do not have to worry about surrounding the entire message text in quotation marks. The JAWS message format consists of the following parts:

Messages

@MsgName

message body

@@

EndMessages

**Beginning the Block of Messages**

You use the key word, Messages, to begin the entire block of messages. You place this key word at the beginning of your message file. You only use this key word once within the message file. This key word tells JAWS that anything following the word is part of your messages block.

Beginning an Individual Message. All of your individual messages immediately follow the Messages key word. You start each individual message with the @ symbol followed by the name of the message. When you name your messages, it is often helpful to begin the name with "Msg." When you start each message in this way, you will know immediately where the message is stored when you reference the message in your script file. You can think of the message name as a constant. Once you have named an individual message, you will most likely not modify the message content during the execution of your scripts. When you reference your message from within a script, you do not include the @ symbol. This symbol only tells JAWS that the text immediately following the symbol is the actual name of the message. For example, if you have messages defined in your message file as @Msg1 and @Msg2, you would reference those messages using the Say function as follows:

Say (Msg1, OT\_MESSAGE)

Say (Msg2, OT\_MESSAGE)

In the above examples, only the actual message name is used as the parameter for the Say function.

**The Body of the Message**

The text to be spoken by JAWS immediately follows the @MsgName statement. This is the body of the individual message. The body of the message can contain any text or punctuation. You can type anything inside the body of the message that you would in a word processor or text editor. You can press ENTER to create a new line and do not have to use escape characters to represent certain symbols such as the backslash. The following example shows the message name and the text contained in the body of a message:

@MsgName

This text is part of the body of a message. Notice it is written just as if you typed it into a text editor or word processor. When you press the keystroke to activate the script that uses this message, JAWS speaks this text.

Message Placeholders

You can also use placeholders within the body of your message. These placeholders essentially hold a place in your message for information you want to add to the message during the execution of your script or function. You can use up to nine placeholders within the body of the message. To add a placeholder to your message, type the percent sign followed by a number from 1 to 9. You can find the following examples of the use of placeholders in the common.jsm message file:

;for cMsgMenuItem\_L/S, %1=app title, %2=current menu selection, this is used in SayWindowTitle

@cMsgMenuItem\_L

title= %1 menu

%2

@@

@cMsgMenuItem\_S

%1

menu

%2

@@

The message referenced by MsgMenuItem\_L shown above is heard when you press INSERT+T to read the title of an application and a menu is open. The placeholders allow for changes in the menu name and the selected item in the menu. This message is heard when you are using beginner verbosity. The message referenced by MsgMenuItem\_S is heard when you are using intermediate or advanced verbosity.

**Ending the Message**

After you have completed the body of the message, you need to end the individual message. You type 2 @ symbols "@@" to tell JAWS that this is the end of your individual message. You should type the @ symbols together without spaces between them. The @@ symbols should go on their own line following the individual message. The following example shows a complete individual message:

@MsgName

This text is part of the body of a message. Notice it is written just as if you typed it into a text editor or word processor. When you press the keystroke to activate the script that uses this message, JAWS speaks this text.

@@

**Ending the Messages Block**

After you have entered all of your messages into your message file, you must end the messages block. The key word EndMessages tells JAWS this is the end of all of your messages. You only type this key word once in your message file. It must follow all individual messages. There is no space separating the words of "End" and "Messages." The following example shows a complete messages block:

Messages

@MsgName

This text is part of the body of a message. Notice it is written just as if you typed it into a text editor or word processor. When you press the keystroke to activate the script that uses this message, JAWS speaks this text.

@@

EndMessages

**Adding Comments**

You can add comments at the top of your message file or within the messages block. Just as in your script file, you begin all comments with a ";" semi colon. When you add comments inside the messages block, make sure you have not added the comments in the body of an individual message. Even though you begin a comment line with a semi colon, JAWS will think it is just part of the body of your message. Thus, the comment will be treated just as if it is part of the message. An example of comments within a script message file along with an message block follows:

Example 1: Full Message Block Example

; JAWS 12.0.512 Message file for Notepad in Windows Vista

; created by the Freedom Scientific Training Department

Messages

; my first practice message

@MsgName

This text is part of the body of a message. Notice it is written just as if you typed it into a text editor or word processor. When you press the keystroke to activate the script that uses this message, JAWS speaks this text.

@@

EndMessages

In the above example, the comment within the message block is placed before the start of the

**individual message.**

**8.3 FORMATTING YOUR MESSAGES**

If you have used placeholders in your message, you can use the built-in function, FormatString, to replace those placeholders with information. You provide the message as the first parameter of the function. The subsequent parameters are used for the information that replaces the placeholders in your message. If you have only used two placeholders, then you need only provide information for the next two parameters following the message name. The function formats the message then returns the formatted message to the script or function as a string of text. You can store this formatted message in a string variable for use in some of the speaking functions such as SayFormattedMessage. The following example shows the use of the FormatString function with a message that contains placeholders:

Example 1: Using the FormatString Function

@MsgMyName

Hello, my name is %1 %2. You can just call me %1.

@@

The call to the FormatString function looks like the following:

let sMessage = FormatString (MsgMyName, "John", "Doe")

The text the FormatString function returns follows:

Hello, my name is John Doe. You can just call me John.

NOTE: Beginning in JAWS 12 update 1, the SayFormattedMessage function allows you to substitute place holders directly. To speak the message using this function, you no longer have to use FormatString first.

**8.4 SPEAKING YOUR MESSAGES**

After you have created your messages, you can use one of the built-in speaking functions to speak the message. Your choices are:

Say

SayMessage

SayFormattedMessage

You will find other functions that speak messages such as SayUsingVoice, but the three listed above are the most widely used. The Say built-in function does offer the flexibility of allowing for user verbosity preferences about types of messages spoken by JAWS. However, the function does not honor your message length verbosity setting. In other words, the function does not allow for short and long messages. For example, pressing CTRL+C to copy text to the clipboard while using beginner verbosity causes JAWS to speak "Copied selection to clipboard." When you press the same keystroke to copy text using intermediate verbosity, JAWS speaks "copied." The SayMessage and SayFormattedMessage built-in functions not only allow for output type considerations, but also allow for short and long messages. Of the 2 functions, the SayFormattedMessage function is the newest and is more widely used by script developers at Freedom Scientific. Therefore, you should always use the SayFormattedMessage function when speaking messages in your scripts.

Using the SayFormattedMessage Function. When you use the SayFormattedMessage function to speak messages in your scripts, you must provide two parameters; output type and long message. A third parameter, the short message, is an optional parameter along with nine additional parameters for substituting place holder values. You should always try to provide a short message when you are speaking messages to account for both long and short message length settings.

The first parameter is an output type constant. This output type constant represents a numeric value that indicates the message type. JAWS uses this value to determine how and when to speak the message. You can find a list of all output type constants in the default constant file, HJConst.jsh. Output type values range from screen messages, help messages, smart help messages, and more. You should determine what type of output type to use based on the context of your message. For example, when you create a help message that is spoken from a script, you should choose the OT\_ HELP output type constant. This constant value tells JAWS this is a help message and it should be spoken accordingly. Not only should you consider the context of the message, but you should also consider the possibility of the message being turned off from within the Settings Center. You do not want a critical error message turned off. You can use the output type constant, OT\_NO\_DISABLE, to tell JAWS to always speak the message. When you use this constant value, JAWS will speak the message regardless of the verbosity setting in use. You provide the long message in the second parameter of the SayFormattedMessage function. JAWS speaks this message when you are using the beginner verbosity setting. When you create this message in your message file, you can append the text of "\_L" to indicate the long message. For example, you could use the following message names to indicate long messages:

MsgMyName\_L

MsgCopiedText\_L

When you add the "\_L" to the end of your message name, you can easily identify the message in your script file as being a long message. You provide the short message as the third parameter of the SayFormattedMessage function. JAWS speaks this message when you use the intermediate or advanced verbosity setting. You can indicate the short message in your message file in much the same way as the long message. You can append "\_S" to the end of the message name. For example, the short versions of the previous long message examples would be:

MsgMyName\_S

MsgCopiedText\_S

When you add "\_S" to the end of your short messages, you can easily identify them in your script files.

Note: The short message parameter is optional. If you only want to speak one message regardless of the message length setting, do not provide a message in this parameter. You can also use the message constant value of cmsgSilent. You can find this message in the common.jsm message file located in your JAWS shared settings folder. The fourth through the twelfth parameters are optional and should be used to substitute values for percent place holders you may have used in your message. For example, the fourth parameter of SayFormattedMessage should be the value that will replace %1 in your message. These values can be quoted strings of text, constants, variables, or functions that return string values. If you did not use percent place holders in your message you should not use these parameters.

**8.5 CHAPTER EXERCISES**

The following exercises give you practice in creating messages. You will also use the FormatString and SayFormattedMessage built-in functions in these exercises.

Exercise 8.1: Creating a Message File and Adding Messages

The objective of this exercise is to give you practice creating a new message file and adding messages to the file. You will use Notepad for the active application. You will add two messages to the notepad.jsm file. After you have created the messages, you will create a script that speaks those messages.

Activate Notepad from your Start Menu | Accessories group.

From within Notepad, press INSERT+F2 to activate the Run JAWS Managers dialog. Press S followed by ENTER to start the Script Manager.

You need to create a message file for Notepad as it does not have a message file by default. Press CTRL+N to activate the New File dialog. You are placed in the New list box. The list box contains five entries for creating files. You can use this list box to create a new source file, header file, documentation file, message file, or key map file.

Press M to select the Messages file (jsm) list item followed by ENTER to create the new message file. When the New File dialog closes, your insertion point is placed in a new untitled message file.

Press CTRL+S to save the file. This action causes the Script Manager to display the Save As dialog. The focus is in the File Name edit box. Give the new file the name of Notepad. You do not need to include the file extension as the script Manager does that automatically for you.

After the Save As dialog closes, press INSERT+T to read the title bar of the Script Manager. Did you hear the file name of Notepad.jsm?

Note: When you create script message or header files, give them the same name as the executable file name for the application. This makes it easier for you to locate the message file for a given application should you decide to share the file with someone else. If you are creating a generic message or header file, one not tied to a specific application, you can give it any name you want.

You are now ready to add a comment to the top of the file along with the key word used to indicate the beginning of your messages block. Type the following text into your message file:

; Message file for Notepad

Messages

Next, you need to add a long message to the message file. If the insertion point is still on the line containing the key word "messages," press ENTER to create a new blank line. Type the following text:

@MSGText\_L

This is an example of a long message. This message is only spoken by JAWS when the beginner verbosity setting is in use.

@@

Press ENTER twice on the line containing the two @ symbols. This will create a blank line to separate your long message from your short message.

The short message is next. Type the following text:

@MsgText\_S

This is an example of a short message.

@@

Press ENTER once to create a new blank line following your short message. Type the following text:

EndMessages

Press CTRL+S to save your message file.

Note: JAWS does not speak "Compile complete." The Script Manager does not compile message files only script source files.

After following the steps above, you should have the following text in the Notepad message file:

; Message file for Notepad

Messages

@MSGText\_L

This is an example of a long message. This message is only spoken by JAWS when the beginner verbosity setting is in use.

@@

@MsgText\_S

This is an example of a short message.

@@

EndMessages

Next, you need to create a script that uses the messages you just created. Press CTRL+TAB to move back to the Notepad.jss script source file.

Note: You can have more than one file open in the Script Manager at a time. Press CTRL+TAB to move between the files.

Press CTRL+HOME to move to the top of the Notepad script source file. You need to include the new message file. Press DOWN ARROW until you reach the blank line following the include statement for the hjconst.jsh file. Type the following text:

Include "Notepad.jsm"

After you have typed the text, press ENTER twice to create a new blank line.

Press CTRL+END to move the insertion point to the bottom of the script file. Press ENTER twice to create a blank line between the last script in the file and your new script.

Press CTRL+E to activate the New Script dialog and enter the following information in the fields in the dialog:

Script Name: SpeakMessage

Can be Attached to Key: checked

Synopsis: Speaks long and short messages in notepad.

Description: Speaks long and short messages in Notepad using the SayFormattedMessage function.

Category: none

Assign to: CTRL+3

After you have entered all the information above, press ENTER to close the dialog.

Press UP ARROW to move to the first blank line in the body of the SpeakMessage script.

Press CTRL+I to activate the Insert Function dialog. The focus is placed in the Function Name edit box. Look for the built-in function SayFormattedMessage. When you find the function, press ENTER. JAWS prompts you for the first parameter, output type. Type the following text:

OT\_MESSAGE

Press ENTER. JAWS prompts you for the second parameter, the long message. Type the following text in the Parameter 2 edit box:

MsgText\_l

Press ENTER. JAWS prompts you for the third parameter, the short message. Type the following text in the Parameter 3 edit box:

MsgText\_S

You do not need to enter any information for the remaining parameters after the short message name. Press TAB to move to the Finish button and press ENTER to close the Insert Function dialog. Make sure to remove the extra commas and spaces before the right parenthesis

Press DOWN ARROW to move to the next line. Press DELETE repeatedly until you have removed all blank lines from the script. After you have completed the previous steps, your script should look like the following:

Script SpeakMessage ()

SayFormattedMessage (OT\_MESSAGE, MsgText\_l, MsgText\_S)

EndScript

Press CTRL+S to compile the Notepad script source file. If you hear JAWS speak, compile complete, switch to Notepad and test your script. If you encountered syntax errors, compare your script to the one shown above. Continue to compile your script until you hear JAWS speak compile complete.

Hint: To test your script, use the Adjust JAWS Options dialog to change your verbosity setting from beginner to intermediate to advanced. You can access the Adjust JAWS Options dialog by pressing INSERT+V. After you have opened the Adjust JAWS Options dialog, press U to move to the User Verbosity option. Press SPACEBAR to cycle through the options. When you have selected the desired option, press ENTER to close the dialog and save the verbosity setting change. Each time you switch the verbosity setting, test your script. JAWS should only speak the long message when you are using the beginner verbosity setting. JAWS should speak the short message when you use either intermediate or advanced verbosity setting.

Exercise 8.2: Using Placeholders

The objective of this exercise is to give you practice using placeholders in your messages, formatting your messages, and speaking those messages. This exercise should be completed in the Notepad script source and message files. If you do not have Notepad or the Script Manager running, refer to steps 1 and 2 in the previous exercise to get started.

Within the script Manager, press CTRL+TAB until you reach the notepad.jsm message file.

Press CTRL+END to move to the bottom of the file.

Press UP ARROW until you reach the line containing "EndMessages." Press HOME to move the insertion point to the beginning of the line.

Press ENTER twice to create a blank line between the last individual message in the messages block and your new message.

Type the following text in the notepad message file:

@MsgMyName

Hello, my name is %1 %2. You can just call me %1.

@@

Press CTRL+S to save the changes to the Notepad message file.

Press CTRL+TAB to move to the Notepad script source file, Notepad.jss.

Press CTRL+END to move to the bottom of the file. Press ENTER twice to create a blank line between your last script and the new script.

Press CTRL+E to activate the New Script dialog. Type the following information into each control within the dialog:

Script Name: SpeakMyName

Can be Attached to Key: checked

Synopsis: Speaks a message containing my name.

Description: Speaks a message containing my name in Notepad. The message uses placeholders.

Category: none

Assign to: CTRL+SHIFT+N

After you have filled in all of the fields, press ENTER to close the New Script dialog.

Press UP ARROW to move the insertion point to the first blank line in the body of the script. You need to create a local variable declaration section here. Type the following text:

Var

string sMessage

After you have typed in the variable declaration statement for sMessage, press ENTER to move the insertion point to a new blank line. Type the following text:

let sMessage =

Note: Be sure to type a space after the equals sign. This allows for proper spacing when you insert the next function.

Press CTRL+I to activate the Insert Function dialog. You need to find the built-in function, FormatString. Once you have located the function, press ENTER to begin adding the parameters.

The first parameter the function requires is the name of the message that contains the text and placeholders. Type the following text:

MsgMyName

Press ENTER. JAWS accepts the first parameter and then prompts you for the second parameter.

The second parameter is the information that replaces the first placeholder in your message. Type your first name surrounded in quotation marks, for example "John."

Press ENTER to accept the parameter. JAWS prompts you for the third parameter. This parameter is the information that replaces the second placeholder within your message. Type your last name surrounded by quotation marks. For example, "Doe."

Instead of pressing ENTER to accept this parameter, move to and activate the Finish button. The finish button tells JAWS you are finished entering parameters for the function. After the Insert Function dialog closes, your insertion point is at the end of the line you just entered. Read the line with INSERT+UP ARROW. JAWS should speak the following text:

let sMessage = FormatString (MsgMyName, "John", "Doe", , , , , , , , )

You need to remove all the commas and spaces following the quotation mark following your last name. Press LEFT ARROW until you reach the comma immediately following the quotation mark. Press DELETE repeatedly to remove all commas and spaces until you reach the right parenthesis. The line should look like the following:

let sMessage = FormatString (MsgMyName, "John", "Doe")

Press END to move to the character following the right parenthesis. Press ENTER to complete the line. Your insertion point is now on a new blank line.

Press CTRL+I to activate the Insert Function dialog. You are looking for the built-in function, SayFormattedMessage. Once you have located the function, press ENTER.

JAWS prompts you for the first parameter for the SayFormattedMessage function. This parameter is the output mode. Type the following text:

**8.5 CHAPTER EXERCISES**

The following exercises give you practice in creating messages. You will also use the FormatString and SayFormattedMessage built-in functions in these exercises.

Exercise 8.1: Creating a Message File and Adding Messages

The objective of this exercise is to give you practice creating a new message file and adding messages to the file. You will use Notepad for the active application. You will add two messages to the notepad.jsm file. After you have created the messages, you will create a script that speaks those messages.

Activate Notepad from your Start Menu | Accessories group.

From within Notepad, press INSERT+F2 to activate the Run JAWS Managers dialog. Press S followed by ENTER to start the Script Manager.

You need to create a message file for Notepad as it does not have a message file by default. Press CTRL+N to activate the New File dialog. You are placed in the New list box. The list box contains five entries for creating files. You can use this list box to create a new source file, header file, documentation file, message file, or key map file.

Press M to select the Messages file (jsm) list item followed by ENTER to create the new message file. When the New File dialog closes, your insertion point is placed in a new untitled message file.

Press CTRL+S to save the file. This action causes the Script Manager to display the Save As dialog. The focus is in the File Name edit box. Give the new file the name of Notepad. You do not need to include the file extension as the script Manager does that automatically for you.

After the Save As dialog closes, press INSERT+T to read the title bar of the Script Manager. Did you hear the file name of Notepad.jsm?

Note: When you create script message or header files, give them the same name as the executable file name for the application. This makes it easier for you to locate the message file for a given application should you decide to share the file with someone else. If you are creating a generic message or header file, one not tied to a specific application, you can give it any name you want.

You are now ready to add a comment to the top of the file along with the key word used to indicate the beginning of your messages block. Type the following text into your message file:

; Message file for Notepad

Messages

Next, you need to add a long message to the message file. If the insertion point is still on the line containing the key word "messages," press ENTER to create a new blank line. Type the following text:

@MSGText\_L

This is an example of a long message. This message is only spoken by JAWS when the beginner verbosity setting is in use.

@@

Press ENTER twice on the line containing the two @ symbols. This will create a blank line to separate your long message from your short message.

The short message is next. Type the following text:

@MsgText\_S

This is an example of a short message.

@@

Press ENTER once to create a new blank line following your short message. Type the following text:

EndMessages

Press CTRL+S to save your message file.

Note: JAWS does not speak "Compile complete." The Script Manager does not compile message files only script source files.

After following the steps above, you should have the following text in the Notepad message file:

; Message file for Notepad

Messages

@MSGText\_L

This is an example of a long message. This message is only spoken by JAWS when the beginner verbosity setting is in use.

@@

@MsgText\_S

This is an example of a short message.

@@

EndMessages

Next, you need to create a script that uses the messages you just created. Press CTRL+TAB to move back to the Notepad.jss script source file.

Note: You can have more than one file open in the Script Manager at a time. Press CTRL+TAB to move between the files.

Press CTRL+HOME to move to the top of the Notepad script source file. You need to include the new message file. Press DOWN ARROW until you reach the blank line following the include statement for the hjconst.jsh file. Type the following text:

Include "Notepad.jsm"

After you have typed the text, press ENTER twice to create a new blank line.

Press CTRL+END to move the insertion point to the bottom of the script file. Press ENTER twice to create a blank line between the last script in the file and your new script.

Press CTRL+E to activate the New Script dialog and enter the following information in the fields in the dialog:

Script Name: SpeakMessage

Can be Attached to Key: checked

Synopsis: Speaks long and short messages in notepad.

Description: Speaks long and short messages in Notepad using the SayFormattedMessage function.

Category: none

Assign to: CTRL+3

After you have entered all the information above, press ENTER to close the dialog.

Press UP ARROW to move to the first blank line in the body of the SpeakMessage script.

Press CTRL+I to activate the Insert Function dialog. The focus is placed in the Function Name edit box. Look for the built-in function SayFormattedMessage. When you find the function, press ENTER. JAWS prompts you for the first parameter, output type. Type the following text:

OT\_MESSAGE

Press ENTER. JAWS prompts you for the second parameter, the long message. Type the following text in the Parameter 2 edit box:

MsgText\_l

Press ENTER. JAWS prompts you for the third parameter, the short message. Type the following text in the Parameter 3 edit box:

MsgText\_S

You do not need to enter any information for the remaining parameters after the short message name. Press TAB to move to the Finish button and press ENTER to close the Insert Function dialog. Make sure to remove the extra commas and spaces before the right parenthesis

Press DOWN ARROW to move to the next line. Press DELETE repeatedly until you have removed all blank lines from the script. After you have completed the previous steps, your script should look like the following:

Script SpeakMessage ()

SayFormattedMessage (OT\_MESSAGE, MsgText\_l, MsgText\_S)

EndScript

Press CTRL+S to compile the Notepad script source file. If you hear JAWS speak, compile complete, switch to Notepad and test your script. If you encountered syntax errors, compare your script to the one shown above. Continue to compile your script until you hear JAWS speak compile complete.

Hint: To test your script, use the Adjust JAWS Options dialog to change your verbosity setting from beginner to intermediate to advanced. You can access the Adjust JAWS Options dialog by pressing INSERT+V. After you have opened the Adjust JAWS Options dialog, press U to move to the User Verbosity option. Press SPACEBAR to cycle through the options. When you have selected the desired option, press ENTER to close the dialog and save the verbosity setting change. Each time you switch the verbosity setting, test your script. JAWS should only speak the long message when you are using the beginner verbosity setting. JAWS should speak the short message when you use either intermediate or advanced verbosity setting.

Exercise 8.2: Using Placeholders

The objective of this exercise is to give you practice using placeholders in your messages, formatting your messages, and speaking those messages. This exercise should be completed in the Notepad script source and message files. If you do not have Notepad or the Script Manager running, refer to steps 1 and 2 in the previous exercise to get started.

Within the script Manager, press CTRL+TAB until you reach the notepad.jsm message file.

Press CTRL+END to move to the bottom of the file.

Press UP ARROW until you reach the line containing "EndMessages." Press HOME to move the insertion point to the beginning of the line.

Press ENTER twice to create a blank line between the last individual message in the messages block and your new message.

Type the following text in the notepad message file:

@MsgMyName

Hello, my name is %1 %2. You can just call me %1.

@@

Press CTRL+S to save the changes to the Notepad message file.

Press CTRL+TAB to move to the Notepad script source file, Notepad.jss.

Press CTRL+END to move to the bottom of the file. Press ENTER twice to create a blank line between your last script and the new script.

Press CTRL+E to activate the New Script dialog. Type the following information into each control within the dialog:

Script Name: SpeakMyName

Can be Attached to Key: checked

Synopsis: Speaks a message containing my name.

Description: Speaks a message containing my name in Notepad. The message uses placeholders.

Category: none

Assign to: CTRL+SHIFT+N

After you have filled in all of the fields, press ENTER to close the New Script dialog.

Press UP ARROW to move the insertion point to the first blank line in the body of the script. You need to create a local variable declaration section here. Type the following text:

Var

string sMessage

After you have typed in the variable declaration statement for sMessage, press ENTER to move the insertion point to a new blank line. Type the following text:

let sMessage =

Note: Be sure to type a space after the equals sign. This allows for proper spacing when you insert the next function.

Press CTRL+I to activate the Insert Function dialog. You need to find the built-in function, FormatString. Once you have located the function, press ENTER to begin adding the parameters.

The first parameter the function requires is the name of the message that contains the text and placeholders. Type the following text:

MsgMyName

Press ENTER. JAWS accepts the first parameter and then prompts you for the second parameter.

The second parameter is the information that replaces the first placeholder in your message. Type your first name surrounded in quotation marks, for example "John."

Press ENTER to accept the parameter. JAWS prompts you for the third parameter. This parameter is the information that replaces the second placeholder within your message. Type your last name surrounded by quotation marks. For example, "Doe."

Instead of pressing ENTER to accept this parameter, move to and activate the Finish button. The finish button tells JAWS you are finished entering parameters for the function. After the Insert Function dialog closes, your insertion point is at the end of the line you just entered. Read the line with INSERT+UP ARROW. JAWS should speak the following text:

let sMessage = FormatString (MsgMyName, "John", "Doe", , , , , , , , )

You need to remove all the commas and spaces following the quotation mark following your last name. Press LEFT ARROW until you reach the comma immediately following the quotation mark. Press DELETE repeatedly to remove all commas and spaces until you reach the right parenthesis. The line should look like the following:

let sMessage = FormatString (MsgMyName, "John", "Doe")

Press END to move to the character following the right parenthesis. Press ENTER to complete the line. Your insertion point is now on a new blank line.

Press CTRL+I to activate the Insert Function dialog. You are looking for the built-in function, SayFormattedMessage. Once you have located the function, press ENTER.

JAWS prompts you for the first parameter for the SayFormattedMessage function. This parameter is the output mode. Type the following text:

OT\_MESSAGE

Press ENTER to accept the parameter. JAWS prompts you for the next parameter, the long message.

This parameter is the return value from the call to the FormatString function, sMessage. This is the message that has been formatted with your name replacing the placeholders. Type the following text:

sMessage

Press ENTER to accept the parameter. JAWS then prompts you for the third parameter. This is the short message. You do not need this parameter or any of the remaining parameters; so, activate the Finish button to close the Insert Function dialog.

Your insertion point is at the end of the line. Press INSERT+UP ARROW to read the current line. JAWS should speak:

SayFormattedMessage (OT\_MESSAGE, sMessage, , , , , , , , , , )

You need to remove the commas immediately following the sMessage parameter. Press LEFT ARROW until you reach a comma. Press DELETE until you remove the commas and spaces. The line should now look like the following:

SayFormattedMessage (OT\_MESSAGE, sMessage)

Press DOWN ARROW once to move to the next line. If this line is blank, press DELETE to remove it and any other blank lines.

After following the steps above, your script should look like the following:

Script SayMyName ()

Var

string sMessage

let sMessage = FormatString (MsgMyName, "John", "Doe")

SayFormattedMessage (OT\_MESSAGE, sMessage)

EndScript

Press CTRL+S to compile the Notepad script source file. If you did not hear JAWS speak "Compile Complete," return to your script. Correct the error the Script Manager noted and try compiling your script again. Continue this process until you hear "Compile complete."

Note: When the Script Manager encounters an error, the line containing the error is highlighted.

Return to Notepad and test your script. Did JAWS speak the message?

Using the SayFormattedMessage Function to Substitute Placeholders. New to JAWS 12 update 1, is the ability to replace placeholder values directly in the SayFormattedMessage function rather than having to use the FormatString function to do so. This eliminates the need for the sMessage variable and the FormatString function. If using this method, the script would look like the following.

Script SayMyName ()

SayFormattedMessage (OT\_MESSAGE, MsgMyName, MsgMyName, "John", "Doe")

EndScript

Exercise 8.3: Adding the InputBox Function to the Script Created in Exercise 8.2

The objective of this exercise is to modify the script created from exercise 8.2 above. Instead of using quoted strings of text as parameters for the FormatString function, use a built-in function to prompt for the first and last name values. You can use the built-in function, InputBox, to accomplish this task. The function displays a dialog box using the title and prompt you specify. When you type information into the edit box contained in the dialog and activate the Ok button, the function returns the text as a string. The InputBox function uses three parameters:

string of text for the dialog prompt

string of text for the title of the dialog box

variable to contain the text typed in the dialog. You should create two additional local variables. The first variable should store the first name. The second variable should contain the last name. When you have made the modifications to the script, it should look like the following:

Script SpeakMyName ()

Var

sMessage,

string sFirstName,

string sLastName

InputBox ("Enter your first name:", "First Name Entry", sFirstName)

InputBox ("Enter your last name:", "Last Name Entry", sLastName)

let sMessage = FormatString (MsgMyName, sFirstName, sLastName)

SayFormattedMessage (OT\_MESSAGE, sMessage)

EndScript

After you have modified the script, compile the script. When you hear the "Compile complete" message, return to Notepad, and test the script.

Note: For more information on the InputBox function, open the builtin.jsd script documentation file within the Script Manager

OT\_MESSAGE

Press ENTER to accept the parameter. JAWS prompts you for the next parameter, the long message.

This parameter is the return value from the call to the FormatString function, sMessage. This is the message that has been formatted with your name replacing the placeholders. Type the following text:

sMessage

Press ENTER to accept the parameter. JAWS then prompts you for the third parameter. This is the short message. You do not need this parameter or any of the remaining parameters; so, activate the Finish button to close the Insert Function dialog.

Your insertion point is at the end of the line. Press INSERT+UP ARROW to read the current line. JAWS should speak:

SayFormattedMessage (OT\_MESSAGE, sMessage, , , , , , , , , , )

You need to remove the commas immediately following the sMessage parameter. Press LEFT ARROW until you reach a comma. Press DELETE until you remove the commas and spaces. The line should now look like the following:

SayFormattedMessage (OT\_MESSAGE, sMessage)

Press DOWN ARROW once to move to the next line. If this line is blank, press DELETE to remove it and any other blank lines.

After following the steps above, your script should look like the following:

Script SayMyName ()

Var

string sMessage

let sMessage = FormatString (MsgMyName, "John", "Doe")

**9.0 CONTROLLING THE FLOW OF SCRIPTS WITH DECISION MAKING**

In the previous chapters, you created scripts that performed each line once and only once. When JAWS performs your scripts in this manner, each statement is performed regardless of any situations that may arise during the execution of the script. This manner of performing your scripts is called sequential flow control. This is the most basic way in which JAWS performs your scripts. JAWS performs each statement sequentially beginning with the first statement in your script and continuing to the EndScript statement. You can think of a script that flows sequentially as a script that JAWS performs step-by-step. In addition to creating scripts that JAWS performs sequentially, you can add statements to your scripts that perform an evaluation. This type of flow control is called branching, conditional, or decision-making. You may find that at some point during the execution of your script, the script must evaluate certain conditions before continuing. The conditions your script evaluates can be anything from the value contained within a variable to the return value from a function. Once your script completes the evaluation, the statements that are processed next are based on the outcome of the evaluation. When the outcome of the evaluation is true, processing may continue down one branch. If the outcome is found to be false, a second branch of statements is performed. For example, suppose you come to a group of two doors at the end of a hallway. Before you continue, you must evaluate which door to go through. Your decision will be based on a set of conditions. If it is dark outside, you might choose one door but if it is light out, you might choose another.

Your script may need to do that same type of decision making. You use the If-Then-Else statement structure to make a decision. When your script comes to the If-Then-Else statement a condition is evaluated. Based on the outcome of the evaluation, your script may perform one branch of code as opposed to another. In this chapter, you will learn how to create basic If statements to make decisions in your scripts. You will learn the structure of the If statement and the uses of the Else and ElIf key words. You will also learn how to create more complex If statements that use compound evaluations as well as nesting If statements inside one another. You will also learn how to create a loop that executes a section of code multiple times.

**Table of Contents**

Chapter 9.0 Controlling the Flow of Scripts with Decision Making, contains the following sections:

9.1 Logical Operators

9.2 The If Statement

9.3 Adding a Second Branch, the Else Keyword

9.4 Adding Multiple Branches, the ElIf Statement

9.5 Nested If Statements

9.6 Compound If Statements

9.7 Looping

9.8 Chapter Exercises

**9.1 LOGICAL OPERATORS**

You use logical operators to make comparisons of variables with other variables or constants. You frequently need to know if one variable has the same value, a lesser value, or a greater value than another variable. You generally use logical operators within If-Then statements and While loops to check whether conditions required for logical decisions are true or false. A list of the JAWS logical operators is shown below.

Operator

Description

==

Two equals signs placed together ask if two values are equal to each other. For example, the statement (A == B) is true when a is equal to b.

!=

An exclamation point and an equals sign placed together determine if two values are not equal to each other. That is, is the value or expression to the left of the! = sign not equal to the value or expression to the right? For example, the statement (A != b) is true if A does not equal B.

<

A < sign asks if the first value is less than the second value. That is, is the value or expression to the left of the < sign less than the value or expression to the right side? For example, the statement (A < B) is true if A is less than B.

<=

A < than sign placed together with an equals sign asks if the first value is less than or equal to the second value. That is, is the value or expression to the left of the <= sign less than or equal to the value or expression to the right side? For example, the statement (A <= B) is true if A is less than or equal to B.

>

A > sign asks if the first value is greater than the second value. That is, is the value or expression to the left of the > sign greater than the value or expression to the right side? For example, the statement (A > B) is true if A is greater than B.

>=

A > sign followed by an equals sign asks if the first value is greater than or equal to the second value. That is, is the value or expression to the left of the >= signs greater than or equal to the value or expression to the right side? For example, the statement (A >= B) is true if A is greater than or equal to B.

**9.2 THE IF STATEMENT**

You use the If statement structure in your scripts to evaluate a condition. The process your script performs after the evaluation of the condition is based on the outcome of that evaluation. The basic form of the If statement structure includes the key words If, Then and EndIf. You must include each of these key words in all If statements you use within your scripts. If you fail to include any of these key words, then the script compiler generates a syntax error. The basic structure of an If statement follows:

If condition to be evaluated Then

statement or statements to be performed when the If statement is true

EndIf

You always place the condition to be evaluated by JAWS between the If and Then key words. JAWS evaluates this condition to determine what statements should be performed next. When the condition is found to be true, then JAWS performs all statements following the If statement. If the condition is determined to be false, then JAWS skips the statements contained in the If statement and begins performing any statements following the EndIf key word. In the basic form of the If-Then-EndIf statement structure, JAWS only has one branch to choose from. When you construct If statements in your scripts, it is a good idea to indent any statements JAWS performs when the preceding If statement is true. Using indentation makes it easier to determine which statements are contained within the If statement and which are not. You can press TAB in the Script Manager to accomplish this task. This indents the statements inside the If-Then-EndIf structure by approximately 4 spaces. The following If statement compares the values of two variables. The code of the example follows:

Example 1: Using an If-Then-EndIF Statement

var

int iVerbosity

let iVerbosity = GetVerbosity ()

If iVerbosity == BEGINNER Then

SayFormattedMessage (OT\_STATUS, "beginner verbosity is in use")

EndIf

In the above example, JAWS begins by retrieving the current verbosity setting storing it in the variable, iVerbosity. JAWS then performs the If statement comparing the value stored in iVerbosity to the verbosity setting value represented by the constant, BEGINNER. JAWS performs the SayFormattedMessage function following the If statement and speaks "Beginner verbosity is in use" when the values are equal. However, when the values are not equal, then JAWS does not perform the SayFormattedMessage function and no message is heard.

**9.3 ADDING A SECOND BRANCH, THE ELSE KEYWORD**

You can add the Else key word to an If-Then-EndIf statement structure to provide an additional branch. JAWS performs any statements following the Else key word when the condition in the preceding If statement is found to be false. You add the Else key word to the If-Then-Endif structure following any statements to be performed when the preceding If statement is found to be true. Thus, you can perform a statement or group of statements when the condition you are testing is either true or false. An example of an If statement structure that includes the Else key word follows:

Example 1: Adding the Else Key Word to the If-Then-EndIF Statement

var

int iVerbosity

let iVerbosity = GetVerbosity ()

If iVerbosity == BEGINNER Then

SayFormattedMessage (OT\_STATUS, "Beginner verbosity is in use.")

Else

SayFormattedMessage (OT\_STATUS, "Intermediate or advanced verbosity is in use.")

EndIf

In the above example, JAWS compares the value stored in the iVerbosity variable to the value contained in the BEGINNER constant. When they are equal, JAWS performs the SayFormattedMessage function following the If statement and speaks "Beginner verbosity is in use." When the values are not equal, then JAWS performs the SayFormattedMessage function that follows the Else key word and speaks "Intermediate or advanced verbosity is in use." By adding the Else key word to the If statement above, you can account for both true and false conditions in the initial If statement.

**9.4 ADDING MULTIPLE BRANCHES, THE ELIF STATEMENT**

You can add the ElIf statement to an If-Then-EndIf structure to provide a secondary way of formulating a branch. You use the ElIf statement to check a second condition when the condition in the preceding If statement is false. You can use the ElIf statement as a replacement for the Else key word. You can also add one or more ElIf statements to your If statements. However, they must be added before the Else key word. Each ElIf statement must include the Then key word. An example of an ElIf statement follows:

ElIf iVerbosity == INTERMEDIATE Then

When the condition being evaluated in the ElIf statement is found to be true, then JAWS performs all statements following the ElIf statement. If JAWS determines the condition in the ElIf statement is false, then JAWS will do one of the following:

evaluate the next ElIf statement

perform any statements following the Else key word

continue processing the statements in your script that follow the EndIf key word. The example shown in the previous section illustrated the use of the Else key word to allow for both true and false branches when JAWS compared two values. The following example illustrates the use of the ElIf statement as a replacement for the Else key word

Example 1: Using ElIf Instead of Else

If iVerbosity == BEGINNER Then

SayFormattedMessage (OT\_STATUS, "Beginner verbosity is in use")

ElIf iVerbosity != BEGINNER Then

SayFormattedMessage (OT\_STATUS, "Intermediate or advanced verbosity is in use")

EndIf

In the above example, JAWS compares the value stored in iVerbosity with the BEGINNER constant. When they are equal to each other, JAWS performs the SayFormattedMessage function following the If statement and speaks "Beginner verbosity is in use." If JAWS determines the two values are not equal, then JAWS performs the SayFormattedMessage function following the ElIf statement and speaks "Intermediate or advanced verbosity is in use." Although the ElIf statement in the above If statement structure takes the place of the Else key word, it does not allow for separate messages to be spoken when intermediate or advanced verbosity is in use. You may find situations where you need to perform a group of statements when a variable could possibly contain three or more different values, such as the verbosity setting in JAWS. Likewise, yu may want to perform a group of statements when two values are equal to each other, a second group of statements when one value is greater than the other and a third group of statements when one value is less than the other. Thus you can determine if the values are equal, one is greater than the other or one value is less than another. The following example illustrates the use of ElIf statements within an If-Then-EndIf statement structure:

Example 2: Using the ElIf Key Word to Check for Different Conditions

let A = 5

let B = B + 1

If A == B Then

SayFormattedMessage (OT\_MESSAGE, "The values of A and B are equal")

ElIf A > B Then

SayFormattedMessage (OT\_MESSAGE, "The value of A is greater than the value of B")

ElIf A < B Then

SayFormattedMessage (OT\_MESSAGE, "The value of A is less then the value of B")

EndIf

In the above example, JAWS compares the values of A and B. If JAWS determines the values are equal, then the If statement is true. JAWS performs the SayFormattedMessage function following the If statement and speaks "The values of A and B are equal." If JAWS determines the values of A and B are not equal, then the If statement is false. Since the values of A and B are not equal, then JAWS must perform the ElIf statement to determine if the value of A is greater than the value of B. If A is greater than B, the ElIf statement is true. JAWS performs the SayFormattedMessage function following the first ElIf statement and speaks "The value of A is greater than the value of B." If the value of A is not greater than the value of B, then the ElIf statement is false. JAWS must then perform the next ElIf statement that determines if A is less than B. If A is less than B, then the ElIf statement is true. JAWS performs the SayFormattedMessage function following the ElIf statement and speaks "The value of A is less than the value of B."

**9.5 NESTED IF STATEMENTS**

You can place or nest an If statement inside another If statement. When you nest If statements, you can check for a condition only when another condition is found to be true. When the first condition is found to be true, JAWS evaluates the nested If statement. JAWS evaluates it just as it would do for any other If statement. Thus, if the condition in the nested If statement is found to be true, JAWS performs any statements following the If statement. You can use the Else key word within the nested If statement structure. You can also add any number of ElIf statements to the nested If statement structure. When you nest If statements, you should use indentation. Using indentation makes it easier to match the If, Else, Elif, and EndIf keywords up with each other. Thus, you can be assured you have an EndIf that corresponds to each If key word.

Note: You can add any number of nested If statements within your scripts and functions. However, the more nested If statements you add, the harder your scripts are to debug when errors are encountered at compilation time. You should try to limit nesting If statements to one or two levels.

Example 1: Nested If statements

Var

Int iValue

Let iValue = CalculateNumber (); Assign a number to iValue

If iValue > 4 Then

If iValue == 5 Then

SayFormattedMessage (OT\_MESSAGE, "The value is 5")

ElIf iValue == 7 Then

SayFormattedMessage (OT\_MESSAGE, "The value is 7")

Else

SayFormattedMessage (OT\_MESSAGE, "The value is greater than 4 but is not equal to 5 or 7")

EndIf; complete the nested If statement

Else

SayFormattedMessage (OT\_MESSAGE, "The value is less than or equal to 4")

EndIf; complete the outer If statement

In the above example, JAWS performs the CalculateNumber function and stores the result in the iValue local variable.

The outer If statement first determines if the value stored in iValue is greater than 4.

If the value is greater than 4, then the inner If statement executes.

Thus, if the value is 5, then JAWS performs the SayFormattedMessage function following the If statement and speaks, “The value is 5."

If the value is equal to 7, then JAWS performs the SayFormattedMessage function following the Elif Statement and speaks, “The value is 7."

If the value stored in iValue is greater than 4 but is not 5 or 7, JAWS performs the SayFormattedMessage function following the Else Statement and speaks, "The value is greater than 4 but is not equal to 5 or 7."

If the value is less than or equal to 4, JAWS does not process the nested If statement. Instead, JAWS jumps directly to the Else statement following the nested If statements and performs the final SayFormattedMessage function that speaks, "The value is less than or equal to 4." Regardless of the number stored in iValue, JAWS will speak one of the above messages.

**9.6 COMPOUND IF STATEMENTS**

Compound statements contain two or more conditions that JAWS evaluates before your script continues processing. There are two types of compound If statements:

Statements joined by "AND"

Statements joined by "OR" The type of compound statement you use within your script is based largely on the type of processing that needs to occur under certain situations. Compound statements can often replace nested If statements.

Compound Statements Using "And"

A compound statement joined by “and” evaluates two or more conditions before processing continues. All conditions must be true before the entire If statement is said to be true. If JAWS determines that one of the conditions is false, then the entire If statement is considered to be false. You use two ampersands (&&) placed next to each other to indicate a compound statement joined by “and.”

Example 1: A compound statement joined by “and”

Var

Int iValue

Let iValue = CalculateNumber (); Assign a number to iValue

If iValue > 5 && iValue < 10 Then

SayFormattedMessage (OT\_MESSAGE, “The value is between 5 and 10.”)

Else

SayFormattedMessage (OT\_MESSAGE, “The value is not between 5 and 10.”)

EndIf

In the above example, the compound statement determines if the value stored in iValue is between 5 and 10.

If the value stored in iValue is greater than 5 but less than 10, then the If statement is true and JAWS speaks, “The value is between 5 and 10.”

If the value stored in iValue is less than or equal to 5 or greater than or equal to 10, then the If statement is false and JAWS speaks, “The value is not between 5 and 10."

**Compound Statements Using "Or"**

A compound statement joined by “or” evaluates two or more conditions before processing continues. Any of the conditions can be true to make the entire If statement true. If JAWS determines that none of the conditions are true, then the entire If statement is false. You use two vertical bars (||) placed next to each other to indicate a compound statement joined by “or.”

Example 2: A compound statement joined by “or"

If iValue == 5 || iValue == 10 Then

SayFormattedMesssage (OT\_MESSAGE, “The value is either equal to 5 or 10.”)

Else

SayFormattedMessage (OT\_MESSAGE, “The value is not equal to either 5 or 10.”)

EndIf

In the above statement, the compound statement determines if the value stored in iValue is equal to 5 or 10.

If the value is either 5 or 10, the If statement is true and JAWS speaks, “The value is either 5 or 10.”

If the value stored in iValue is not equal to 5 or 10, then the If statement is false and JAWS speaks, “The value is not equal to 5 or 10.”

**Short Circuit Evaluation**

JAWS uses a feature called short circuit evaluation to speed up the execution of scripts. This feature keeps JAWS from having to evaluate all parts of the expression whether it is needed or not. Short circuit evaluation works by evaluating an expression for an "or condition" such that if the first variable condition is true there is no need to evaluate the second variable condition. In an "and condition," if the first variable condition is false then there is no need to evaluate the second condition. Consider the following if condition:

If (function1() || function2()) then

In versions of JAWS prior to 5.0, both function1 and function2 would be executed even if function1 returned TRUE. Now however, if function1 returns true then function2 is not executed. In this example, function2 is only executed if function1 returns false. Similarly:

If (f1() && f2()) then

If f1 returns false, f2 will not be evaluated because it will not be possible for both sides of the "and" expression to be true. The function f2 will only be executed if f1 returns true.

**9.7 LOOPING**

You can use the iterative statement structure to perform the same statement or group of statements repeatedly while a condition is true. This type of flow control is also referred to as "looping". You can use a loop in your script to perform a sequence of statements several times. A loop essentially shortens the number of statements required. Looping provides a way to determine the number of repetitions automatically, based on evaluations made in your script.

The While-EndWhile Statement Structure. You use the While-EndWhile statement structure to perform Looping. A While Loop repeats or loops WHILE a condition is true. A While loop consists of three parts:

The While key word that begins the loop

the condition to be tested each time the loop iterates or is performed

the EndWhile key word that ends the loop.

JAWS performs all statements within the boundaries of the While and EndWhile key words repeatedly until the condition in the While statement becomes false.

Example 1: While-EndWhile loop structure

Var

Int iCount

let iCount = 1 ;Assign the value of 1 to the variable

While iCount < 5 ; this loop will be performed 4 times

; Statements to be performed go here

Let iCount = iCount + 1 ; Increases the value of iCount so the While condition will become false after 4 iterations

EndWhile

In the above example, JAWS performs the While loop while iCount is less than five. Once the value of iCount becomes equal to 5, the loop ends and no further processing occurs.

The For-EndFor Statement Structure. Another method of performing looping in your scripts is by using the For-EndFor looping statements. Similar to a While loop, a For loop consists of three parts: the keyword For that starts the loop, the condition being tested, and the EndFor keyword that terminates the loop. JAWS performs all statements found in the boundaries of the loop as long as the loop condition is true. An example of the syntax of a For loop follows.

For x = Start to End

...Statements to be performed go here

EndFor

The value of x must be an integer variable and the values of Start and End must be integers, integer variables, or integer constants. The major difference between the While and For looping structures is that a For loop will run a predetermined number of times. An example of the use of a For loop follows.

Var

int iCounter

For iCounter = 1 to 10; Loop will execute 10 times

SayInteger(iCounter); Speak the value of the iCounter variable

EndFor

In the above example the For loop will execute ten times. As set by the For statement, the iCounter variable starts with a value of 1 and increments by 1 each time the loop executes. Notice that the iCounter variable is automatically incremented each time the loop executes. At the end of the tenth loop iteration the value of icounter becomes 11 and the For statement is no longer true thus ending the loop. Each time the loop executes the SayInteger function speaks the value of iCounter. When this script is run you will hear JAWS count from one to ten.

NOTE: Since a loop continues until a condition becomes false, you must be careful not to set up a loop with a condition that will never become false. To do so would create an infinite loop, which will cycle forever, and lock up the computer until the program is terminated manually. Think carefully about the condition meant to terminate the loop to be sure it will become false at some time. If you find that the computer seems to lock up after you execute a new script with a While loop in it, this is probably what is happening. For loops are less likely to cause an infinite loop because beginning and ending alues are explicitly set in the For statement.

**Avoiding Infinite Loops**

When you use a While loop in your script, care should be taken not to create an infinite loop. An infinite loop occurs when the condition in the While statement is not satisfied. The loop will continue to run until you manually end the program. This often causes a computer to lock up and may require the computer be restarted. Since JAWS performs a loop until a condition is met, you must be certain that the condition being tested can be satisfied. One way to avoid this problem while designing While loops is to include statements designed to end the loop after a certain number of repetitions.

Example 2: Ending a loop after a certain number of iterations

Var

int iSafety

Let iSafety = 1

While iSafety < 10

; Loop statements go here

Let iSafety = iSafety + 1 ; Increase the value of iSafety by one so it will eventually reach ten and end the loop

EndWhile

In the above example, the iSafety variable is initialized to the value of 1.

The loop checks the value of the iSafety variable each time it repeats.

Inside the While loop, the iSafety variable is incremented by one to count the number of repetitions of the loop.

When the value of iSafety reaches 10, the While statement becomes false and the loop no longer executes.

A second example of the use of the While statement follows. This example could be used to move down through a menu until a specific item is found.

Example 3: Looping through a menu

Var

int iExitLoop,

string sSearch

let iExitLoop = 0; set the value of the variable to zero

let sSearch = “Open”; assign the string value of "Open" to the sSearch variable

TypeKey (“ALT+F”); Activates the file menu

While iExitLoop == 0

NextLine (); Moves the position to the next line

If StringContains (GetLine (), sSearch) == TRUE Then; We found the word "Open" in the menu item

EnterKey (); Passes the ENTER key through to the application just like pressing it from the keyboard

let iExitLoop = 1; Set the value of the iExitLoop variable to 1. This will end the loop

EndIf

EndWhile

In the above example, two variables are set before the While loop begins.

The value of the iExitLoop variable is set to zero.

The value of the sSearch variable is then set to the value of “Open”. The sSearch variable contains the string of text you want to locate in the menu item as you move down through the menu.

The While loop repeats until the value of iExitLoop changes from zero to one.

The first statement in the while loop, NextLine, moves the active cursor down one line.

After the cursor is moved, the String Contains function checks to see if the highlighted menu item contains the string, “Open.”

If “Open” is found in the menu item, then the If statement is found to be true and the value of iExitLoop is changed from zero to one.

When the value of iExitLoop becomes one, the While statement will no longer be true and the loop will stop executing.

If running the above script, it is possible JAWS could end up in an infinite loop. If the file menu does not have an option with the word, "Open" in it, the value of the iExitLoop variable will never be set to one. This means the While statement will always be true and the loop will never end. In order to prevent this from happening, we need to let the iExitLoop variable act as a counter that will reach a certain value and cause the loop to end. Here is the example written again using a counter to insure the loop will end, even if the word, "Open" is not found in the menu.

Example 4: Using a counter to end a loop

Var

int iExitLoop,

string sSearch

let iExitLoop = 0; set the value of the variable to zero

let sSearch = "Open"; assign the string value of "Open" to the sSearch variable

TypeKey ("ALT+F"); Activates the file menu

While iExitLoop < 15; This insures the loop will be forced to end after 15 iterations if "Open" is not found

NextLine (); Moves the position to the next line

If StringContains (GetLine (), sSearch) == TRUE Then; We found the word "Open" in the menu item

EnterKey (); Passes the ENTER key through to the application just like pressing it from the keyboard

let iExitLoop = 15; Set the value of the iExitLoop variable to 15. This will end the loop

EndIf

let iExitLoop = iExitLoop + 1; Increases the value of iExitLoop by 1. When the value becomes 15, the loop will end (even if "Open" was not found in the menu.)

EndWhile

This example differs from the previous example in the following ways:

The While statement runs as long as the value of iExitLoop is less than 15.

If the word, "Open" is found in a menu item, the value of iExitLoop is set to 15. This insures the loop will end. During each iteration of the loop, the value of iExitLoop is incremented by one. This means that even if the search string is not found, the loop will end when iExitLoop is incremented to 15.

**9.8 CHAPTER EXERCISES**

The following exercises will give you practice in creating If-Then statements. Note: The exercises in this chapter will not walk you through the steps needed to open the Script Manager or the steps needed to activate the New script dialog. To review these steps, refer to the previous chapters in this manual.

**Exercise 9.1: Using the If statement to Check for User Verbosity**

The following script for Notepad uses an If statement to determine which user verbosity level is active and speaks an appropriate message. You will need to use the GetVerbosity function to determine the current verbosity level. The GetVerbosity function returns the constant BEGINNER, INTERMEDIATE, or ADVANCED depending on which level JAWS is currently set on. The constants are defined in the header file hjconst.jsh. Use the If statement and the GetVerbosity function to determine which verbosity level is currently active and speak a message accordingly.

Make sure that the Notepad script file is open and your cursor is at the bottom of the file ready to insert a new script.

Add the following information in the New Script dialog:

Script Name: CheckVerbosity

Can be Attached to Key: checked

Synopsis: Speaks a message based on which verbosity level is active.

Description: This script uses the GetVerbosity function to determine which verbosity level is active and then speaks a message accordingly.

Category: None

Assign to Key: CTRL+SHIFT+C

The script should first check to see if the verbosity level is set to beginner. If so, a message should be spoken. After typing the keyword If, press CTRL+I to activate the Insert Function dialog box and insert the GetVerbosity function. Notice the function requires no parameters. The statement should look like the following and should be placed beneath the line containing the script name:

If GetVerbosity () == BEGINNER Then ; Check to see if verbosity is beginner.

NOTE: You do not need quotes around the word BEGINNER because it is a constant value rather than a string value. On the next line use the SayFormattedMessage function to have JAWS speak the message, "Your verbosity is beginner." Be sure to indent the line first. The line should look like the following:

SayFormattedMessage (OT\_Message, "Your verbosity is beginner")

Since we are checking for three conditions (beginner, intermediate, or advanced) we must use the ElIf statement to check for intermediate verbosity. The statement should look like the following:

ElIf GetVerbosity () == INTERMEDIATE Then ; Check to see if verbosity is INTERMEDIATE

On the next line use the SayFormattedMessage function to have JAWS speak the message, "Your verbosity is intermediate."

Since the verbosity level must be one of three levels, we know that if it is not beginner or intermediate then it must be advanced. Therefore we can simply use the Else statement and have JAWS speak the message, "Your verbosity is advanced."

After inserting the Else statement and final SayFormattedMessage function don't forget the EndIf keyword before the EndScript statement. When finished, your script should look like the following:

Script CheckVerbosity ()

If GetVerbosity () == BEGINNER Then ; Check to see if verbosity is beginner.

SayFormattedMessage (OT\_Message, "Your verbosity is beginner")

ElIf GetVerbosity () == INTERMEDIATE Then ; Check to see if verbosity is INTERMEDIATE

SayFormattedMessage (OT\_Message, "Your verbosity is intermediate")

Else ; If the verbosity is not beginner or intermediate, it must be advanced

SayFormattedMessage (OT\_Message, "Your verbosity is advanced")

EndIf

EndScript

Press CTRL+S to save and compile the script. If any errors exist be sure your script looks like the one above.

Switch over to Notepad and press CTRL+SHIFT+C to run your script. JAWS should speak the message telling you what your verbosity currently is.

Be sure to change your verbosity to all three levels and test your script. Do the following to change your verbosity:

Press INSERT+V to open the Adjust JAWS Options dialog box.

Press the letter 'U' until you find the User Verbosity option and press SPACEBAR to cycle between the three values.

Press ENTER to accept your change and close the Adjust JAWS Options dialog box and return to Notepad.

**Exercise 9.2: Looping Example**

The following script in Notepad should make JAWS count out loud from one to ten. You will accomplish this using a While EndWhile loop. During each iteration through the loop, JAWS will speak the value of an integer variable and then increase the value of the variable by one. The SayInteger function will be used to speak the value of the integer variable.

Make sure that the Notepad script file is open and your cursor is at the bottom of the file ready to insert a new script.

Add the following information in the New Script dialog:

Script Name: LoopingExample

Can be Attached to Key: checked

Synopsis: Counts out loud to ten.

Description: This script uses a loop and the SayInteger function to count from one to ten.

Category: None

Assign to Key: CTRL+SHIFT+L

You must first create an integer variable that will be used to count to ten. Make sure the cursor is in the body of the script and create an integer variable named iCount.

Using a Let statement, assign a value of 1 to the iCount variable.

Since we want JAWS to stop counting after it reaches ten, we use a While loop that only executes while the value of iCount is less than eleven. Enter the following on a blank line below the Let statement:

While iCount < 11; This loop will execute 10 times

You can now insert the SayInteger function that will speak the value of the iCount variable. On a blank line below the While statement, press TAB to indent the code inside the loop. Then press CTRL+I to activate the Insert Function dialog box and locate the SayInteger function.

After choosing the SayInteger function, The Script Manager prompts you for two parameters. The first parameter is the numeric value or integer variable containing the numeric value to be spoken. When prompted by the Script Manager for this parameter, type "iCount" without the quotation marks to indicate that the function will speak the value of the iCount variable. The Script Manager then prompts you for the second parameter, the base of the number to be used. Since this parameter is optional, press TAB until you reach the Finish button followed by the SPACEBAR to activate the button and close the Insert Function dialog. After the function is inserted into your script, you will need to remove the comma and space between the iCount variable and the right parenthesis. The line should look like the following:

SayInteger (iCount)

On the next line you can now increase the value of the iCount variable by one. First press TAB to indent the new line to the same level as the previous line. Now type the following Let statement:

Let iCount = iCount + 1; Adds 1 to the value of iCount. When it reaches 11, the loop will stop.

On the next line add the EndWhile keyword to conclude the loop.

After performing the above steps, your script should look like the following:

Script LoopingExample ()

var

int iCount

Let iCount = 1

While iCount < 11; This loop will execute 10 times.

SayInteger (iCount); Speaks the value of the iCount variable

Let iCount = iCount + 1; Adds 1 to the value of iCount. When it reaches 11, the loop will stop.

EndWhile

EndScript

Press CTRL+S to save and compile the script. If you get a compile error, check to make sure your script matches the example above.

You can now move to Notepad and press CTRL+SHIFT+L to activate your script. You should hear JAWS count from one to ten.

**10.0 PASSING KEYSTROKES AND TYPING TEXT WITH SCRIPTS**

People new to writing scripts with JAWS often find it hard to differentiate between JAWS and Windows keystrokes. For example, the script writers at Freedom Scientific have attached scripts to common Windows keystrokes such as CTRL+C and CTRL+V, copy and paste. When you press either keystroke, Windows still copies and pastes using these keystrokes. However, something else also happens; when you press CTRL+C JAWS speaks "copied selection to clipboard." Likewise, when you press CTRL+V, JAWS says, "Pasted from clipboard." These messages let you know the expected action has taken place. The scripts associated with these keystrokes are examples of "pass-through" scripts. It is called a passhrough script because JAWS intercepts the keystroke before it reaches the application and performs the associated script. The script adds some type of extra functionality to the keystroke and then passes the keystroke through to the application. In this case the added functionality is the message JAWS speaks to notify you the action has been completed. In this chapter, you will learn how to capture keystrokes, add extra functionality to those keystrokes and pass them through to the active application. You will learn how to access menu bars and give keyboard access to commands in the menu system. You will also learn how to type text using scripts.

**Table of Contents**

Chapter 10.0 Passing Keystrokes and Typing Text with Scripts, contains the following sections:

10.1 Pass-Through Scripts

10.2 Giving Keyboard Access

10.3 Delaying Your Script

10.4 Typing Text

10.5 Chapter Exercises

**10.1 PASS-THROUGH SCRIPTS**

When you create a pass-through script, you must make certain to pass the keystroke through to the application. It is important to remember that JAWS determines what script to process each time you press a keystroke. If the application keystroke is now attached to a script, then JAWS performs the script first. If you do not pass the keystroke through within the script, then the keystroke never activates the intended action and functionality of the keystroke is lost.

Creating the Script.

You can use the built-in function, TypeKey, to pass the keystroke through to the active application once you have enhanced its functionality in your script. The TypeKey function requires a single parameter to complete its task. This parameter is a string of text and can be enclosed in quotation marks, stored in a string variable or a string constant. The keystroke can be anything from a single character such as the letter S to a keystroke that opens a menu such as ALT+F. You call the TypeKey function using the following syntax:

TypeKey ("Alt+F")

The above example shows the parameter being passed to the function as a quoted string of text. An alternative way of passing the parameter is to store the keystroke in a local variable you have declared in the script. The following code example declares a local variable, stores the keystroke and then uses the variable as a parameter for the TypeKey function:

Var

String sKeystroke

Let sKeystroke = "ALT+F"

TypeKey (sKeystroke)

You can also declare all of your keystrokes in a JAWS script header file. To declare a keystroke constant for ALT+F, type the following two statements in a JAWS script header file:

Const

KSFileMenu= "Alt+F"

In the above constant declaration, the “KS” indicates it is a constant containing a keystroke. You can review a number of more commonly used keystroke constants by viewing the Common.jsm JAWS script message file within the Script Manager. You will find constant definitions for most of the more common keystrokes such as SPACEBAR, CTRL+PAGE UP and CTRL+PAGE DOWN in the file. The following code example illustrates the use of a constant as the parameter for the TypeKey function:

TypeKey (KSFileMenu)

**10.2 GIVING KEYBOARD ACCESS**

When you encounter an application that does not use keyboard shortcuts, there are several keystrokes needed to perform a specific menu item. With a script you can accomplish the same thing with one keystroke. You can also add a helpful message to the script that JAWS speaks after the keystrokes have been processed. Before you write this type of script, you must determine the keystrokes required to move to the menu, select to the desired item and perform that item. You should be certain that the keystrokes you pass in your script match those steps exactly. For example, if you want to select a menu item that begins with S and has no access key, you must make certain there are no other menu options starting with S. If there are multiple items in the File beginning with S, you must use conditional statements to make sure you are selecting the correct menu item.

Moving to the Menu Bar. The first action your script should perform is to move to the menu bar. You can move to the menu bar in one of two ways:

Use the ActivateMenuBar function

Use the TypeKey function

The ActivateMenuBar function moves to or closes the Menu bar for the active application. When you use this function, it acts as if you pressed ALT to move to the menu bar or to close open menus. This function is useful in situations where you only want to move to the menu bar or close menus that are already open. You can use the TypeKey function to move to the menu bar as well. If you only want to move to or close the menu bar, you should use the ActivateMenuBar function. But if you want to move to and open a specific menu, then the TypeKey function works best. The following examples illustrate the use of both functions to move to the menu bar:

ActivateMenuBar ()

TypeKey ("Alt")

**Checking the Menu Bar**

After you move to the menu bar, you should make sure the move was successful. You don’t want to call the TypeKey function to pass a keystroke to activate a menu item when your script did not move to the menu bar successfully. To do so may produce unwanted results. You can use the MenusActive function to determine the state of the menu bar. The function returns a constant value of ACTIVE to indicate the move to the menu bar was successful. When the menu bar is closed, the function returns the constant value of inactive. You can use this function in a conditional statement to determine if the move to the menu bar was successful. The following code example moves to the menu bar and verifies the move was successful:

Example 1: Moving to the Menu Bar

ActivateMenuBar (); move to the menu bar

If MenusActive () Then

SayFormattedMessage (OT\_STATUS, "The move to the menu bar was successful")

Else

SayFormattedMessage (OT\_ERROR, "The move to the menu bar was not successful")

EndIf

The following code example moves to the menu bar, opens the File menu, makes sure the move to the menu was successful then passes S through to the application to activate the Save option. You could use this example in any application that does not make use of the CTRL+S keystroke shortcut.

Example 2: Opening and Selecting an Option in the File Menu

; constant definitions for keystrokes

; these should be added to a JAWS script header file

Const

KSFileMenu="F",

KSSave="S"

; script to save the current file

Script SaveFile ()

ActivateMenuBar ()

If MenusActive () Then; determine if menu is open

TypeKey (KSFileMenu); open the File menu

TypeKey (KSSave); select and initiate the Save action

Else

SayFormattedMessage (OT\_ERROR, "The File menu could not be opened.")

EndIf

EndScript

In the above example, the ActivateMenuBar function moves to the menu. The MenusActive function determines the state of the menu bar. If the move to the menu was successful, then JAWS performs the two TypeKey functions to open the File menu and perform the Save option. If the move to the menu bar was not successful, then JAWS speaks an appropriate error message.

**10.3 DELAYING YOUR SCRIPT**

Your scripts generally process faster than the system you are using. When you pass keystrokes from within your script, you may need to pause the script to give the system time to process the keystrokes. You can use one of two built-in functions to pause the processing of your script:

Pause

Delay

The two functions differ in how much time they allow for the system to catch up with your script. The Pause function pauses the processing of your script so that other applications can complete tasks. The delay function allows you to specify the amount of time your script pauses before processing resumes.

You don’t have to limit the use of these functions to only scripts that pass keystrokes. You can use both functions in other circumstances where the script needs to be paused to allow for system processing. Knowing when to use one of these functions depends on the application and the PC. You may not need to pause the processing of a script to allow the system to catch up on a fast PC. On the other hand, a slower PC may require more time to process a command.

**The Pause Function**

The pause function causes JAWS to yield to the time needs of other applications. Once other applications have been given the opportunity to use processing time, then JAWS resumes processing the script. An example of the use of the Pause function follows:

Example 1: Using the Pause Function

TypeKey ("alt+f") ; open the File menu

Pause () ; allow system to open the File menu

If MenusActive () Then; make sure the menu bar opened

TypeKey ("s") ; activate the Save option

Else

SayFormattedMessage (OT\_ERROR, "The menu bar did not open")

EndIf

In the above example, the TypeKey function opens the File menu. Next, the Pause function stops the processing of the script to give the application time to open the menu. Before JAWS performs the TypeKey function, the MenusActive function must determine if the menu bar is open. If the menu bar is open, then JAWS performs the TypeKey function and the menu item is activated. If the menu did not open successfully, then JAWS speaks an appropriate error message.

The Delay Function. The Delay function gives you more control than the Pause function when stopping your script. Since the Pause function relies on other system processes, the amount of time your script is stopped can vary. If this amount of time is insufficient for your script to process correctly, then you should use the Delay function.

The Delay function requires a numeric value as a parameter. This value is a whole number that tells JAWS the length of time to pause script execution. This value can be a numeric value such as 5, an integer variable containing a value or a constant representing a numeric value. The amount of time JAWS pauses script execution is measured in tenths of a second. For example, a value of 1 tells the delay function to stop for a tenth of a second. In JAWS 5.0 and above, the Delay function accepts an optional second parameter. This parameter is a numeric value that tells JAWS to suppress checking for and processing FocusChangedEvent and NewTextEvent functions before returning from the delay. When you pass TRUE in this parameter, then JAWS suppresses any processing of the NewText or FocusChanged events. If you pass FALSE in this parameter, then JAWS will not suppress the processing of these two events. You call the Delay function using the following syntax:

Delay (nDelayValue, , nNoEvents)

An example of the use of the Delay function follows:

Example 2: Using the Delay Function

TypeKey ("alt+f") ; open the File menu

Delay (3); allow system to open the File menu

If MenusActive () Then

TypeKey ("s") ; activate the Save option

Else

SayFormattedMessage (OT\_ERROR, "The menu bar did not open")

EndIf

In the above example, the first TypeKey function activates the File menu. JAWS then performs the Delay function to pause script processing to give the File menu time to open. The MenusActive function then determines if the menu is open. If it is open, then JAWS performs the second TypeKey function and the Save option is activated. Otherwise, JAWS speaks an appropriate error message.

**10.4 TYPING TEXT**

You may encounter situations where you have to type the same block of text repeatedly. For example, you may use a number of different signatures in your email client. You might use one signature for business correspondence, while using a second signature for personal messages. Adding these signatures to their respective email messages may become tedious. You can create a script that uses built-in functions to type the information into the message for you with only one keystroke.

**Typing the Text**

You can use the built-in function, TypeString, to type a string of text into a document. You supply the function with the string of text to be typed by the function as its only parameter. This string of text can be surrounded by quotation marks, stored in a variable or declared as a string constant. After JAWS places the text into the document, JAWS places the insertion point at the end of the string of text. The following code example illustrates the use of the TypeString function:

TypeString ("This text is being typed into my document")

The TypeString function types the following text into your document:

This text is being typed into my document

Since the TypeString function does not add a return character to end the line just typed, the insertion point is placed to the right of the last character typed by the function. In the above example, the insertion point is to the right of the last “t” in the word “text.”

**Adding Carriage Returns**

When you call the TypeString function multiple times, the text typed is not separated by spaces. When you want to add a carriage return between two blocks of text, you must perform this action from within your script. The EnterKey built-in function passes the ENTER key through to the active application. If you have positioned the Virtual PC cursor on a link or button, then the EnterKey function activates it. If the Virtual PC cursor is on another form control, then the EnterKey function activates forms mode. You call the EnterKey function using the following syntax:

EnterKey ()

The following example illustrates the use of the TypeString and EnterKey functions:

TypeString ("This is the first line of text.")

EnterKey ()

TypeString ("This is the second line of text.")

The above example produces the following text in the active application:

This is the first line of text.

This is the second line of text.

Since the EnterKey function is called between the two TypeString functions, the text did not run together and is separated by a carriage return. The same example without the EnterKey function follows:

TypeString ("This is the first line of text.")

TypeString ("This is the second line of text.")

This example produces the following text in the active application:

This is the first line of text.This is the second line of text.

Since the EnterKey function was not performed after the first TypeString function, the text runs together as there is nothing separating the two strings of text.

**10.5 CHAPTER EXERCISES**

The following exercises give you practice creating scripts that pass keystrokes, access menu options that do not have shortcut keys and type text into the active document window. These chapter exercises do not walk you through the steps needed to create the script. Instead, the goal of the script is shown first, the script documentation second and the code of the script third. If you need to review the steps used to create a script or add functions to the script, refer to chapters 1 through 9 for information on the steps needed to start the Script Manager and the use of the New Script dialog.

**Exercise 10.1: Creating a Pass-Through Script**

The objective of this exercise is to create a pass-through script that takes an existing application keystroke and enhances its functionality. Within Internet Explorer, you can press, ALT+HOME, to move immediately to your home page. When you press this keystroke, Internet Explorer does not indicate audibly that you are going back to your home page.

Create this script within the script file for Internet Explorer, Internet Explorer.jss. . Your script should begin by passing the ALT+HOME keystroke through to Internet Explorer. Before the script completes, JAWS should speak an informational message advising you that you are now going back to your home page. After you have compiled the script, you can test the script by going to a Web page other than your home page and pressing ALT+HOME. After you hear JAWS speak the informational message, your home page should begin loading into Internet Explorer. If you find that your home page is loaded into Internet Explorer, then you can confirm the action was successful.

Script Documentation:

Script Name: GoToHomePage

Attach to Key: Checked

Synopsis: Loads your home page.

Description: Passes ALT+HOME to Internet Explorer to load your home page and announces it.

Category: Keyboard

Assign To: ALT+HOME

GoToHomePage Script:

Script GoToHomePage ()

TypeKey ("ALT+HOME")

SayFormattedMessage (OT\_STATUS, "Go to home page.", cMsgSilent)

EndScript

NOTE: You can add the messages JAWS speaks to a JAWS header or message file and then include that file within the script file for Internet Explorer. See 8.0 Creating and Speaking Messages for more information on creating JAWS message files and adding messages to the file.

**Exercise 10.2: Typing Text with a Script**

The objective of this exercise is to create a script that types your e-mail signature in any active application window. You can use this script in your e-mail application, any text editor, or a word processor. You can use a series of TypeString functions to type the text of your signature. Between each call to the TypeString function, be sure to insert calls to the EnterKey function. After you have typed all the necessary text for your signature, JAWS should speak a message informing you the signature was added.

Script Documentation:

Script Name: TypeSignature

Can be Attached to Key: Checked

Synopsis: Types an e-mail signature in any application.

Description: Types an e-mail signature in any application using the TypeString function.

Category: Keyboard

Assign To: CTRL+SHIFT+W

TypeSignature Script:

Script TypeSignature ()

TypeString ("Training Department")

EnterKey ()

TypeString ("Freedom Scientific Inc.")

EnterKey ()

TypeString ("11800 31st CT. N")

EnterKey ()

TypeString ("St. Petersburg, FL 33716")

EnterKey ()

TypeString ("PH: (800) 444-4443")

EnterKey ()

TypeString ("FAX: (727) 471-7927")

EnterKey ()

TypeString ("training\_info@FreedomScientific.com")

EnterKey ()

SayFormattedMessage (OT\_STATUS, "Signature added")

EndScript

**11.0 USING THE VIRTUAL VIEWER**

When you press INSERT+H or INSERT+W to display hot key help, JAWS displays the keystrokes within the virtual viewer. Likewise, when you press INSERT+F1 to review screen sensitive help, JAWS displays the information in the virtual viewer as well. When JAWS displays information in the virtual viewer, the virtual cursor is active. You can use normal navigation keystrokes to read, select, and copy selected text to the clipboard.

The virtual viewer is a Virtual Buffer you create using the Scripting Language and is a part of JAWS. The virtual viewer is also known as the user buffer. Text displayed in the virtual viewer is black on a white background, has a font size of 12 and a font type of Arial. The text displayed in the virtual viewer is identical to text found in any word processor or text editor. In this chapter, you will learn how to display messages in the virtual viewer. You will also learn how to create keystroke links within the virtual viewer.

**Table of Contents**

Chapter 11.0 Using the Virtual Viewer, contains the following sections:

11.1 Checking the Virtual Viewer Status

11.2 Creating the Messages

11.3 Displaying the Messages

11.4 Displaying Keystrokes as Links

11.5 Chapter Exercises

**11.1 CHECKING THE VIRTUAL VIEWER STATUS**

Before you display text in the virtual viewer, you should check to make sure it is not currently being displayed by another script or function. When you find the virtual viewer is being displayed, you should deactivate it. Deactivating the virtual viewer before you display new information prevents the text from running together. You use the built-in function, UserBufferIsActive, to check the status of the virtual viewer. The function returns a value of 1 when the virtual viewer is active or displayed and a value of 0 when the viewer is not displayed. You can substitute the constant, TRUE, for the value of 1 and the constant, FALSE, for the value of zero in your scripts. When the virtual viewer is active, you use the built-in function, UserBufferDeactivate, to close the virtual viewer. This function closes the virtual viewer just as if you pressed ESC from the keyboard. You should place both the UserBufferIsActive and the UserBufferDeactivate functions within a conditional statement before you display the virtual viewer. An example of the use of both functions follows:

If UserBufferIsActive () Then

UserBufferDeactivate () ; close the virtual viewer

EndIf

**11.2 CREATING THE MESSAGES**

The messages you display in the virtual viewer are stored in a JAWS message file (.jsm) using the JAWS message format. You can format the text using spaces, carriage returns, tab stops etc. and JAWS retains the format of your message when it is displayed in the virtual viewer. Unlike spoken messages, you cannot create and use long and short messages for use within the virtual viewer. If you want to account for different levels of verbosity, then you will have to speak the messages with your scripts instead of displaying them in the virtual viewer. You can also use place holders in messages you display in the virtual viewer. The use of place holders allows for text to be added to the message at the time JAWS performs the script. For more information on creating messages and using place holders, see 8.0 Creating and Speaking Messages.

**11.3 DISPLAYING THE MESSAGES**

After you have created your messages and determined the virtual viewer is not active, you are ready to display text in the virtual viewer. You can use either of the following functions:

SayMessage

SayFormattedMessage

Each of these functions requires two parameters. The first parameter is the output type that tells JAWS how to speak or display the message. You use the output type constant, OT\_USER\_BUFFER, in either function to tell JAWS to display your message within the virtual viewer. The second parameter is the text of the message JAWS displays in the virtual viewer. This parameter can be a quoted string of text, a string variable containing the text or a message constant contained within a JAWS message file. Examples of the use of both statements follow:

SayMessage (OT\_USER\_BUFFER, "This text is being displayed in the virtual viewer")

SayFormattedMessage (OT\_USER\_BUFFER, "This text is being displayed in the virtual viewer.")

**11.4 DISPLAYING KEYSTROKES AS LINKS**

In addition to text, you can create keystroke links in the virtual viewer. To create a keystroke link in the virtual viewer, you add the KeyFor function within the body of any individual message stored in a JAWS message file. The KeyFor function requires the name of a script as its only parameter. When JAWS displays the message containing the KeyFor function in the virtual viewer, JAWS retrieves the keystroke for the script named passed as a parameter to the function. The keystroke is displayed as a link that you can activate by pressing ENTER on the line containing the keystroke. The keystroke is displayed in underlined, blue text on the same white background as other text contained within the viewer. The syntax for using the KeyFor function in a message follows:

%KeyFor(ScriptName)

The percent sign preceding the function name acts as a placeholder for the keystroke within the message. The script name does not include the parentheses and can be located in either the default or application-specific script file. An example of a message that contains the KeyFor function follows:

@MSGVirtualViewer

This text is being displayed in the virtual viewer.

If you want to see a list of general JAWS hot keys, press %KeyFor (HotKeyHelp).

@@

JAWS displays the message in the virtual viewer as:

This text is being displayed in the virtual viewer.

If you want to see a list of general JAWS hot keys, press JAWSKey+H.

JAWS displays the portion of the message, “This text is being displayed in the virtual viewer,“ in black text on a white background. JAWS displays the portion of the message, “If you want to see a list of general JAWS hot keys, press JAWSKey+H,“ in blue text on a white background. The text is displayed in blue text on a white background to indicate visually the text is part of the keystroke link, INSERT+H. JAWS also indicates verbally that this text is a link.

**11.5 CHAPTER EXERCISES**

The following exercises give you practice creating and using messages in the virtual viewer. Before you start these exercises, make sure you have the Notepad.jss script file open within the Script Manager. This file should contain any practice scripts from earlier chapters of this manual. If you run Script Manager from within Notepad and a blank file is opened, type the following lines at the top of the file:

; Script file for Notepad

; JAWS version 12.0

include "hjconst.jsh"

include "common.jsm"

**Exercise 11.1: Displaying a Message in the Virtual Viewer**

The objective of this exercise is to create a message and display it in the virtual viewer. You should start Notepad and run the Script Manager from within Notepad for this exercise. Before you write your script, you need to create the message used by your script in the Notepad.jsm message file. You can open the file within the Script Manager and add your message. If you already have a Notepad.jsm message file, do the following to open the file in the Script Manager:

Press CTRL+O to display the Open File dialog.

Type Notepad.jsm in the File Name edit box and press ENTER. If you have not previously created the Notepad message file, perform the following:

Press CTRL+N to display the New File dialog.

The active control is a list of file types. Press M to select Messages followed by ENTER. This will open an untitled message file in the Script Manager.

Type the following text in the file:

; Message file for Notepad

Messages

EndMessages

Press CTRL+S to save the file. JAWS prompts you for the filename. Type Notepad in the File Name Edit box followed by ENTER. Now you are ready to insert the individual message in the Notepad.jsm message file. To add the individual message, perform the following:

Press UP ARROW until you reach the line entitled "Messages."

Press END to move to the end of the line.

Press ENTER twice to create two blank lines.

Type the following message:

@MsgVirtualViewer

This text is being displayed in the virtual viewer. You can use the virtual viewer to display helpful messages and other information.

Press ESC to close this message.

@@

Press CTRL+S to save the file.

Press CTRL+TAB to move back to the Notepad.jss script source file.

NOTE: After you have moved back to the Notepad.jss script file, be sure to include the Notepad.jsm file. Failure to do so will result in errors at the time of compilation. For more information on the include statement see 5.0 Creating Scripts. After you have added the message to the Notepad.jsm message file, you are ready to create the script. Your script should begin by determining if the virtual viewer is currently being displayed. If it is, then use the UserBufferDeactivate function to close the viewer before displaying the new message. After you have determined if the viewer is active and closed it if necessary, you should use the SayFormattedMessage function to display the message in the virtual viewer.

DisplayVirtualViewer Script Documentation:

Script Name: DisplayVirtualViewer

Can be Attached to Key: checked

Synopsis: Displays a message in the virtual viewer.

Description: Displays a message in the virtual viewer using the SayFormattedMessage function.

Category: None

Assign to: CTRL+SHIFT+V

DisplayVirtualViewer Script:

Script DisplayVirtualViewer ()

If UserBufferIsActive () Then

UserBufferDeactivate ()

EndIf

SayFormattedMessage (OT\_USER\_BUFFER, MsgVirtualViewer)

EndScript

**Exercise 11.2: Displaying Keystrokes in the Virtual Viewer**

The objective of this exercise is to take the previous exercise and add a keystroke link to the message JAWS displays in the virtual viewer. To add a keystroke to the message, you need only modify the message, save the message file and recompile the Notepad.jss script file. Start by performing the following steps to modify the message from the previous exercise:

Press CTRL+TAB until you reach the Notepad.jsm file.

Press CTRL+END to move the insertion point to the bottom of the file.

Press UP ARROW until you reach the line containing the text of “Press escape to close this message.”

Press HOME to move to the beginning of the line followed by ENTER twice. This will insert two blank lines above the current line of text.

Press UP ARROW to move to one of the blank lines.

Type the following text on this line:

Press %KeyFor(DisplayVirtualViewer) to redisplay this message

Follow this text by pressing ENTER twice. After you complete the above steps, your message should look like the following:

@MsgVirtualViewer

This text is being displayed in the virtual viewer. You can use the virtual viewer to display helpful messages and other information.

Press %KeyFor(DisplayVirtualViewer) to redisplay this message.

Press ESC to close this message.

@@

After you have modified the individual message, press CTRL+S to save the message file. You then need to recompile the Notepad script file. When you receive the “compile complete” message, you can move back to Notepad. Test the script by pressing the keystroke to activate the script. Your message should be displayed in the virtual viewer. Use the arrow keys to move through the message. Did you hear JAWS speak the keystroke as a link? If so, then press ENTER on the link to redisplay the message. If not, then return to the Script Manager and make sure your message is identical to the message shown above.

**12.0 UNDERSTANDING FUNCTIONS**

In previous chapters, you created scripts that performed a variety of actions ranging from speaking messages to typing keystrokes and more. Each time you wanted to perform the script, you pressed the assigned keystroke and JAWS performed the associated script. As you created these scripts, you called upon "built-in" functions. These types of functions are actually part of JAWS. Unlike the scripts you previously created, you will not find built-in functions within a JAWS script source file. While built-in functions are not contained within the JAWS script source file, "user-defined functions" are.

User-defined functions are written both by Freedom Scientific and JAWS script developers. You create user-defined functions using the Script Manager. In addition to built-in and user-defined functions, there are also "event" functions. JAWS automatically performs event functions each time an associated event occurs in the Windows operating system. For example, each time you press TAB to move from control to control within a dialog, you are causing the "focus changed" event to occur. Although you can see these functions within the script files that come with JAWS, you cannot create your own event functions.

This chapter introduces you to built-in and user-defined functions. You will gain a better understanding of built-in functions and their use. You will also learn how to create a user-defined function, pass parameters to a function, and return values using functions.

**Table of Contents**

Chapter 12.0 Understanding Functions, contains the following sections:

12.1 Functions, An Overview

12.2 Built-in Functions

12.3 User-defined Functions

12.4 Functions that Return Values

12.5 The Use Statement

12.6 Chapter Exercises

**12.1 FUNCTIONS, AN OVERVIEW**

Functions differ from scripts in three ways. Unlike scripts, functions are not activated by a keystroke. Instead, they are called from a script or another user-defined function. When you call a function, the function performs its designated task and then returns control to the calling script or function. Second, functions can accept parameters. A parameter is data the function uses to complete its task. For example, if you want JAWS to speak a message, you use the SayFormattedMessage function. In order for this function to perform its task of speaking the desired message, you must provide the text of the message along with the desired output type. JAWS then uses the text and desired output type to determine if the message should be spoken and then speak it. Without this information, JAWS cannot perform the task of the function.

Finally a function differs from a script in that it can also send a value back to the calling script. This value is referred to the function's return or return value. For example, the GetLine built-in function retrieves the line of text found at the location of the active cursor. When it determines the correct text, the function returns the text as a string that you can then store in a local string variable. You create functions to carry out repetitive tasks. Instead of placing the same code in any number of different scripts, you can put the code in one function and then call it from those scripts. This makes maintaining your scripts and user-defined functions much easier. Suppose you create scripts and user-defined functions for an application. Within the script file for that application, you have placed the same code in several different scripts. Now suppose there is a newer version of the application available. You discover that your scripts no longer perform as expected and need to be updated due to the changes in the application. Since you placed the same code in several scripts, you must modify each of them and then test those modifications. On the other hand, had you placed the same code in a single user-defined function, then you would have only had to modify and test one block of code.

**12.2 BUILT-IN FUNCTIONS**

JAWS comes with over 800 built-in functions available for your use. These functions are called "built-in" because they are actually a part of JAWS. Since these functions are a part of JAWS, you cannot actually see the statements that comprise them. The tasks built-in functions perform range from speaking messages to retrieving window information. Consider these functions as the building blocks of your scripts and user-defined functions. You will find that a number of built-in functions require parameters to complete their designated task. Some built-in functions may only require a single parameter while other functions may require multiple parameters. The type of parameters the function requires can be any one of the four data types: handle, integer, object or string. When a function requires multiple parameters, they can consist of any combination of the four data types. For example, the TypeString built-in function requires a single string parameter, the text to be typed, to accomplish its task. You also use the SayFormattedMessage function to cause JAWS to speak a message in the earlier exercises within this manual. The function required three parameters to accomplish its task. The first parameter is of the type of an integer. The function uses this parameter to determine the type of message that is to be spoken by JAWS. The second and optional third parameters are both springs and contain the actual message JAWS speaks when the function is performed.

Since built-in functions are a part of JAWS, you must review the Builtin.jsd script documentation file to determine the type and number of parameters a specific built-in function uses. Many built-in functions also return values to the calling script or user-defined function. You can then use these return values within your scripts to perform other tasks. These values can range from the block of text found at the location of the active cursor to the window handle found at the location of the PC cursor. To learn more about the built-in functions that are a part of JAWS, do one of the following:

Open the Builtin.jsd file within Script Manager

Review the list of functions in the Insert Function dialog within Script Manager

Review the FSDN found in the Downloads section of the Freedom Scientific website

The Builtin.jsd File

Like each script file, the built-in functions have a corresponding JAWS script documentation (jsd) file. This file is called Builtin.jsd and you can find it in your JAWS shared settings folder. You can view the contents of this file using the script Manager or your favorite text editor. This file contains information on all the built-in functions available for your use in creating scripts and user-defined functions. The information contained in the file includes the function name, synopsis and description, category, return type and description and any parameters used by the function.

You can navigate this file just as you would any other text file. You can perform searches within the file using the find feature in either the Script Manager or your favorite text editor. But you should exercise care when you view this file. If you alter its contents, you may interfere with the normal operation of JAWS.

The Insert Function Dialog. You can also use the information displayed in the Insert Function dialog to learn more about any of the built-in functions. JAWS retrieves the information displayed in this dialog from the Builtin.jsd file. After you have selected the desired function, press TAB to move to the Description edit box. You can use any of the normal navigation keys such as the ARROW Keys, HOME and END to read the text contained in this edit box. Pressing TAB from the Description edit box takes you to the Returns edit box. If the function returns anything to the calling script or user-defined function, then this edit box will contain the return type and its description.

Note: When you move to either the Description or Returns edit boxes, the PC cursor is placed at the bottom of the text. Press CTRL+HOME to move to the beginning of the text. There are a couple of drawbacks to using the Insert Function dialog to view information about built-in functions. First, any user-defined functions contained within the current application-specific or default script files are also listed with the built-in functions. This makes it difficult to differentiate between built-in functions and any user-defined functions contained in the current application-specific or default script file. Second, the parameters for the built-in functions are not displayed in this dialog. This means that if you want to view information about any parameters the function uses, you must actually insert the function into your script and wait for the Script Manager to prompt you for each parameter.

The FSDN.

The Freedom Scientific Developers Network (FSDN) is a collection of all of the built-in functions and their corresponding documentation. The document is in the Windows Help format (CHM) and can be downloaded from the Freedom Scientific website. Functions are grouped by category and include all documentation including parameter information and return information. Use the FSDN as an easy way to browse through functions by category.

**12.3 USER DEFINED FUNCTIONS**

In cases where built-in functions don't perform the task you need, you can create your own functions. These types of functions are called "user-defined" functions. Unlike built-in functions, any script developer can modify the contents of a user-defined function. You create user-defined functions within your script file using the same methods you would when creating a script. However, the New Script dialog changes its contents when you create your function. Instead of prompting you for a keystroke to assign to the function, the New script dialog prompts you for a return type and description. For more information on the New Script dialog, see section 5.2 The New Script Dialog.

Basic User-Defined Functions. The simplest form of a user-defined function is one that does not require any parameters to complete its task. These same functions also do not return any values to the calling script or user-defined function. This type of function is often referred to as a "void" function.

When you create this type of user-defined function, you need only use the General Page of the New Script multi-page dialog. After you add the name, synopsis, description and category for the function, you should select the "void" return type from the Function Returns edit combo box. This return type tells JAWS that the function does not return any information to the calling script or function. When you create a function that does not require any parameters or return any values, the script Manager inserts the following beginning line of the function into your script file:

Void Function MyFunction ()

The key word "void" preceding the key word "Function" indicates the function does not return anything to the calling script or user-defined function. The empty pair of parentheses following the function name indicate the function does not require any information to perform its task. An example of a function that does not require any parameters or return a value follows:

void Function GeneralJAWSHotKeys ()

If UserBufferIsActive () then

UserBufferDeactivate ()

EndIf

SayFormattedMessage (OT\_USER\_BUFFER, cmsgHotKeyHelp1\_L, cmsgHotKeyHelp1\_S)

EndFunction

You can find this function in the Default.jss script source file. The HotKeyHelp script calls this function to display a list of JAWS hot keys in the virtual viewer. Since the function only displays information, it does not require any data from the calling script to perform its task. When the function finishes, it does not return anything to the calling script.

Functions with Parameters. When you create a user-defined function, your goal is to be able to call this function from a variety of scripts and other user-defined functions. In order for the function to determine what it should do, you may need to pass some information to the function. Just like the built-in functions discussed earlier in this chapter, user-defined functions can accept parameters. A parameter is a piece of information that is passed to a function to allow it to complete its task. A parameter can be any one of the four data types: handle, integer, object, or string. See 7.0 Using Variables and Constants, for more information on available variable types.

You use the New Script dialog to create parameters for your user-defined functions. After you have added all the necessary function information in the General page, press CTRL+TAB to move to the parameters page. When you first enter this page, the focus is placed in the existing Parameters list box. This list box displays all existing parameters for the function. This list box is empty when you create your new function but may be populated with information for an existing function. To add parameters to a function, perform the following:

Press TAB to move from the Existing Parameters list box to the New Parameter edit box.

Give the parameter a meaningful name. Follow the same rules for naming parameters as you do for naming variables. You should use Hungarian notation to denote the type of parameter. You should also avoid using spaces or any punctuation marks other than an underline (\_) characters. For example, you could give a string parameter the name of sName as it stores a string value containing a name. See 7.0 Using Variables and Constants for more information.

Press TAB to move to the By Reference check box. Leave this check box cleared as this allows for a one-way data exchange between the calling script or user-defined function and your new function.

Press TAB to move to the Parameter Description edit box. Type a meaningful description in this edit box as this is used by the Script Manager when you insert the function into a script.

Press TAB to move to the available Types combo box. Select the appropriate type for the parameter. For example, the sName parameter discussed above would have a type of string.

Press TAB to move to the Add Parameter button. Press SPACEBAR to activate this button. After you activate this button, the Script Manager returns the focus to the New Parameter edit box in preparation for the addition of another parameter.

After you have finished adding all necessary parameters, press TAB until you reach the OK button. Press SPACEBAR to activate the button and the Script Manager inserts the new function into your script file. When you declare a parameter using the New Script dialog, the Script Manager places the type and name of the parameter between the parentheses following the function name. An example of a beginning line of a function that uses a single string parameter follows:

Void Function MyFunction (string sParameter)

In the line above, the parameter, sParameter, and its type, string, are placed between the parentheses. As you add statements to your function to perform the given task, you refer to the parameter using the name that appears between the parentheses. Using the example shown above, you would refer to sParameter within the body of the function any time you want to determine the value it contains just as you would if it were declared as a local variable. An example of a function from the default script file using a single parameter follows:

Void Function InitializeGlobalVoiceSettings (int iParamToSet)

If iParamToSet == V\_RATE then

let GlobalPcRate = GetVoiceRate (VCTX\_PCCURSOR)

let GlobalJawsRate = GetVoiceRate (VCTX\_JAWSCURSOR)

let GlobalKeyboardRate = GetVoiceRate (VCTX\_KEYBOARD)

let GlobalMessageRate = GetVoiceRate (VCTX\_MESSAGE)

ElIf iParamToSet == V\_VOLUME then

let GlobalPcVolume = GetVoiceVolume (VCTX\_PCCURSOR)

let GlobalJawsVolume = GetVoiceVolume (VCTX\_JAWSCURSOR)

let GlobalKeyboardVolume = GetVoiceVolume (VCTX\_KEYBOARD)

let GlobalMessageVolume = GetVoiceVolume (VCTX\_MESSAGE)

ElIf iParamToSet == V\_PITCH then

let GlobalPcPitch = GetVoicePitch (VCTX\_PCCURSOR)

let GlobalJawsPitch = GetVoicePitch (VCTX\_JAWSCURSOR)

let GlobalKeyboardPitch = GetVoicePitch (VCTX\_KEYBOARD)

let GlobalMessagePitch = GetVoicePitch (VCTX\_MESSAGE)

endIf

EndFunction

In the function shown above, the iParamToSet parameter contains the value used to determine the set of global variables to be initialized. Since the speech characteristic can be different each time the function is called, a parameter of type integer was used. The If statement contained within the function determines which speech characteristic should have its corresponding global variables initialized. Notice that the parameter is referred to just as if it had been declared as a local variable within the body of the function. When you declare multiple parameters, the Script Manager places a comma after each parameter declaration except for the last in the list. Essentially, a list of parameters is identical to a list of local or global declarations in that each declaration is followed by a comma except for the last. The only difference is that you do not use the key words "Var" or "Globals" to begin the parameter declarations. Instead, a pair of parentheses surrounds the parameters in the beginning line of the function. An example of a function beginning line containing multiple parameters follows:

Void Function MyFunction (string sParameter1, string sParameter2, string sParameter3)

In the example above, the Script Manager inserted a comma after the first and second parameter declarations, but not the third. The following function beginning lines are taken from the default script file and use multiple parameters to perform their tasks:

Void Function DoCustomHighlightColors (string strForeground, string strBackground, string sApp)

void Function KeyboardHelpHook (string ScriptName, string FrameName)

void Function ProcessSelectText(int nAttributes,string buffer)

**12.4 FUNCTIONS THAT RETURN VALUES**

A return is a value that a function returns to the calling script or function when it completes its task. A return value can be any one of the four variable types: handle, integer, object, or string. The type of value your function returns depends largely on the task it performs. You use the Function Returns edit combo box in the General page of the New Script dialog to tell JAWS the type of value the function returns. You also type the description of the return in the Return description edit box. Adding a description for the return, helps you and anyone else using your function to determine exactly what the value should be used for within the calling script or function.

When you create a new function that returns a string value, the Script Manager places the following function beginning line into your script file:

String Function MyFunction ()

The "string" key word that precedes the "function" key word tells you that the MyFunction function returns a string value to the calling script or user-defined function.

**The Return Statement**

You use the return statement to send a value back to the calling script or user-defined function. This key word tells JAWS to return the specified value to the calling script or function. You can return the value as a literal value or within a variable. The syntax of a return statement that sends a string value stored in a variable called sText back to the calling script or user-defined function follows:

Return sText

A return statement that returns a string of text, "This is a string", as a literal follows:

Return "this is a string"

When a function returns a value to the calling script or user-defined function, you must store that value in either a local or global variable. You can then use that variable to make decisions on what the calling script or user-defined function should do next. An example of storing a string value returned by a function called MyFunction in a local variable follows:

Example 1: Assigning the Output of a Function to a Variable

Script MyScript ()

Var

String sText

Let sText = MyFunction (); store the return value of MyFunction in sText

If sText != "" Then; the function actually returned a value other than null or nothing

SayFormattedMessage (OT\_MESSAGE, sText)

EndIf

EndScript

You can also use the return value of one function as the parameter for another. Using the previous example, the return value from MyFunction could be passed directly to the SayFormattedMessage function without storing it in a local variable first. The only disadvantage to this approach is that the MyFunction function may not retrieve any text. Therefore, the SayFormattedMessage function won't cause JAWS to speak anything. The example reworked to use the return value of MyFunction as the parameter for SayFormattedMessage follows:

Example 2: Using a Function as a Parameter For Another Function

Script MyScript ()

SayFormattedMessage (OT\_MESSAGE, MyFunction()); use the return value from MyFunction as the message text for SayFormattedMessage

EndScript

Another example of a user-defined function that returns a value follows:

string function VerbosityLevelToggle(int iRetCurVal)

Var

int Verbosity

if not iRetCurVal then

; update it

VerbosityLevel ()

endIf

let Verbosity = GetVerbosity ()

if Verbosity == 0 then

return cmsg12\_L; Beginner

elif Verbosity == 1 then

return cmsg13\_L; Intermediate

elif Verbosity == 2 then

return cmsg14\_L; Advanced

endIf

EndFunction

You can find the function shown above in the default script file. JAWS performs this function each time you press SPACEBAR on the "User Verbosity" entry in the Adjust JAWS Options dialog. The function begins by determining the value of the iRetCurVal parameter. If the value of the parameter is 0 or false, then JAWS performs the VerbosityLevel function to change the verbosity level from its current value to the next value. For example, if you are using the beginner verbosity level, the VerbosityLevel function changes your verbosity level to intermediate. Next, the GetVerbosity function retrieves the verbosity setting now being used and stores it in the Verbosity local variable. The If statement then determines the value stored in the Verbosity variable and returns the appropriate string of text to indicate the new verbosity level. The actual text is stored in the Common.jsm JAWS message file found in your JAWS settings folder.

**12.5 THE USE STATEMENT**

In a previous chapter of this manual, you learned about the include statement. You use this statement to "include" the contents of a JAWS script header or message file within a JAWS script file. There is a second statement you can add to your script file that allows you to use another type of Script Manager file. If you were to look at the default script file, you would find a number of include statements for a variety of header and message files. Following these include statements, you will find a series of "use" statements. These "use" statements allow you to create a link between the binary version of one script file with the binary of another. For example, you will find the statement use "Magic.jsb" in the default script file. This statement tells JAWS to create a link between the compiled version of the default script file, Default.jsb, and the Magic.jsb binary file. This link then allows the script developers at Freedom Scientific to call functions from the Magic.jsb file just as if the functions had been created in the default script file or were part of the built-in functions of JAWS.

The syntax of the use statement is identical to that of the include statement. You begin by typing the word "use" followed by a space then the name of the binary file you want to include. The name of the binary file must include the extension of ".jsb" and be surrounded in quotation marks. Some examples of use statements taken from the default script file follow:

use "say.jsb"

use "Braille.jsb"

Use "Virtual.jsb"

**12.6 CHAPTER EXERCISES**

The following exercises will give you practice in creating varying types of user-defined functions. Each exercise creates a user-defined function and a corresponding script to call the function. Each exercise will indicate the script file that should be used.

**Exercise 12.1: Basic User-Defined Function**

The purpose of this exercise is to create a function that speaks a message and then types a string of text into an Notepad document. If you don't already have Notepad open, start Notepad and activate the script Manager from within Notepad. This should open the Notepad.jss script file that contains scripts from the previous exercises in this manual. If you run Script Manager from within Notepad and a blank file is opened, type the following lines at the top of the file:

; Script file for Notepad

; JAWS version 12.0

include "hjconst.jsh"

include "common.jsm"

First, create the function that speaks a message and then types a string of text into Notepad. Use the SayFormattedMessage function to speak the message and the TypeString function to type the text.

SpeakGreeting Function Documentation:

Script Name: SpeakGreeting

Can be Attached to Key: not checked

Synopsis: Speaks a message and types a string of text.

Description: Speaks a message and types a string of text into an open Notepad text document.

Category: none

Function Returns: void

Return Description: none

SpeakGreeting Function:

Void Function SpeakGreeting ()

SayFormattedMessage (OT\_MESSAGE, "Hello world, it's a great day for writing scripts.")

TypeString ("Hello world, it's a great day for writing scripts.")

EndFunction

Once you have successfully compiled your function, you will need to create the script to call it. Use the Insert Function dialog to insert the SpeakMessage function into the body of your script. This is the only statement that goes into the body of your script as the function performs the desired task.

TypeAndSpeakGreeting Script Documentation:

Script Name: TypeAndSpeakGreeting

Can be Attached to Key: checked

Synopsis: Calls the SpeakGreeting function to speak a message and type text.

Description: Calls the SpeakGreeting function to speak a message and type text in Notepad.

Category: none

Assign to: CTRL+SHIFT+G

TypeAndSpeakGreeting Script:

Script TypeAndSpeakGreeting ()

SpeakGreeting ()

EndScript

**Exercise 12.2: User-Defined Function with a Single Parameter**

The purpose of this exercise is to create a user-defined function that speaks a message using a string of text passed to it as a parameter. If you don't already have Notepad open, start Notepad and activate the script Manager from within Notepad. Before you create your user-defined function and related script, you need to create the message used by your function in the Notepad.jsm message file. You can open the file within the Script Manager and add your message. If you already have a Notepad.jsm message file, do the following to open the file in the Script Manager:

Press CTRL+O to display the Open File dialog.

Type Notepad.jsm in the File Name edit box and press ENTER. If you have not previously created the Notepad message file, perform the following:

Press CTRL+N to display the New File dialog.

The active control is a list of file types. Press M to select Messages followed by ENTER. This will open an untitled message file in the Script Manager.

Type the following text in the file:

; Message file for Notepad

Messages

EndMessages

Press CTRL+S to save the file. JAWS prompts you for the filename. Type Notepad in the File Name Edit box followed by ENTER.

Now you are ready to insert the individual message in the Notepad.jsm message file. To add the individual message, perform the following:

Press UP ARROW until you reach the line entitled "Messages."

Press END to move to the end of the line.

Press ENTER twice to create two blank lines.

Type the following messages:

@MsgName

Hello world, my name is %1.

@@

; message for missing or no name

@MsgNoName

You did not enter a name.

@@

Press CTRL+S to save the file.

Press CTRL+TAB to move back to the Notepad.jss script source file.

NOTE: After you have moved back to the Notepad.jss script file, be sure to include the Notepad.jsm file if you haven't done so already. Failure to do so will result in errors at the time of compilation. For more information on the include statement see Chapter 5 Creating Scripts. Now you are ready to create the function that will speak your message and the script that calls the function. Your function should accept a single string parameter: the name passed to it from the calling script. Before the function speaks the message, check to make sure the parameter actually contains some text using an If statement. If there is no text in the parameter, speak an error message and use the return statement to exit the function. Use the FormatString function to combine the passed string parameter and the MsgName message you created above into one message. Next, call the SayFormattedMessage function to speak the message after it has been formatted.

SpeakTheName Function Documentation:

Script Name: SpeakTheName

Can be Attached to Key: not checked

Synopsis: Speaks a message that includes a name.

Description: Speaks a Hello World message using a string parameter passed to the function containing a name.

Category: none

Function Returns: void

Return Description: none

After you have entered the information shown above into the General page of the New Script dialog, press CTRL+TAB to move to the Parameters page. You will use this page to tell JAWS the name and type of the parameter the function needs to perform its task. When you reach this page, press TAB to move to the New Parameter edit box and add the following information:

New Parameter: sName

By Reference: not checked

Description: String containing the name to be combined with the message and spoken.

Available Types: string

After you have entered the information shown above, press ALT+A to activate the Add button and add your new parameter. Next, press TAB until you reach the Ok button. Press SPACEBAR to activate the button and close the New Script dialog.

SpeakTheName Function:

Void Function SpeakTheName (string sName)

var

String sMessage

If sName == "" Then; check to make sure text was actually passed in the sName parameter

SayFormattedMessage (OT\_ERROR, MsgNoName); speak an error message

Return; exit the function

EndIf

Let sMessage = FormatString (MsgName, sName); format the message using the passed parameter

SayFormattedMessage (OT\_MESSAGE, sMessage)

EndFunction

Once you have successfully compiled the SpeakTheName function, you will be ready to create the script that calls the function. Your script should assign a value to a variable called sName. Use the Insert Function dialog to insert the SpeakTheName function. The variable sName will be the parameter to the SpeakTheName function.

SayName Script Documentation:

Script Name: SayName

Can be Attached to Key: checked

Synopsis: Prompts you for a name to be spoken by JAWS.

Description: Displays a dialog box that allows you to enter your name. JAWS then speaks a message containing the name you entered.

Category: none

Assign to: CTRL+SHIFT+N

SayName Script:

Script SayName ()

var

String sName

let sName = "Your name goes here"

SpeakTheName (sName)

EndScript

**Exercise 12.3: User-Defined Function with Multiple Parameters**

In the previous exercise, you created a function that accepted a string parameter containing your name. This parameter could hold only your first name or the combination of your first and last names. In this exercise, modify the SpeakTheName function to accept two string parameters. The original parameter of the function, sName, will be used to hold the first name. The second parameter created for this exercise, will hold the last name. Before you modify the function, you will need to change the content of the MsgName message contained in the Notepad message file. Press CTRL+TAB to move from the Notepad.jss file to the Notepad.jsm file. In the Notepad.jsm message file, change the MsgName message to look like the following:

@MsgName

Hello world, my name is %1 %2.

@@

Press CTRL+S to save your changes followed by CTRL+TAB to move back to the Notepad script file. Now you are ready to modify the existing SpeakTheName function. To add an additional parameter to the function, perform the following:

Move to the body of the SpeakTheName function.

Press CTRL+D to display the script Documentation multi-page dialog.

Press CTRL+TAB to move to the Parameters page. When you move to this page, the focus is placed in the Existing Parameters list box.

Press TAB to move to the New Parameter edit box.

Add the following information for your new parameter:

New Parameter: sLastName

By Reference: not checked

Description: String containing the last name.

Available Types: string

After you have added all the necessary information for the new parameter, press ALT+A to activate the Add button. This moves the focus back to the New Parameter edit box. Press TAB until you reach the Ok button. Press SPACEBAR to activate the button and close the Script Documentation multipage dialog.

Move the insertion point to the beginning line of the SpeakTheName function. It should now look like the following:

Void Function SpeakTheName (string sName, string sLastName)

You are now ready to modify the function to use the second parameter. First, you should add a second If statement to check if anything was passed in the sLastName parameter. Move your insertion point to the line containing the EndIf statement. Press END to move to the end of the line followed by ENTER to create a new blank line. Add the following statements to the function:

If sLastName == "" Then

SayFormattedMessage (OT\_ERROR, MsgNoName); speak an error message

Return; exit the function

EndIf

Second, you will need to add a third parameter to the call to the FormatString function. Move your insertion point to the line containing the FormatString function and change it to look like the following:

Let sMessage = FormatString (MsgName, sName, sLastName) ; format the message using the passed parameters

After you modify the function to use the new parameter, compile the script file. Once you have compiled the file successfully, you will be ready to modify the script.

NOTE: You can alter the messages used to speak the error messages when nothing is contained in either the first or last name parameters. To do this, you would add a second message to the Notepad.jsm message file for the last name and alter the contents of the message for the first name. Then you could use each message appropriately to speak the correct error message. To modify the script you will need to add a second variable to hold the last name. Use a Let statement to assign a value to the variable and pass it as the second parameter into the SpeakTheName function. Modify the SayName script to look like the following:

Script SayName ()

var

String sLastName,

String sName

let sName = "Your first name goes here"

let sLastName = "Your last name goes here"

SpeakTheName (sName, sLastName)

EndScript

**Exercise 12.4: User-Defined Function with a Return Value**

The purpose of this exercise is to create a function that returns a value to the calling script. In the previous 2 exercises, you created a script and function that used 2 parameters to speak a message. The script used the Let statement to assign values to variables which were then passed into the function to speak the message. In this exercise, you will create a function that uses the InputBox function to prompt you for your first and last names. The function will then format the message to be spoken using the FormatString function. The FormatString function uses the MsgName message from the previous exercise along with the values for the first and last names entered using the InputBox function. Finally, the function returns the message as a string to the calling script.

The InputBox function is a built-in function that displays a dialog box with an edit field for entering text. Using the parameters of the function, you pass strings of text to be used as the title of the dialog box and the prompt for the edit field. The third parameter should be a variable name that will contain the text entered when the OK button is activated in the dialog box. An example of the InputBox function follows:

InputBox ("Name entry", "Please enter your first name", $sName)

In the example above, the title of the dialog box will be "Name entry." The text "Please enter your first name" will be the prompt for the edit box. sName will contain the text entered by the user. If no text is entered, sName will be empty.

RetrieveTheName Function Documentation:

Script Name: RetrieveTheName

Can be Attached to Key: not checked

Synopsis: Prompts you for your first and last names.

Description: Uses the InputBox function to prompt you for your first and last names.

Category: none

Function Returns: string

Return Description: String of text containing the formatted message to be used by any of the Say functions.

RetrieveTheName Function:

String Function RetrieveTheName ()

var

String sFirstName,

string sLastName,

string sMessage,

string sPrompt,

string sTitle

let sTitle = "Name Entry"

; retrieve the first name

let sPrompt = "Enter your first name:"

InputBox (sPrompt, sTitle, sFirstName)

; retrieve the last name

let sPrompt = "Enter your last name:"

InputBox (sPrompt, sTitle, sLastName)

; format the message

let sMessage = FormatString (MsgName, sFirstName, sLastName)

; return the formatted message

return sMessage

EndFunction

Once you have compiled the function successfully, you will be ready to create the script that calls it. You will need to declare a single string variable to hold the message that is returned from the function. Your script should then call the function and store the returned message in the local variable. Finally, your script will use the SayFormattedMessage function to speak the message.

SpeakFormattedMessage Script Documentation:

Script Name: SpeakFormattedMessage

Can be Attached to Key: checked

Synopsis: Speaks a message containing first and last names.

Description: Speaks a message returned from the RetrieveTheName function.

Category: none

Assign to: CTRL+SHIFT+M

SpeakFormattedMessage Script:

Script SpeakFormattedMessage ()

var

String sMessage

Let sMessage = RetrieveTheName (); call the function to retrieve and format the message

SayFormattedMessage (OT\_MESSAGE, sMessage); speak the message

EndScript.

**13.0 WINDOWS PROGRAM STRUCTURE**

Most people who use the Windows operating system never really understand what the word "window" actually means. Most users realize the rectangular box that appears on the screen when a new application is opened is a window. They probably think of the text area of their word processor as a window as well. In most applications, every dialog box, menu, button, edit field, list box, etc. is actually a separate window with one or more identifiers that can be used to refer to it. However, a programmer may choose to create controls, such as entry fields, without making them actual windows. Programs like this are the most difficult to make accessible because there's no simple way to identify and speak the various controls. In this chapter you will learn about Windows program structure and hierarchy. You will learn about different types of windows and how they are identified by the operating system. You will learn how to determine characteristics that uniquely identify windows. You will also learn how to use JAWS to explore the window structure of an application.

**Table of Contents**

Chapter 13.0 Windows Program Structure, contains the following sections:

13.1 Windows Hierarchy

13.2 Reassigning Window Classes

13.3 The Script Utility Mode

13.4 Customizing an Unknown Application

13.5 Debugging Script Files

**13.1 WINDOWS HIERARCHY**

The operating system does not randomly place the various windows that may be present on your screen at any one time. They are interrelated with a special hierarchy that allows the Windows operating system and the programmers who work within it to keep track of each window. You can think of these inter-relationships as one of parents and children. The desktop is the parent of all application windows. Thus, the JAWS window, your word processor's window and the Internet browser's window are all children of the desktop. These windows are all one level down from the desktop, which makes them all children of the desktop at the same logical level. It also means that all of them are peers of each other. When you later encounter terms such as PriorWindow and NextWindow, the reference is to moving among peer windows at the same logical level. The word processor window has several child windows, the text area, the menu bar, the toolbar and the status line. These are all children of the word processor window and are all children at the same logical level. When you press the keystroke to open a file in your word processor, the word processor displays a child window that prompts you for the file to open. This window is often referred to as a dialog box, but it is a child window to the word processor, its parent.

If you think of the Open File dialog box as a parent then you will find that the File Name edit box and all other controls are children of the open file dialog box. These are all children of the parent dialog and are at the same logical level. Each window is a child to the window one level up that spawned or generated it, and each window is a parent to the windows one level down from it. You can find a good analogy of the parent-child relationship by using the file folder structure of your PC. On your disk drive, the C:\ drive is typically the parent of all other folders on your drive. All of the subfolders one level down from C:\ are children of that parent and are at the same logical level. Thus, if you have subfolders named Documents and Settings, Program Files and Windows on your system, they are all one level down from C:\. They are all children of C:\ and are all one logical level down. For the purposes of folder tree structure and window hierarchy, you can assume that being at the same logical level means that all windows were spawned or generated by the same parent. When you open the Program Files subfolder, you should find subfolders such as Common Files, Freedom Scientific and Internet Explorer. These three subfolders are all one level down from Program Files, are children of Program Files, and are at the same logical level as each other, two levels down from C:\. This can continue until the lowest logical level of a particular branch is reached.

The parent/child structure of Windows works much the same way as a folder structure. Each application is a branch of the tree, and child windows are spawned from it. All windows one level down from the parent application are children at the same logical level. Each of these children can spawn children of their own, and these children are three levels down from the parent application. Another example of a parent child relationship is the world we live in. Think of your house as being at the bottom end of a hierarchy. The parent of your house is the city you live in. All the other houses in your city are children of your city and are at the same logical level. For example, all the houses in St. Petersburg are at the same logical level. They all have the same parent which is the city of St. Petersburg. The parent of your city is the state you live in. Every city is at the same logical level and has the same parent. The parent of St. Petersburg is the state of Florida. All the cities in Florida are at the same logical level and have the same parent. The parent of each state is your country. Every state is at the same logical level and is a child of its country. The parent of Florida is the United States. All the states in the US are at the same logical level and have the same parent. You can think of the planet itself as being the parent of all the countries. As you can see, the hierarchy can contain many different levels.

Identifying Windows.

An application programmer assigns a number of identifiers to each window. These identifiers allow the window to be referred to individually. Each window has a class, type, type code, window subtype code, and control ID. These window categories are important because JAWS behaves differently, depending on what category of window is active. For example, if an edit field is active, JAWS tracks the insertion point. If you press DOWN ARROW, JAWS speaks the line of text to which the insertion point moves. When a list box or menu is active, JAWS tracks the highlight or lightbar as the arrow keys are used to move it up and down the list. JAWS knows to do these things because of the scripts that are associated with the up and down arrow keys in the default script file. These scripts contain logic that decides what to do based on the type of window that has the focus when one of these keys is used.

**The Window Handle**

The window handle is a unique number assigned by the operating system to each window when it is created during a given session. Window handles change each time the window is created. However, the handle value remains constant as long as the window exists. As long as a window exists, you can use the handle to refer to and identify that window. But when a window is closed, it relinquishes its handle. If the window is created later, the operating system assigns a different handle.

You can retrieve the window handle using the built-in functions, GetCurrentWindow and GetFocus. There is a distinct difference between these two functions. The GetFocus function retrieves the handle of the window that contains the insertion point or system focus. The GetCurrentWindow function retrieves the handle of the window containing the active cursor. Many of the JAWS functions used to manipulate windows use the window handle as a parameter. You can use the GetFocus and GetCurrentWindow functions as parameters to other window related functions. An example of how the GetFocus function is used follows:

Example 1: Using the GetFocus Function

Script GetWindowFocus ()

var

handle hFocusWindow

let hFocusWindow = GetFocus ()

; the GetFocus statement could be replaced by the following line:

; let hFocusWindow = GetCurrentWindow ()

SayInteger (hFocusWindow); speak the value of the window handle

EndScript

**Window Class**

The window class is a string of text that denotes some information about a window and what it does. A window class can be an edit field, list box, or a button. It provides valuable information about the window. However, the window class does not always provide specific information. Checking the window class will return the same information for a button, radio button, and checkbox all of which have a Window Class of button. JAWS recognizes 30 different classes. Many times, programmers use custom window classes that JAWS cannot recognize. In these cases, you must tell JAWS how to treat the unrecognized class. After you tell JAWS how to treat a custom window class, JAWS reacts to the window class just as if it were a recognized window class. You can retrieve the Window Class with the function, GetWindowClass. An example of how the window class is retrieved follows:

Example 2: Using the GetWindowClass Function

Script RetrieveWindowClass

var

string sTheClass

let sTheClass = GetWindowClass (GetFocus ())

If sTheClass == “Edit” Then ; the window class is an edit box

Say ("Focus is in an edit box." OT\_MESSAGE)

Else

Say ("The focus is not in an edit box.", OT\_MESSAGE) ;the window class is something other than Edit

EndIf

EndScript

**Window Type**

The window type is a string of text that describes the window. Occasionally, the window type reveals more specific information about certain windows. The window type and window class are the same if no more specific information is available. For example, the window class and window type of an edit box are exactly the same. You can use the window type to differentiate between a checkbox and a radio button. You use the function GetWindowType to retrieve the Window Type. The GetWindowType function returns the string of "unknown window type" when the application programmer used a custom window class to define the window. The Window Type is always in English and is not very useful for those using JAWS in other languages. An example of how the GetWindowType function is used follows:

Example 3: Using the GetWindowType Function

Script RetrieveWindowType ()

Var

string sTheType

Let sTheType = GetWindowType (GetFocus ())

If sTheType == "Edit" Then ; the window type returned is an edit box

Say ("The focus is in an edit box", OT\_MESSAGE)

Else

Say ("The focus is not in an edit box.", OT\_MESSAGE) ; a window type other than edit was found

EndIf

EndScript

**Window Type Code**

The window type code is a numeric value instead of a string. This number is translated into a recognizable string by constant definitions found in the file, HJConst.jsh. For example, in this file you can find lines such as WT\_BUTTON=1, WT\_COMBOBOX=2, WT\_EDIT=3, WT\_LISTBOX=4, and WT\_SCROLLBAR=5. The WT prefix of the constant indicates that this is a constant that refers to a window type code. While the window type code does not provide any more information than the window type, it does have the advantage of being language independent. You retrieve the window type code with the function, GetWindowTypeCode. An example of how the GetWindowTypeCode function is used follows:

Example 4: Using the GetWindowTypeCode Function

Script RetrieveWindowTypeCode ()

Var

int iTheTypeCode

Let iTheTypeCode = GetWindowTypeCode (GetFocus ())

If iTheTypeCode == WT\_Edit Then

; the window type code of 3 was found. The constant, WT\_Edit

; is defined in HJConst.jsh and given a value of 3.

Say (“The focus is in an edit box.”, OT\_MESSAGE)

Else

Say ("The focus is not in an edit box.", OT\_MESSAGE)

; a window type code other than 3 was found, so focus is not in an edit box

EndIf

EndScript

**Window Subtype Code**

Like the window type code, the window subtype code is also a numeric value. However, the window subtype code may provide more detailed information. If no subtype code is available for the window, then the window type code and window subtype code are identical. You should use the window sub type code as it generally provides more specific information about a window than does the window type code. You can use the same constant definitions to refer to window subtype codes as you can for window type codes in your scripts and functions. For example, a radio button has a window type code of 1 (WT\_Button) but a subtype code of 19 (WT\_RadioButton). Window subtype codes are also language independent. You use the function, GetWindowSubTypeCode, to retrieve the window subtype code for a given window. An example of the use of the GetWindowSubTypeCode function follows:

Example 5: Using the GetWindowSubTypeCode Function

Script RetrieveWindowSubTypeCode ()

Var

int iTheSubTypeCode,

int iTheTypeCode

let iTheTypeCode = GetWindowTypeCode (GetFocus ())

let iTheSubTypeCode = GetWindowSubTypeCode (GetFocus ())

If iTheTypeCode == WT\_Button Then

If iTheSubTypeCode == WT\_RadioButton Then

; the window type code of 1 was retrieved. The constant, WT\_Button, is defined in HJConst.jsh and

; given the value of 1.

; the next If-Then statement checks for a subtype code of 19. The constant, WT\_radiobutton is

; given the value of 19 in the HJConst.jsh file.

Say ("The active window is a radio button." OT\_MESSAGE)

Else

Say ("The active window is not a radio button." OT\_MESSAGE)

EndIf

Else

Say ("The active window is not a button." OT\_MESSAGE)

EndIf

EndScript

**Window Control ID**

The control ID is a number assigned by the programmer to each window during application development. The control ID, unlike the window class, has no special meaning, but you can use it to refer to a particular window. One example where the use of control ID’s are extremely useful is in applications where the buttons are labeled with bit map images instead of text labels. Normally, JAWS speaks the text label of a button when you move to it. If the label is a drawing, there is no name to be spoken. By determining the control ID numbers of the various bit map buttons, you can write a script that speaks the name of the button when the focus moves to it. In most cases, the control ID is defined during application development and can be relied upon to uniquely identify a specific window. However, you may find cases where control ID’s have not been assigned by the application developer. In these cases, you will find that each window in a given application has a control ID value of 0. You must then rely on other window identifiers along with the window’s hierarchical position to uniquely identify it.

You may also find that the control ID’s in a given application are dynamically assigned. This means that each time an application window or any of it’s child windows are created, the control ID’s are assigned and do not remain constant. In these cases, you will find that you need to use the other windows identifiers and the window hierarchical position to uniquely identify the window. You can determine if control ID’s are dynamically assigned by running the application, noting the control ID values of some of it’s child windows, then restarting the application and looking at those same control ID values. If they have changed, then you will know that the ID’s are dynamically assigned. You use the function, GetControlID, to retrieve the control ID for a given window. An example of how the GetControlID function is used follows:

Example 6: Using the GetControlID Function

Script IdentifyWindow ()

Var

int iControlID

let iControlID = GetControlID (GetFocus ())

If iControlID == 1100 Then; look for a specific control ID

Say ("This is the file name edit box", OT\_MESSAGE)

; the file name edit box has a control ID of 1100

Else

Say ("This is not the file name edit box.", OT\_MESSAGE)

EndIf

EndScript

**13.2 REASSIGNING WINDOW CLASSES**

There is a standard list of window classes used by most programmers during the development of Windows based applications. The JAWS default scripts contain the necessary logic to allow proper speaking of these standard classes. However, application programmers do not always use standard window classes during application development. Many times, a programmer creates custom window classes that are non-standard. JAWS is unable to recognize the custom class and speak the window correctly. You use JAWS to equate the unknown class to a standard class to allow JAWS to process the window correctly. This process is called reassigning the window class. Once you have reassigned a custom window class, JAWS is able to identify the window and speak it correctly. During window reassignment, you may need to try more than one class to determine what standard class works best.

To reassign a window class, you can use the JAWS, invisible or PC cursor to locate the window. To reassign a window class, do the following:

Move the JAWS, invisible or PC cursor to the window with the non-standard class.

Press INSERT+F2 to activate the Run JAWS Manager dialog.

Press W to move to the Window Class Reassign list box item followed by ENTER.

The Window Classes dialog opens with the New Class edit box being the active control. The non-standard window class is shown in this edit box.

Press TAB to move to the Assign To list box. This list box contains all of the standard window classes JAWS recognizes.

Use your arrow keys to move through the list. Select the window class you think best suits the non-standard window class.

Press TAB to move to the Add Class button and press SPACEBAR. You can also press ENTER from the Assign To list box to activate the Add Class button.

Press SPACEBAR to activate the Ok button to close the dialog. Focus returns to your application.

**13.3 EXPLORING APPLICATIONS WITH THE SCRIPT UTILITY MODE**

The JAWS® screen reader provides you with a powerful utility to quickly analyze an unknown application. This feature is called the Script Utility Mode. When the Script Utility Mode is activated, the keyboard is placed in a shifted state. This utility gives you access to system level window information that you can use when scripting an unknown application. You can determine parent-child relationships, windows identifiers, and retrieve information about Microsoft® Active Accessibility® (MSAA®) objects. Press INSERT+WINDOWS KEY+MINUS to turn on the Script Utility Mode. Press INSERT+WINDOWS KEY+MINUS a second time to turn off the Script Utility Mode.

NOTE: While the Script Utility Mode is active, only part of the keyboard is placed in a shifted state. The keys that are not used by the Script Utility Mode, such as the alphabet, function as normal.

NOTE: The Script Utility Mode was previously called the Home Row Utility and was accessed by pressing INSERT+SPACEBAR in versions of JAWS prior to 11.0.

**Speaking Window Information**

After activating the Script Utility Mode, you can speak various types of information about the active window. Press F1 to speak information about the window. By default, the window and related text are spoken when the utility is activated for the first time. You can cycle through the different pieces of information spoken by pressing F3. The pieces of information spoken are referred to as output modes. Each time you press F3 followed by F1, a different output mode is spoken. Some of the output modes include: window type and text, handle, control ID, class, type, subtype, and real name.

As you cycle through the various output modes, you can press F11 to hear JAWS speak the current output mode. Pressing INSERT+HOME on the number pad will reset the output mode to say type and text. After the keystroke is pressed, JAWS announces the output mode is type and text. This keystroke eliminates the need to press F3 or SHIFT+F3 to return to the say type and text output mode.

You can also copy the spoken information to the clipboard. Press CTRL+F1 to copy the spoken information to the clipboard. You can paste this information into your favorite text editor for review.

**Moving to Windows**

In the windows program structure, there is a parent-child relationship between all windows contained within an application. A parent window can have several child windows. All child windows of a specific parent window are siblings of each other and said to be on the same logical level. Using the Script Utility Mode, you can access each of these windows and speak information about them.

Moving Across the Windows Hierarchy. Using the Script Utility Mode, you can access windows even if they are not accessible through the applications tab order when the Script Utility Mode is turned off. Press TAB to move to the next window on the same level. You can continue to press TAB until JAWS speaks “No next window.” This message indicates you have reached the last window on the same logical level. You can move to the prior window on the same level by pressing SHIFT+TAB. You can continue to press SHIFT+TAB until JAWS speaks “No prior window.” This message indicates you reached the first child window on the same level.

As you press TAB or SHIFT+TAB to move through the windows, JAWS automatically speaks the selected output mode. You can press INSERT+TAB to hear the window type and text spoken by JAWS for the active window. If you are not sure where you began, press F5 to return the Script Utility Mode to the location of the active cursor.

**Moving Up and Down the Windows Hierarchy**

You can use the Script Utility Mode to move up and down the windows hierarchy. This functionality gives you the ability to determine if the window with focus has any child windows associated with it. It also gives you the ability to move to the parent of the active child window. Press F2 to determine if the active window has any associated child windows. When a child window is found, the focus is moved to the child window and JAWS speaks the output mode information about the child window. You can continue to press F2 until “Child window not found,” is spoken by JAWS. This message indicates you are at the bottom of the windows hierarchy and no more child windows are present. Press TAB and SHIFT+TAB to determine if there are other child windows on the same logical level. Press SHIFT+F2 to determine if a child window has an associated parent window. You can continue to press SHIFT+F2 until JAWS speaks "Parent window not found." This message indicates you have reached the top of the windows hierarchy. After you have reached a parent window, press TAB and SHIFT+TAB to determine if there are other windows on the same logical level. Remember that a parent cannot only have child windows, but it can be the child of another parent window. After reviewing the parent-child relationships, you can press F5 to route the Script Utility Mode to the active window. This essentially takes you back to the starting point from which you originally activated the Script Utility Mode.

**Determining Window Visibility**

As you move across, up, and down the windows hierarchy you are moving to windows that may or may not be visible on the screen. You can use the Script Utility Mode to identify the windows that are visible and those that are not. Press F6 to hear the visibility status of the current window announced automatically by JAWS. Each time you move from window to window with this option turned on, JAWS announces the visibility of the window immediately following the output mode information. You can turn this feature off by pressing F6 a second time. You can also announce the visibility status of the current window by pressing F7. This announces the visibility status regardless of the state of the F6 visibility status toggle.

**Node and Tree Capture**

After you have moved to a specific window within the structure of an application, you can use the node capture command to quickly gather all the information about that window. The node capture command places the gathered information into the virtual viewer. You can then use standard reading and selecting commands to read or copy the information to the clipboard. Press ALT+CTRL+ENTER to perform a node capture. Some of the types of information collected by the node capture command includes: window handle, class, type, subtype code, control ID and window name. When you are finished reviewing the information, press ESC to close the Virtual Viewer.

A second and more powerful command is called tree capture. This command allows you to quickly view information about the current window and any child or parent windows associated with it. When you press ALT+CTRL+SHIFT+ENTER to perform a tree capture, the command places all the information for the current window along with all associated child windows into the virtual viewer. You can then use standard JAWS reading and selection commands to read or copy the information to the clipboard. information collected by the tree capture includes: windows handle, class, type and subtype code, control ID, window name, real window name, window hierarchical values and text contained within the window.

**Retrieving MSAA Information**

When the active cursor is on an MSAA object, you can use the Script Utility Mode to provide information about the object. However, you cannot navigate through MSAA objects using the Script Utility Mode. If you need to retrieve information on an object you must first activate that object by moving to it with either the Jaws or PC Cursor.

Press F9 to speak information about the object. By default, the object name is spoken when the utility is activated for the first time. You can move to the next output mode by pressing F10. You can move to the prior output mode by pressing SHIFT+F10. Each time you press either F10 or SHIFT+F10 followed by F9, a different output mode is spoken. Output modes include:

Name

Type

SubType

Value

State

Control Attributes

Description

You can also copy the output mode information about the active object to the clipboard by pressing CTRL+F9. You can then paste the information into a text editor for review after you complete your analysis of the application.

**Script Utility Mode Keystroke Summary**

The following commands are helpful in finding all the information you need to know about controls, parent/child relationships, and text attributes.Keystroke Description

INSERT+H Speak JAWS hot key shortcuts for the Script Utility Mode.

F1 Speak the selected mode of information.

F1 pressed twice in succession Spell the selected mode of information.

INSERT+F1 Display output mode information in the virtual viewer.

CTRL+F1 Copy output mode information to the clipboard.

INSERT+CTRL+ F1 Output Window Technical Info in the virtual viewer (You do not have to be in script utility mode to use this command)

CTRL+ALT+ENTER Activates node capture and places all window information into the virtual Viewer

CTRL+ALT+SHIFT+ENTER Activates tree capture and places all the window information for child or parent windows into the virtual viewer

F2 Move to the first child of the current active window and speak output mode information.

SHIFT+F2 Move to the parent of the current active window and speak output mode information.

F3 Select the next output mode.

SHIFT+F3 Select the previous output mode.

CTRL+F3 When repeatedly pressed, this keystroke moves to the first item of each group in the utility output modes. The modes jumped to are: Window handle, window type and real window name.

SHIFT+CTRL+F3 When repeatedly pressed, this keystroke moves to the prior first item of each group in the utility output modes. The modes jumped to are: Window handle, window type and real window name.

F4 Select the attribute search mode. Attributes include bold, italic, underline, highlight, and strikeout.

F5 Initialize the Script Utility Mode to the window containing the active cursor.

F6 Toggle auto speaking of window visibility status. When this option is turned on, the window's visible status is spoken as you move to it.

F7 Speak the visibility status of the window currently referenced by the Script Utility Mode.

F8 Speak the contents of the window currently referenced by the Script Utility Mode.

F9 Speak the MSAA Object output mode information. The active object is indicated by the position of the active cursor.

INSERT+F9 Display the MSAA object output mode information in the virtual viewer. This is not available on output modes that return integers.

ALT+F9 Displays the list of MSAA objects in the window currently accessed by the Script Utility Mode.

CTRL+F9 Copy requested MSAA object output mode information to the clipboard.

SHIFT+F9 Displays a list of MSAA objects by name in the window currently accessed by the Script Utility Mode. Other pieces of information about the MSAA objects contained in the list include: name, value, subtype code, state and more.

ALT+CTRL+F9 Displays information about the current MSAA object and its hierarchical ancestors or parents in the Virtual Viewer. Level 0 is the current MSAA object, level 1 is its parent, 2 its grand parent and so on.

F10 Select the next MSAA object output mode.

SHIFT+F10 Select the prior MSAA object output mode.

F11 Speak current output mode.

F12 Speak current MSAA object output mode.

TAB Move to next window and speak output mode information.

SHIFT+TAB Move to prior window and speak output mode information.

INSERT+TAB Speak the window prompt and text for the window currently referenced by the Script Utility Mode.

INSERT+NUM PAD HOME Set output mode to SayTypeAndText.

INSERT+7 Activate the window-reclassification dialog.

INSERT+NUM PAD MINUS Route the Jaws or Invisible cursor to the window currently indicated by the Script Utility Mode if it is visible on the screen.

GRAVE ACCENT Move to the next text attribute.

SHIFT+GRAVE ACCENT Move to the prior text attribute.

CTRL+GRAVE ACCENT Move to the first text attribute.

SHIFT+CTRL+ GRAVE ACCENT Move to the last text attribute.

LEFT or RIGHT ARROW keys Changes the value of pixel movement performed by the MouseLeft and MouseRight functions

UP or DOWN ARROW keys Changes the settings for which special characters are spoken

**13.4 CUSTOMIZING AN UNKNOWN APPLICATION**

Many questions should be answered about a new application to determine what customization is necessary. Before you write your first script, you should determine how JAWS works with the application without any customization.

The Main Application Window.

The first place you should start your analysis is the main application window. Begin by activating the menu bar using the ALT key. Explore the main application window to determine if any of the controls can be accessed using TAB or SHIFT+TAB. Note any areas within the main application window that contain static text. You will also need to determine if any of the options in the menu or tool bars offer keyboard access. The following checklist gives you some initial areas to assess how well JAWS performs:

Do menu bars speak automatically when activated?

Do menu options speak automatically when the arrow keys are used to move from option to option?

Can all controls on the main application window be accessed?

Is there keyboard access to menu or toolbar items?

Are there areas of the screen that contain static text that provides useful information during application execution?

If there is static text within the main application window, can the user quickly locate the text?

Are the tool bars and other important graphics labeled?

What information should be shown on the Braille display for quick access?

The above list gives you suggestions for reviewing the main application window of a new and unknown application. You may find that there are other items to check that are not contained on the list. While exploring the application with the JAWS or Invisible Cursors, you may find graphical buttons that are not labeled. You can find out if the button has tool tip text associated with it. In order to have JAWS read tool tip text for a button when the JAWS Cursor is placed on it, you will need to enable the reading of tool tip text in the verbosity preferences found in Settings Center. By default, JAWS is configured not to read tool tip text.

**Dialog Boxes**

After reviewing the main application window, you should then check each of the dialog boxes contained within the application. When the dialog box first opens, does JAWS speak the name and active control automatically? Note whether or not the TAB and SHIFT+TAB keystrokes move the focus from control to control. You should also note any areas within each dialog box that contain static text. You can use the following check list to assess the amount of information spoken by JAWS within dialog boxes:

Does each text label speak?

If it is a multi-page dialog box, do the page tabs read correctly?

If it is a multi-page dialog box, do CTRL+TAB and CTRL+SHIFT+TAB cycle through the tabs?

Does JAWS announce control types correctly?

Can all controls be accessed while moving through the dialog box with TAB and SHIFT+TAB?

Is each control its own window?

Can static text contained within the dialog box be quickly located?

**Application Structure and Windows Identifiers**

After you have made an initial assessment of the application, you will need to look at information for each of the windows contained within. You can use the Script Utility Mode to determine things such as parent-child relationships, control ID values, window classes and more. You should review the main application window and each subsequent dialog box in much the same way as before. However, you will need to note window identification information for each control or window. The following checklist can help you review the application using the Script Utility Mode:

Are control ID’s defined?

If the control ID’s have been assigned, are they dynamic or constant?

Does each window have standard window classes?

Are there other unique characteristics for each window?

Review the parent-child relationships in the main application window and each dialog box.

**Beginning the Customization**

After you have completed a thorough review of the application, you can now begin to think about how best to customize JAWS to work with it. If you found non-standard window classes, you should go back through the application and reassign each non-standard window class to a class that JAWS recognizes. Keep in mind that once you reassign a non-standard class, that reassignment is used for each occurrence of the non-standard window class throughout the application. Many times, reclassifying non-standard window classes significantly improves the ability of JAWS to read window information automatically using the default settings. It is best to be certain that reclassifying windows has not caused unwanted behavior in other parts of the application.

Before you create scripts to read areas of static text within any of the application windows, try creating frames around the text first. Creating frames around static text that is not contained within it’s own window keeps from having to write scripts to access the information. You can then add frame events to monitor for changes within the boundaries of the frame and speak new text automatically when it appears. You can also attach a keystroke to the frame surrounding the text that will allow you to quickly read the information. Keep in mind though that frames are dependent on screen resolution and thus may not work correctly on all computers.

**Customizing JAWS with Scripts**

After you have tried frames and reclassifying non-standard window classes, you may find that you still need to create scripts for the application. Before you begin to look at the built-in functions used to create your scripts, there are a few things you should do. Begin by using pseudo-code to list the steps your script should take to reach the desired outcome. Pseudo-code is nothing more than writing down the steps in short phrases that your script will need to complete the desired action. Write each step of the script down in a list. After you have written down the initial steps the script will need to execute, try to recreate those same steps with JAWS commands using the keyboard. You should also review the notes you made previously when you used the Script Utility Mode. You may be able to take advantage of the relationships between parent and child windows to access information within the application.

After you have either tried recreating the steps using the keyboard or reviewed the program structure, note any additional steps you had to take that are not listed in your pseudo-code. Now you can begin to look at the built-in functions you will need to use to create the script and achieve the desired outcome. You can follow this procedure for each script you need to write to customize JAWS to work with the application.

**13.5 DEBUGGING SCRIPT FILES**

Although the JAWS script language does not have any built in error handling, there are several things you can do prior to testing your new scripts.

**The Desk Check**

Before you test a script, you should check the script to review for logic errors. You can find many logic and syntax errors by doing a thorough "desk check" of the new or modified script. The desk check involves reviewing a new or modified script line by line pretending you are the computer. Verify each statement performs the desired action and in the correct sequence. Correct mistakes as you find them or add comments to the line containing the mistake. Review your notes or pseudo code for the application being customized to make sure you have not left out any vital statements.

**Compiling the Script File**

After you complete the desk check of your script, you can compile the script file. Remember that the script compiler only checks for syntax and will not validate the logic contained within your scripts. The compiler also does not check for correct function names within each script. All Include files are examined as they are included. Include statements should be placed at the beginning of the script file. When a constant or variable is used before an Include statement, the compiler generates an error informing you that the constant or variable is unknown. This type of error is generated because the Include statement is improperly placed.

When the script compiler encounters a syntax error, an error message is displayed on the screen identifying the error. When you clear the dialog box by activating the Ok button, the cursor is placed on the line containing the error. Normally, you can find the error on the line where the cursor is. However, there are times when an error in a preceding line or in an included file can cause the cursor to be on a line with no errors. It is important to look at the lines above the highlighted error line when the error is not evident.

NOTE: If several nested If-Then statements are used without the correct number of corresponding EndIf statements, the compiler will produce an error, but the cursor may not be located where the missing EndIf statement should have been placed. The compiler only checks for correct If-Then statement structure.

**Starting to Test**

After you have successfully compiled the script file, you are ready to begin testing your scripts. You do not need to unload and restart JAWS before testing your scripts unless you have modified the default script file, Default.jss. Switch to the application and begin testing.

**Testing Tips**

There are several techniques you can use to test your scripts. They range from breaking up your script into smaller segments to including spoken messages at certain points within the script.

Testing Smaller Segments

Whenever possible, compile and test fragments of complex scripts and functions separately. This allows you to test each fragment to determine if the script fragment is working correctly. Breaking up a script or script file into smaller statement sequences allow for better testing of complex scripts or functions. Smaller statement sequences also make it easier to eliminate any logic errors.

**Speaking Messages**

Speaking messages at crucial points in your script is extremely beneficial when testing your scripts. Messages spoken by JAWS help to determine if the logic in the script is correct. It also helps you track the logic flow of the script to ensure statements are being processed in the order you intended. Although obsolete, the SayString function provides a good method for speech output. The function only requires one parameter, the string of text to be spoken. A series of SayString messages, announcing different information, is very informative when determining correct script performance. SayString messages also provide a good method of determining correct branch processing. For instance, SayString messages in each branch of an If statement will announce what branch is being executed.

After you have tested the flow of your script, it is best to remove the SayString function. The SayString function should never be used to provide output in completed scripts. The function does not give you the ability to honor verbosity settings as the SayMessage and SayFormattedMessage functions do.

**Speaking Numeric Values**

You can also speak numeric values during the execution of your scripts. You may want to monitor the changes in value of local variables to ensure they are getting the proper values. The SayInteger function provides a good method of announcing the values of integer variables. The SayInteger function also provides a method of announcing how values are changing during script execution. You may find that these variables do not have the expected values at certain points as the script executes. The SayInteger function is also useful in tracking integer variable values during the execution of a While loop.

**Saving and Restoring Values**

Failing to save and restore states is also a cause for some script errors. This refers to cursor states, screen echo states, verbosity states, etc. When you change a state during the execution of your script, be sure you restore the original state. For instance, changing screen echo to All to hear a help bubble spoken is fine. But don't leave the system set that way as it is very confusing to the user.

**Accounting for Event Functions**

Your scripts also interact with Event Functions. You must keep in mind that JAWS has event functions present in the default script file that may trigger automatically as an application is running. These event functions can interact in unexpected ways with the scripts you are writing. For example, you may place reading functions in your script to read areas of the screen at certain times, only to find that the information is being spoken twice. This indicates that one of the event functions such as the NewTextEvent function or the Focused Changed Event function is being called to speak the text automatically. Your script is then speaking the information a second time. Analyzing the script will not reveal the cause of the problem. The event function is actually speaking the information the second time. It is important you have an understanding of what the JAWS event functions do and when they are performed.

**APPENDIX A: SCRIPTING LANGUAGE KEY WORDS**

Listed below are the key words used in the JAWS scripting language. These key words are reserved for use by the scripting language and cannot be used as variable names, function names or script names.

**Const**

This key word begins a declaration section of constant values. All constant declarations follow on subsequent lines. The const key word must appear outside of any individual script or function. You can normally find constant definitions at the top of a script file or within a JAWS script header file.

**EndFunction**

This key word terminates the definition of an individual user-defined function. Each user-defined function must be terminated with EndFunction. The Script Manager automatically places this key word into your script when you use the New Script dialog to create a user-defined function.

**ElIf**

This key word offers a second way of creating a branch in an If statement structure. The ElIf key word is followed by a condition that JAWS evaluates. When the condition is found to be true, any statements following the ElIf are performed by JAWS.

**Else**

This key word is an optional part of an If-Then-EndIf statement structure. The Else key word provides for a second branch in an If statement structure. When the preceding If statement is false, then any statements following this key word are performed by JAWS.

**EndFor**

This key word terminates the For loop statement structure. Each For loop must be terminated with this key word. All statements placed between the For key word and this key word are performed by JAWS while the condition being evaluated in the For statement structure is true.

**EndIf**

This key word marks the end of an If-Then-Else-EndIf statement structure. The EndIf key word is always required to terminate any If statement structure.

**EndScript**

This key word terminates the definition of an individual script. JAWS automatically places this key word into your script file when you use the New Script dialog from within the Script Manager to create a script. Each script must be terminated by this key word.

**EndWhile**

This key word terminates the While loop statement structure. Each While loop must be terminated with this key word. All statements placed between the While key word and this key word are performed by JAWS while the condition being evaluated in the While statement structure is true.

**For**

The For key word begins the For loop statement structure. The For key word must be followed by a statement that dictates how many times the loop should execute. While the condition in the For statement structure is found to be true, the loop continues to be processed. Once the condition becomes false, the loop terminates.

**Function**

This key word designates the beginning of a user-defined function. The key word is preceded by the return type of the function. The name of the user-defined function follows this key word along with any parameters the function requires enclosed in parentheses.

**Globals**

This key word begins a declaration section of global variables. All global variable declarations follow on subsequent lines. The Globals key word must appear outside of any individual script or function. You can normally find global variable declarations at the top of a script file or within a JAWS script header file.

**Handle**

This key word begins the definition of a handle variable type. The name of the variable follows this key word as in handle hMyHandle.

**If**

This key word marks the beginning of an If-Then-Else-EndIf structure. A fully formulated statement includes: If, Then, Else, and EndIf. If statements evaluate a condition, that is part of the statement structure. When the condition being evaluated is true, then JAWS performs any statements following the If statement.

Int

This key word begins the definition of an integer variable type. The name of the variable follows this key word as in int iMyInteger.

Not

This key word reverses the question asked by an If statement structure. The key word is placed between the If key word and the condition being evaluated.

Object

This key word begins the definition of an object variable. The name of the variable follows the key word as in object oMyObject.

Return

This key word terminates execution of the function in which it appears. When you use this key word to return a value from a function, the information to be returned must follow the return key word. Control, along with any value, is then returned to the calling script or function. Since a script cannot return values, using the return key word within in a script terminates the script at the point the key word is used.

Script

This key word designates the beginning of an individual script. This key word precedes the actual name of the script. Each time you use the New Script dialog in Script Manager to create a script, this key word, along with the script name, is placed in the script file automatically.

Then

This key word follows the condition being evaluated in either an If statement or and ElIf statement.

String

This key word begins the definition of a string variable. The name of the variable follows the key word as in string sMyString.

Var

This key word begins a declaration section of local variables. All local variable declarations follow on subsequent lines. The Var key word must appear within an individual script or function. The first line of a script or function usually contains the Var key word as local variables must be declared before they can be used.

While

The While key word begins the While loop statement structure. The While key word must be followed by a conditional statement that is evaluated each time the loop is repeated. While the condition in the While statement structure is found to be true, the loop continues to be processed. Once the condition becomes false, the loop terminates.

**APPENDIX B: KEYBOARD MANAGER COMMANDS**

The following tables list all keyboard commands for use within the Keyboard Manager.

File Menu

All keystrokes listed below are only available when the left hand pane of the Keyboard Manager is active.

**Description Command**

Create new file CTRL+N

Delete file CTRL+DEL

Open default file CTRL+SHIFT+D

**Action Menu**

All keystrokes listed below are only available when the right hand pane of the Keyboard Manager is active.

**Description Command**

Add keystroke CTRL+A

Change keystroke CTRL+H

Remove keystroke DELETE

Find keystroke CTRL+F

View documentation CTRL+D

**APPENDIX C: SCRIPT MANAGER COMMANDS**

The following tables list all keyboard commands for use within the Script Manager.

**File MenuDescription Command**

New file CTRL+N

Open user file CTRL+O

Open default file CTRL+SHIFT+D

Save and compile CTRL+S

Save without compile CTRL+W

Print CTRL+P

**Edit MenuDescription Command**

Undo CTRL+Z

Cut CTRL+X

Copy selection to clipboard CTRL+C

Paste text CTRL+V

Select all CTRL+A

Select script CTRL+R

Find CTRL+F

Find next F3

Find prior SHIFT+F3

Replace CTRL+H

Incremental search ALT+CTRL+I

**Script MenuDescription Command**

New script CTRL+E

Delete script CTRL+DEL

Insert function call CTRL+I

Insert perform script CTRL+SHIFT+I

Next script F2

Prior script SHIFT+F2

Go to line CTRL+G

Script list CTRL+L

**View MenuDescription Command**

View documentation CTRL+D

**APPENDIX D: NAMING JAWS SETTINGS FILES**

Each time you start an application, JAWS determines what configuration files to load. Prior to JAWS 5.0, JAWS used the name of the executable file to retrieve and load the appropriate configuration files. JAWS uses only the first portion of the executable file name i.e. the part of the file without the extension. For example, when you started Microsoft® Word 2007, the actual executable file name is winword.exe. JAWS uses only the first part of the file, winword, to locate and load into memory all configuration files for Word from the settings folder. JAWS loads these files on top of the set of default files already in memory.

Many times, the names of those files are not indicative of the actual application. For example, the configuration manager file for Internet Explorer® 9 is Ieframe.jcf. The name of the file gives no indication of the actual program with which JAWS associates the file. This naming convention makes it difficult at times to locate the correct files for a particular application when its name is not clear.

The Confignames.ini File. JAWS 5.0 and later uses an ini style file to equate more meaningful names with the executable file name. JAWS uses the Confignames.ini file to retrieve a more understandable name for a given application. This file can be found in your JAWS shared settings folder. The file contains aliases that are associated with the hard-to-remember executable file names. For example, when you run the Settings Center from within Internet Explorer 8 or 9, the file name JAWS displays in the title bar is Internet Explorer.jcf instead of Ieframe.jcf. As you browse the JAWS settings folder using Windows® Explorer, you will see the same aliases there as you would in the title bar of a given JAWS manager as well.

Loading the Correct Files

Each time you start an application, JAWS uses the Confignames.ini file to determine if an alias exists for the application. If JAWS finds an alias in this file, then JAWS uses the alias to locate all the corresponding configuration files for the application. If no alias exists in the file, then JAWS uses the executable name of the application to locate all the corresponding configuration files. For example, when you start Internet Explorer 9, JAWS notes the applications executable file name, Ieframe.dll, less the extension. JAWS must then search the Confignames.ini file for an entry containing Ieframe. When JAWS finds the entry, the alias of Internet Explorer is noted. JAWS then looks in the settings folder for all configuration files beginning with Internet explorer. When JAWS finds all the corresponding files, they are loaded into memory.

Adding Your Own Aliases

As you customize JAWS to work with new applications, JAWS continues to use the applications executable file name to create the various configuration files. However, you can add your own entries to the Confignames.ini file to give those hard-to-remember configuration files more meaningful names. When you add a new entry to this file, you must provide two pieces of information:

application executable name les the extension

alias to be used In place of the application name. If you are unsure of the exact name of the applications executable file, press INSERT+Q and JAWS speaks the settings file currently in use along with the applications executable file name. If the applications executable file name is hard to understand, you can press INSERT+Q twice in succession to display the settings file in use information in the virtual viewer. Since you are adding an entry to the Confignames.ini file, your entry must follow a specific format. The applications executable name appears first. The executable name does not include the file extension. Following the executable name is the equals sign (=). You then type the alias for the executable file name. For example, the entry for Internet Explorer looks like the following:

ieframe=Internet Explorer

To add a new entry to the Confignames.ini file, do the following:

Open the Confignames.ini file in your favorite text editor such as Notepad or TextPad.

Press CTRL+END to move to the bottom of the file.

Add the new entry using the format of: executable name(less the extension)=alias.>

Save the file.

NOTE: You must add the alias to the Confignames.ini file before using any one of the JAWS managers from within a new application. Failure to do so causes JAWS to use the executable name for any files created before the entry is added.