Dragon NaturallySpeaking™

Creating Voice Commands

Dragon Systems®

August 1998. Version 3.0

03-021-25-01

Contents

About This Guide

Supported Software

This guide describes how to create voice commands and use the scripting language of Dragon NaturallySpeaking Deluxe 2.*x* or Dragon NaturallySpeaking Professional 3.*x*. This manual applies to Dragon NaturallySpeaking editions in all supported languages. If you have a Dragon Systems® product released after August 1998, consult your product's documentation to determine whether this document also applies to your product.

Intended Audience

This guide is for experienced Dragon NaturallySpeaking users. It assumes that you are already familiar with Microsoft® Windows® 95, Windows 98, or Windows NT®. It also assumes that you are familiar with Dragon NaturallySpeaking and any applications that you are using.

This guide does not tell you how to use the New Command and Edit Command wizards. It assumes that you are familiar with these tools. The guide can be used either with these wizards or to edit command definition (.dvc files) directly. Unless otherwise indicated, syntax and examples in this manual are in the format used in the wizards.

Document Conventions

When you see	It means
bold text	In the glossary, a word that is defined elsewhere in the glossary
italic text	Emphasis or Variable name
`constant width`	Syntax definition or Code example

Related Online Documentation

The Dragon NaturallySpeaking Help system contains reference information on the scripting language. This document, however, includes significant additional information and corrections not available at the time the online documentation was created.

The Dragon NaturallySpeaking SDK Help system, part of the Dragon NaturallySpeaking Developer Suite, provides information on using scripting language commands in application programs.

Chapter 1 Introduction to Voice Commands

This chapter provides a description of voice commands and their elements. It includes the following topics:

- Voice commands
- Scripting language

Voice Commands

A Dragon NaturallySpeaking voice command is a word or phrase that causes Dragon NaturallySpeaking to take some action other than type the text you just spoke. Voice commands that you create are sometimes called macros. Each voice command has three elements:

- name
- action
- scope

Voice Command Name

The voice command name is the word or phrase that you say to run the voice command, such as:

What Can I Say

You say the phrase “What Can I Say” to run the voice command.

A voice command name can include the names of one or more lists of words. Dragon NaturallySpeaking recognizes any command that matches the words in the command name and a word (or phrase) from each list. For example, when you use the command:

Set Font <FontFace>

you say the words “Set Font” followed by a word from the FontFace list, such as “Arial”. The full spoken command can be “Set Font Arial”, or “Set Font Times New Roman”, and so on.

Voice command names should consist of two or more words. This makes for a more natural command set and limits confusion with dictation words.

Voice Command Action

The voice command must tell Dragon NaturallySpeaking what to do. This command action can be either of the following:

- A set of keystrokes that performs a specific action, called a keystroke command. You can write a keystroke command as a shorthand for a commonly used phrase such as a return address or a salutation. For example, you can create a Dictated With Dragon voice command that types “This letter was written entirely by speech using Dragon NaturallySpeaking™.”

You can also use a keystroke command to control an application. For example, the global Copy All to Clipboard command is a keystroke command that generates the Ctrl+a and Ctrl+c key combinations, which select all the text in an application and then copy the selection to the clipboard.

Each keystroke command can send only a single, fixed, keystroke sequence.

- A script consisting of one or more commands in the Dragon NaturallySpeaking scripting language. In this case, the voice command is called a script command. Script commands provide significantly more power and flexibility than keystroke commands. The scripting language enables a script command to perform operations that keystrokes cannot do, such as putting Dragon NaturallySpeaking to sleep or activating a message box. Script commands can use lists. They can also use programming constructs such as variables and conditional processing. This book focuses on using the scripting language.

Voice Command Scope

A voice command can be either global or application-specific. If it is global, Dragon NaturallySpeaking recognizes it in all applications. If it is application-specific, Dragon NaturallySpeaking only recognizes it when a specific window or dialog box is active in a specific application. For example, you can create a “Full Date” command that is only recognized in the WordPad Date and Time dialog box.

Simple Command Examples

The following keystroke sequence inserts the text “Today’s date is: ” followed by the current date at the insertion point location in WordPad:

Today’s date is: {Alt+i}d{Down 7}{Enter}

That is, it sends:

- The character keystrokes “Today’s date is: ”
- The Alt+i key combination to select the WordPad Insert menu
- d to select the date and time option
- Seven down-arrows to select the desired format from the date and time control box
- Enter to complete the operation and close the dialog box

The following voice command script has an action that consists of two scripting language commands:

```
MsgBoxConfirm "Take a break now?", 35, "Break Time"
GoToSleep
```

The first command displays a message box with the message “Take a break now?” and Yes and No buttons. If you press the No button, the script exits. If you press the Yes button, the script runs the GotoSleep command and puts Dragon NaturallySpeaking in sleep mode.

Scripting Language

The Dragon NaturallySpeaking scripting language is the programming language for Dragon NaturallySpeaking voice commands. It provides a set of commands that Dragon NaturallySpeaking can interpret and execute. It also provides a set of standard program elements such as data manipulation functions and flow control statements.

A script is a sequence of scripting commands and program elements. A script automates a task that controls Dragon NaturallySpeaking and Windows. Many Dragon NaturallySpeaking commands run scripts.

The scripting language supports the following elements:

- Expressions
- Variables
- Operators
- Data manipulation functions
- Flow control statements
- Scripting commands

The scripting language expressions, variables, operators, data manipulation functions, and flow control statements are modeled after the BASIC computer language. They provide the programming language framework for writing powerful voice command scripts. See chapter 5 for detailed descriptions of all programming language elements.

The scripting commands let you control Dragon NaturallySpeaking and use Dragon NaturallySpeaking to control applications. The remaining scripting language elements, such as variables and flow control statements, affect script execution.

Scripting Command Overview

Dragon NaturallySpeaking scripting commands provide you with a powerful set of tools for controlling Dragon NaturallySpeaking, the Windows environment, and your applications. With them, a voice command script can:

- Launch and control applications on the desktop, including changing the active application.
- Control applications by using DDE commands and DLL calls, sending keystrokes to an application, selecting a window control, and selecting a menu item.
- Control Dragon NaturallySpeaking by putting it into and out of sleep mode and turning the microphone on and off.
- Control script execution by displaying a confirmation box, running another voice command script, and making the script pause.
- Control the mouse, including clicking the mouse buttons, dragging the mouse, controlling mouse motion, and controlling the MouseGrid™.
- Tell Dragon NaturallySpeaking to behave as if it recognized a specific word or set of words.
- Play beeps and .wav files.
- Control dictated text by sending keystrokes to the active window and "pressing" modifier keys such as Shift, Alt, and Ctrl.

You can build powerful voice commands by combining these operations. For example, you can write a single voice command to copy text from one application into the clipboard, bring up another application, and paste the text into the application.

Chapter 2 describes the fundamentals of voice commands, including scripting language rules and conventions. Chapter 3 gives more information on using scripting commands, and chapter 5 describes each scripting command in detail.

Creating Voice Commands

There are two ways to create a voice command:

- Use the Dragon NaturallySpeaking New Command wizard and Edit Command wizard. The Dragon NaturallySpeaking online Help documents how to use these wizards. Unless otherwise indicated, syntax and examples in this manual are in the format used by these wizards.
- Use a text editor or word processor to edit the Global.dvc file. The Global.dvc file defines all voice commands; chapter 4 describes the .dvc file structure and syntax in detail.

Most Dragon NaturallySpeaking built-in commands are voice commands, and many of them run scripts. You may want to look at these commands in either the Edit Command wizard or the Global.dvc file to better understand scripts and scripting commands.

Note

If running a script causes an error, Dragon NaturallySpeaking displays an error message box and the script quits. You must fix the programming error before you can run the script.

Chapter 2 Voice Command Fundamentals

This chapter describes some of the fundamentals of voice commands. It includes the following topics:

- Command scope and how it affects the structure of your command set
- Basic voice command elements and rules
- Voice command limitations, where the commands are stored, and information on deploying the commands you create.

Command Scope

The following sections describe how Dragon NaturallySpeaking determines which voice commands it can recognize and how you can control a command's scope. It also describes what makes an effective global command.

Application-Specific Commands

Dragon NaturallySpeaking uses the active application and window to determine the valid application-specific commands.

When you create an application-specific command you specify the application to which it applies. Dragon NaturallySpeaking only recognizes the command when the application is active. (The application corresponds to a menu in the Global.dvc file that contains the command. See chapter 4 for more information on the Global.dvc file structure.)

You must also specify the application-specific command's target window or dialog box title. The command will only work in windows or dialog boxes whose title bars include the target title text. Target window titles can contain up to 32 characters and are case sensitive. The target title must uniquely identify the window and can consist of any part of the text that appears in the window title bar. For example, if the window title bar includes the words "Turbo TextEdIt, you could use "Turbo TextEd" as the window title. (The target title corresponds to a state in the Global.dvc file.)

If you want a command to work whenever the application window is active, use the application window title, for example, “WordPad”.

If you want the command to work only when a particular document is active, use the document name, for example, “WordPad - Report.txt” or “Report.txt”.

If you want the command to work only when a particular dialog box is active, select the dialog box title, for example, “Font”. See page 36 for more examples of using window names.

Global Commands

Global commands are almost always active. They are not active in sleep mode, and you can tell Dragon NaturallySpeaking to deactivate them in specific application windows if you edit the Global.dvc file directly (see chapter 4 for more details).

Because they are nearly always active, you should not make a voice command global unless you must have the command active at all times. Also, your command’s action should have the same result everywhere. For example, do not create a global command that sends a set of keystrokes that may have different effects in different applications.

In most cases, you only need to use your command at specific times, such as when a particular application or window is active. In these cases, create an application-specific command. By doing so, you ensure that the command gets recognized only at the right times. You also save processing time because Dragon NaturallySpeaking does not waste time determining whether speech matches the voice command name each time it recognizes a word.

Basic Voice Command Elements

Before you can write a voice command you must understand basic elements of the commands, including the rules for voice command names, how to specify words with different written and spoken forms, how to specify lists and list contents, how to specify keystrokes in voice commands, and the elements of a voice command script.

Guidelines for Creating and Naming Commands

You can use any word or phrase as a voice command name. You should keep the following guidelines in mind whenever you create commands:

- Create commands sparingly. In particular, create global commands only if they must be recognized in all applications.
- Enclose each list name in angle brackets (< >). For example, in the command Format That <FontFace>, <FontFace> is the name of a list.
- Use a small set of easily distinguished words at the start of the command(s). For example, Change Color To <Color> instead of Red Text, Blue Text, Green Text, etc.
- Make your commands imperative, for example, Move <Message> To Outbox. Doing this improves the likelihood of Dragon NaturallySpeaking recognizing the phrase as a command.
- Use a descriptive phrase as the voice command name. This helps you remember what function it performs. For example, use Type My Address for a voice command that types your address.
- Do not make a voice command name so long that it is hard for you to say or remember. However, Dragon NaturallySpeaking usually recognizes longer names more easily, so the name should not be too short. For example, you should not have a Convert The Last Things I Said From Standard to Outline Format command. You should also not have a Fix It command.
- Do not include punctuation in voice command names. If you include them, you will have to say the punctuation characters.
- If you use a word combination frequently in dictation, do not use it as a command name. This will limit the likelihood that you will accidentally run a command when you intend to dictate, or vice-versa. (If you say the combination as a complete utterance, that is, with a pause at the beginning and end but no pause in the middle, Dragon NaturallySpeaking interprets it as a command. If you say the combination as part of a larger utterance, Dragon NaturallySpeaking interprets it as dictation.)
- As a general rule, you should follow the Dragon NaturallySpeaking convention of capitalizing all words in the command name. This ensures that command names are readily distinguished from dictation words in the results box.
- All global voice commands must have different names. Each voice command name must be unique for a particular window, but you can use the same name for multiple application-specific commands for different applications and windows. For example, if you have two applications that use different keystroke combinations to copy a

table, you can create a separate Copy Table command for each application that sends the keystrokes it requires.

- You can have a global command that has the same name as a window-specific command. Dragon NaturallySpeaking will recognize the application-specific command when its window is active and will recognize the global command at all other times.
- Do not create voice command names for a window that differ only in their capitalization. You cannot be sure which voice command Dragon NaturallySpeaking will recognize.
- Do not create commands with the same name in different states that are active at the same time. It is unpredictable which takes precedence. (You cannot do this if you are using the command wizards.)

Lists

Each application and window can have one or more lists you can use in creating commands for the window. Each list has a name, such as FontFace, and consists of a number of entries that can be recognized interchangeably in a command. Each entry can be a single word or a phrase and must be on a separate line. Lists enable a single voice command to handle multiple utterances, with results that can vary based on the word recognized from the list.

For example, the FontFace list for the Dragon NaturallySpeaking window contains the following entries:

“Times”
“Times New Roman”
“Courier”
“Courier New”
“Arial”

Dragon NaturallySpeaking uses this list in many commands, such as Set Font <FontFace> and Set Font <FontFace> <FontSize>. If you say “Set Font Times New Roman” the Set Font <FontFace> command changes the current font to Times New Roman; if you say “Set Font Arial” the same command sets the font to Arial. The following scripting command does this work:

```
SendKeys "{Alt+o}f{Alt+f}" + _arg1 + "{Enter}"
```

In the script, _arg1 represents the word recognized from the list. The scripting command opens the Dragon NaturallySpeaking word processor, selects the font name text box, enters the font name, and then sends an Enter keystroke to complete the operation.

Guidelines for Lists

Keep the following guidelines in mind whenever you create lists:

- Each (nonglobal) list is window-specific and is not available to commands for other windows. However, you can define lists with the same names for multiple windows, and each list can contain either the same or different words.
- List names can include alphabetic characters, numbers, spaces, hyphens, apostrophes, and underscores (_). You cannot include any of the following characters: * + = | " [ ] { } < >. As a general rule it is best to limit the name to numbers and letters. The name should be easy to remember.
- The list name, including the angle brackets, counts toward the maximum length of a command name (127 characters). Keeping list names below 20 characters is a good rule.
- The list name is never spoken or recognized, only the words in the list. Therefore, you can use words in the list name that are not in the Dragon NaturallySpeaking backup dictionary.
- Each list entry can consist of one or more words (a phrase).
- Lists cannot contain other lists (that is, lists cannot be nested).
- Do not use a list’s name as one of its items.
- The case of list items is usually not important for recognition. However, if you capitalize names of commands (which is recommended), you should also capitalize list items for consistency.
- List items can be in any order.
- Do not include punctuation in list entries. If you include them, you will have to say the punctuation characters.

Written and Spoken Forms of Words

You can specify both the written and spoken forms of words in lists (and in the argument to the HeardWord scripting command). To do so in the New Command and Edit Command wizards use the form:

written-form\spoken-form

For example, all words in the ICAlphabet list that defines the letters in the International Communications Alphabet are of the form “a\alpha”.

The Dragon NaturallySpeaking dictation commands, such as New-Paragraph, which you can say without pause during dictation, are actually dictation words with no written form. Therefore, in a command script, you must precede them with a backslash.

In the Global.dvc file, use a double backslash (\\) instead of a single backslash between the two forms. This format is required because the backslash is the .dvc file format escape character.

Key Names in Voice Commands

Use the following syntax to specify key names in keystroke commands and command scripts:

- Use printable keys exactly as they appear when typed. For example, use a, A, &, and so on.
- Enclose the names of nonprintable keys and key combinations in braces ({}).
- Use {{} to get an open brace character in cases where it could be confused with the beginning of a key name. For example, use {{}Shift} to send the characters “{Shift}” instead of pressing the Shift key.
- Use the names that appear on most keyboards for special key names. For example, use{Enter}, {Esc}, {F3}, {BackSpace}, {PgUp}, {Tab}, and so on. If you can get a function (such as Home and PgDn) by pressing a key on the extended keypad or on the numeric keypad, use the name that appears on the numeric keypad. The Key Name List starting on page 116 lists all special key names.
- Use uppercase and lowercase if you wish, since special key names are not case sensitive. For example, you can use {esc}, {Esc}, or {ESC}.
- Identify a key if there is more than one key with the same name. Use the “Right” prefix to refer to Shift keys that are to the right of the spacebar. The “Ext” prefix means extended and refers to the keys that are on the extended keypad between the main section of the keyboard and the numeric keypad. For example, {RightShift} specifies the Shift key to the right of the keyboard, and {ExtInsert} specifies the Insert key between the main keyboard and the numeric keypad.
- Spell the names of special keys that do not have names on them, such as {Space}, {Up}, {Left}, {NumKey/}, and {NumKey+}.
- Use the modifier key name, a plus sign (+), and another key to combine the Shift, Control, and Alt modifier keys with other keys. For example, use {Shift+Enter}, {Ctrl+Left}, {Ctrl+Shift+@}, and

{RightAlt+a}. You can also generate uppercase letters with the Shift key. For example, A and {Shift+a}are the same.

- Generate multiple keystrokes by following a key (or key combination) name with a space and the number of keystrokes. For example, {Right 5} generates five right arrow keystrokes and {@ 5} generates "@@@@@".
- Combine the Alt key with numeric keys in one operation to generate symbolic characters. Use the Windows Character Map tool to find the keystrokes for all characters in a font. For example, use {Alt+0174} to generate the ® registered trademark symbol in standard fonts.

Scripting Language Rules and Conventions

The scripting language follows the standard conventions of the BASIC programming language:

- Use uppercase and lowercase, if you wish, since scripting language commands are not case sensitive.
- Place arguments after the command name.
- Put a space between the command name and the first argument.
- Separate arguments with a comma (spaces are optional).
- Surround literal string arguments with quotation marks.
- Use two quotation marks ("") to include a quotation mark in a string argument. (This only applies in scripts. Use \" elsewhere in .dvc files.)
- Do not surround numeric literal arguments and variable names with quotation marks.
- Do not start variable names with a digit.
- Use any combination of alphanumeric characters (A–Z, a–z, 0–9) and underscores (_) within the variable name.
- Put multiple commands on a single line by putting a colon (:) between commands.
- Start a comment with an apostrophe ('). The apostrophe must either be the first non-whitespace character on a line or follow a colon command separator. Dragon NaturallySpeaking ignores all text until the beginning of the next line. (This comment character only applies to text in scripts. Use the # character elsewhere in .dvc files, as shown on page 42.)

Scripting Convention Examples

The following command opens a Windows message box. There are two string arguments and one integer argument.

```
MsgBoxConfirm "Exit database?", 36, "Warning"
```

The following command opens the Microsoft Excel file AMORT.XLS. The DdeExecute scripting language command has three string arguments. The third argument is `[open("amort.xls")]`. There are two quotation marks before and after `amort.xls` to include the quotes in the argument.

```
DdeExecute "excel", "system", "[open(""amort.xls"")]"
```

Developing and Deploying Voice Commands

You can use the command wizards or a text editor to develop your voice commands. In either case, you must understand the following information to effectively develop and deploy your commands: system limitations on the commands, where Dragon NaturallySpeaking keeps voice commands, and how you can distribute the commands.

Voice Command Limitations

Dragon NaturallySpeaking supports a maximum of 3,000 commands per window, in addition to a maximum of 3,000 global commands. As a result, up to 6,000 commands can be active at any time.

Each command action is limited to 16,000 characters.

Where Commands are Stored

Dragon NaturallySpeaking keeps voice commands in a file named Global.dvc. It keeps a master Global.dvc file in the *NatSpeak*\Data directory. It uses this copy for all users *until* they use a command wizard to create or modify a command. You cannot use the wizards to edit this copy of the file.

When a user first creates or edits a command with a command wizard, Dragon NaturallySpeaking copies the master Global.dvc file to *NatSpeak*\Users*userName*\Current\Global.dvc. From then on, Dragon NaturallySpeaking uses this file for the user’s commands.

Dragon NaturallySpeaking always looks for a user-specific Global.dvc when it loads a user. If it does not find one, it loads the master Global.dvc.

You can copy new commands from one Global.dvc file to another. If you do so, you must make sure that the file you are copying into is not currently being used by Dragon NaturallySpeaking, which could overwrite your changes. You should also back up any file before you modify it.

Distributing Voice Commands

Use the following guidelines when distributing commands to users:

- Users must have Dragon NaturallySpeaking Deluxe or Professional edition.
- You can copy and paste sections of one Global.dvc file into another Global.dvc file. However, you must make sure that the resulting file structure is valid, as described in chapter 4.
- If you replace a Global.dvc file, you must make sure the replacement file is complete and has all the necessary commands for the user. The contents of the Global.dvc file that Dragon NaturallySpeaking installs on a system varies, depending on whether the user installs NaturalWord for Microsoft Word, Corel® WordPerfect®, or both.
- If a user has created his or her own commands, add your commands to the user’s current Global.dvc file, *NatSpeak*\Users\userName\Current\Global.dvc. Make sure not to unintentionally delete existing user-specific commands.
- If a user has not created any commands, you can place your Global.dvc file in the *NatSpeak*\Users\userName\Current\ directory.
- If you want newly created users to have your commands, replace the Global.dvc file in \NatSpeak\Data with yours or copy your commands into the existing file. *Always* save a clean copy of the master Global.dvc file before you modify or replace it.

Chapter 3 Writing Effective Voice Commands

This chapter describes how to write effective voice commands. It also provides information on using Dynamic Data Exchange (DDE) with Dragon NaturallySpeaking. This chapter includes the following topics:

- Interacting with Windows applications
- Controlling Dragon NaturallySpeaking
- Using voice command programming techniques
- Writing commands with lists
- Using Dynamic Data Exchange with Dragon NaturallySpeaking

Interacting with Windows and Applications

Windows lets you interact with applications by:

- Using the keyboard
- Using the mouse
- Selecting menus, menu items, buttons, and other controls directly
- Using DDE commands
- Calling functions in DLL files

Dragon NaturallySpeaking lets you use all these methods of interacting with Windows applications in your voice commands. In many cases, you can use any of the several methods to interact with Windows. In others, only some may be available.

The rest of this section describes the various ways Dragon NaturallySpeaking uses these methods to interact with Windows and applications. The reference pages provide more details on each scripting command, including any specific command limitations.

Sending Keystrokes

Keystroke commands can generate any possible keystroke or keystroke combination, so you can create keyboard voice commands to move around in windows and control Windows applications. Many Dragon NaturallySpeaking speech commands are simply keystroke commands.

For example, the global Undo That command is a keystroke command that types the Ctrl+z key combination.

You can also incorporate key combinations in script commands by using the SendKeys or SendSystemKeys scripting commands. Many Dragon NaturallySpeaking commands use the SendKeys command to send keystroke combinations. For example, the Set Font <FontFace> uses the following scripting command:

```
SendKeys "{Alt+o}f{Alt+f}" + _arg1 + "{Enter}"
```

The following scripting commands send text to an application:

Command	Description
SendKeys	Sends keystrokes to the active window.
SendSystemKeys	Sends system keystrokes to Windows. Use this command for short system keystrokes, such as {Ctrl+Esc}, when SendKeys does not work.

Note

If a command includes system keystrokes, such as {Alt+Tab} or {Ctrl+Esc}, do not use a keystroke command. Instead, create a script that uses a SendSystemKeys scripting command. Keystroke commands work only for nonsystem keys.

Controlling the Mouse

The following scripting commands control mouse motion.

Command	Description
SetMousePosition	Places the mouse pointer at a specified position.
ButtonClick	Clicks the specified mouse button.
RememberPoint	Records current mouse pointer position. Use this command to specify the starting location of a DragToPoint command.
DragToPoint	Drags the mouse from the point specified by the RememberPoint command to the current pointer location.

Command	Description
SetMousePosition	Places the mouse pointer at a specified position.
MouseGrid	Uses the Dragon NaturallySpeaking MouseGrid to position the mouse pointer.

Accessing and Selecting Windows, Menus, and Controls

The scripting language lets you access and select windows, menus, and controls in several ways:

- You can send shortcut keystrokes. For example, the global Copy All to Clipboard command is a keystroke command that sends the {Ctrl+a}{Ctrl+c} key combination to select the entire document and copy it to the clipboard. However, global commands that use shortcut keys may have unexpected results in applications that do not follow standard Windows shortcut key conventions.
- You can send {Tab} keystrokes to tab through controls such as text fields and buttons. This technique is useful for accessing controls that do not have shortcut (accelerator) keys. You can also send the {Space} or {Enter} keystroke to activate a control.
- If the application can function as a DDE server, you can use the DdeExecute and DdePoke commands to send Windows DDE requests directly to an application. You may prefer this method to using keystrokes with applications, such as Microsoft Word, where you can change key assignments.
- You can use the following scripting commands to interact directly with the application:

Command	Description
ControlPick	Selects a control (a child window) of the current window. Push buttons, check boxes, edit fields, and radio buttons are types of controls.
MenuPick	Selects a currently visible menu or menu item.
MenuCancel	Cancels the current menu.

Controlling Sounds

The following scripting commands produce sounds:

Command	Description
Beep	Plays the default Windows beep sound.
PlaySound	Plays a .wav file.

Starting, Selecting, and Controlling Other Applications

Dragon NaturallySpeaking provides a number of scripting commands to start applications, change the active application, and minimize applications. It also provides commands to send Dynamic Data Exchange (DDE) commands to other applications and call functions in Dynamic Link Libraries (DLLs). These commands are as follows:

Command	Description
AppBringUp	Starts an application or switches to an instance of the application that is already running.
AppSwapWith	Swaps the current application with another.
ShellExecute	Loads an application. This command always starts a new instance of an application.
ClearDesktop	Minimizes all applications.
DdeExecute	Sends a DDE request to another application.
DdePoke	Sets an item value in a DDE application.
DllCall	Calls a function in a DLL.

Controlling Dragon NaturallySpeaking

Dragon NaturallySpeaking provides you with a number of scripting commands that let you control how it works. These commands let you:

- Turn Sleep mode on and off
- Turn the microphone on and off
- Use the text-to-speech engine to play text
- Control how scripts execute
- Tell Dragon NaturallySpeaking to recognize a word

Controlling Sleep Mode

In sleep mode Dragon NaturallySpeaking normally recognizes only the wake-up command. The following scripting commands let you turn sleep mode on and off. (For more information on sleep mode, see page 37.)

Command	Description
GoToSleep	Puts Dragon NaturallySpeaking in sleep mode.
WakeUp	Resumes normal recognition when Dragon NaturallySpeaking is in sleep mode.

Controlling the Microphone

The SetMicrophone command lets you turn the microphone on and off. This command lets you give other applications or commands access to the I/O channel on systems with half-duplex sound cards. For example, you should turn the microphone off before using the Beep command to send a Windows default beep tone.

Controlling Text-to-Speech

The TTSPlayString command tells the Dragon NaturallySpeaking text-to-speech engine to play text. The text-to-speech engine comes with Dragon NaturallySpeaking Deluxe and Professional editions.

Controlling Script Execution

The following scripting commands control Dragon NaturallySpeaking voice command scripts:

Command	Description
MsgBoxConfirm	Displays a Windows Message box. This box displays a message and one or two buttons. The script then continues or exits, depending on which button you select. Use this command to make a user confirm an action before it happens.
RunScriptFile	Executes a specified script. This lets you run one script from another script and reuse

Command	Description
	common script code.
Wait	Makes the script pause for a specific amount of time. This command is useful when one action must finish before the next starts. For example, use the Wait command between an AppBringUp command that runs a slow-starting application and a command that sends keystrokes to that application.

Simulating Recognition

The HeardWord scripting command causes Dragon NaturallySpeaking to act as if it recognized a specific word or words.

The HeardWord command lets you create aliases for commands without rewriting the entire voice command. For example, you could create a Fix That voice command that has the following script:

```
HeardWord "Correct", "That"
```

Whenever you say Fix That, Dragon NaturallySpeaking automatically acts as if it heard “Correct That” and brings up the Correction dialog box.

HeardWord is also very useful in Commands that use lists, as described in “Using HeardWord in Commands with Lists” on page 30.

Voice Command Programming Techniques

The following sections present information on string concatenation, user defined variables, and flow control. These programming techniques are particularly useful in writing advanced voice commands such as commands that use lists (see page 26).

Using String Concatenation

The string concatenation operator (+) lets you combine strings. Use it to combine variables with literal text to create a single argument string.

For example, the Set Font <FontFace> command script for the Dragon NaturallySpeaking word processor includes the following scripting command with a string concatenation expression:

```
SendKeys "{Alt+o}f{Alt+f}"+arg_1+"{Enter}"
```

The expression is made of:

- The string literal “{Alt+o}f{Alt+f}”
- The + concatenation operator
- The variable _arg1, which represents the string value of the FontFace list item
- The + concatenation operator
- The string literal “{Enter}”

Note that the two keystroke string literals are in quotation marks, but the variable name is not. Also, the plus signs inside the first literal are part of the literal and are *not* concatenation operators.

If you say “Set Font Arial,” the SendKeys command gets the following string as its argument:

```
{Alt+o}f{Alt+f}Arial{Enter}
```

When you use string concatenation, make sure that the variables are strings, not numeric values. If the variables are all numbers, Dragon NaturallySpeaking interprets the plus (+) sign as the addition operator and adds the values. If some variables are strings and some are numbers, Dragon NaturallySpeaking displays an error message. If necessary, use the Str$ function to convert numerical values to their string representations.

User-Defined Variables

You can define your own variables. You can use your variables to store temporary values including indexes for looping, conditionals, and substrings that you generate with data manipulation functions. For example, Dragon NaturallySpeaking commands often use statements such as:

```
if _arg1 = "Cut" then key$ = "{Ctrl+x}"
if _arg1 = "Copy" then key$ = "{Ctrl+c}"
```

The variable key$ depends on the word recognized from a list, represented by _arg1. (Dragon NaturallySpeaking automatically generates variables for lists.) The command script uses the variable in a SendKeys command, which could be as simple as:

```
SendKeys key$
```

All variables are local to the current instance of the voice command in which they appear. If you use the same variable name in two voice commands, they are two different variables. Similarly, there is no way to retain a variable value after a voice command completes.

Flow Control

The Dragon NaturallySpeaking built-in commands use IF THEN and IF THEN ELSE statements to control script execution. For example, these conditionals can help a script handle motion in different directions. Similarly, the Dragon NaturallySpeaking built-in commands use WHILE and LOOP WHILE loops to repeat an operation several times.

The following code uses a DO UNTIL loop to type the message “Hello World.” ten times:

```
I = 10
DO UNTIL I = 0
SendKeys "Hello World.{Enter}"
I = I-1
LOOP
```

The examples of flow control statements in chapter 4 show their use in greater detail.

Caution

You can create an infinite loop where no condition ever causes the loop to stop. If you run a command with an infinite loop, you may have to use the Windows Close Program dialog to shut down Dragon NaturallySpeaking.

Writing Commands with Lists

The following sections contain information to help you write commands that use lists.

List Command Names

When a command uses lists, the command name must follow the naming conventions described in the “Guidelines for Creating and Naming Commands” section on page 8. In addition:

- The lists must be defined for the current application and window or be global lists if this is a global command. (If you are using the New Command wizard to create a command, you do not have to create the list until after you have entered the command script.)

- Angle brackets (< >) must surround each list name in the command name.

The list names identify the list from which Dragon NaturallySpeaking recognizes the variable part of the command. You can only say words in the specified list for the variable part of the voice command.

For example, the voice command Set Font <FontFace> consists of the words Set and Font and the list name FontFace. You must say “Set Font” followed by a valid font name from the FontFace list.

List Command Execution

When you say a word that corresponds to the variable part of a voice command, Dragon NaturallySpeaking recognizes the word or phrase from the list and sets the variable that corresponds to the list to the recognized word or phrase. This variable is only active when the command script runs.

When Dragon NaturallySpeaking finishes recognizing all the words in the command, it executes its voice command script using the values that it assigned to the list variables in the script.

List Command Scripts

Voice commands that use lists almost always have scripts because keystroke commands cannot use variables. (You can use a list for a keystroke command if you want the user to be able to say several different things to get the same result.) The specific script action depends upon the values returned by Dragon NaturallySpeaking when it recognizes each variable word.

While a script can be a single statement, many scripts that control applications can be complex. Conditional looping, data manipulation functions, and operations on variables provide the power needed to write effective command scripts.

List Variables in Scripts

When you write a script for a command that uses lists, you use variables to represent the words that Dragon NaturallySpeaking recognizes from the lists. Dragon NaturallySpeaking automatically creates the names for these variables.

The variable names are all of the form _arg*n*, where *n* is the position of the corresponding list in the command name. Therefore, if the command (from the Word 97 commands) is:

Print Pages <1To100> <toThrough> <1To100>

and you say “Print pages 5 through 15”

_arg1 represents the first word recognized from the <1To100> list, in this case “5”.

_arg2 represents the word “through” recognized from the toThrough list.

_arg3 represents the second word recognized from the <1To100> list, in this case “15”.

A Simple Command Using Lists

In the voice command Move <DirUpDown> <2To20> Lines, assume you said “Move Back 3 lines”. This causes the following sequence of actions:

1. The _arg1 variable gets the value “Back”, recognized from the DirUpDown list.
2. The _arg2 variable gets the string value “3”, recognized from the 2To20 list.
3. The script for the Command action runs. In this case,:

```
if _arg1 = "Back"     then _arg1 = "Up"
   if _arg1 = "Forward"  then _arg1 = "Down"
   SendKeys "{" + _arg1 + " " + _arg2 + "}"
```

This script first changes the value of _arg1 from “Back” to “Up”, and then executes the command SendKeys “{Up 3}”.

4. The command sends three up-arrow keystrokes.

This sample script also shows the use of variable substitution and the string concatenation operator (+) to combine the string literal parts of the argument with the variable name parts of the argument. (“Using String Concatenation” on page 24 has more information on the + operator.)

Using Written and Spoken Forms of Words in Lists

Lists can contain words whose written and spoken forms differ. For example, the ICAlphabet list contains the International Communications Alphabet in the form:

a\alpha
b\bravo

and so on, where “a” is the written form and “alpha “is the spoken form.

When Dragon NaturallySpeaking recognizes the spoken form of a list word in a command, the command variable gets the entire written\spoken form string, *not* just the written form. Therefore, the script must extract the written form from the string. For example, the <PressKey> <ICAlphabet> command lets users say “Press alpha” to enter the letter “a” or “Type zulu” to get the letter “z”. The command script is:

```
SendSystemKeys MID$(_arg2,0,1)
```

This command extracts the first character from the string in _arg2 (for example the “a” in “a\alpha” and sends it to the system.

Your list may contain words with variable-length written parts, such as a list of presidents who can be named by speaking their initials:

```
Franklin Delano Roosevelt\FDR
John Fitzgerald Kennedy\JFK
Lyndon Baines Johnson\LBJ
```

In this case you can use the Instr and Mid$ functions as follows to locate the backslash character and extract the written form:

```
slash_posn = Instr (_arg1, "\")
word_val$ = Mid$ (_arg1, 0, slash_posn -1)
```

The “written form” part of a list word can contain any combination of characters that is meaningful to your script. For example, the written form of the list word can contain the keystrokes to be used in a SendKeys command.

You can use this technique to eliminate the need for multiple IF THEN statements. For example, you could rewrite the <PressKey> <FunctionKey> command so that the <FunctionKey> list has the form:

{F1}\Function-1

and the command script is

slash_posn = Instr (_arg2, "\")
word_val$ = Mid$ (_arg1, 0, slash_posn -1)
SendSystemKeys word_val$

(If you do this as an exercise, remember that you will also have to rewrite the <PressKey> <ShiftKey> <FunctionKey> command, which also uses the FunctionKey list.)

Using HeardWord in Commands with Lists

You can use the HeardWord scripting command in your script to process part of the input of your voice command. This enables you to build a set of commands that use common parts.

For example, the script for the MouseGrid command Mouse <Direction> <1To10> <MouseAction> lets you do any of several actions that require positioning the mouse when the MouseGrid is displayed. For example, it lets you move the mouse and then click mouse buttons, or it marks the location (for a DragToPoint command). The command script is:

```
HeardWord _arg1, _arg2
Wait 10
HeardWord "Mouse", _arg3
```

This command:

1. First “calls” the <Direction> <1To10> command to move the mouse to the desired location.
2. Waits 10 milliseconds (to make sure the mouse gets to the location).
3. Calls the Mouse <MouseAction> command, which does the requested action, such as clicking a mouse button.

This script is efficient because it reuses existing code and eliminates the need to provide duplicate functions in the vocabulary. If you are writing a complex set of voice commands, determine whether you can use any of the commands as modular building blocks for others.

You can nest commands that use lists by using HeardWord to call another command that uses lists. However, all variables are local to the script in which they appear, so the called script cannot use the calling script’s variables.

Using Lists to Construct Natural Commands

You can use lists of similar words or phrases to give users a variety of equivalent ways to say a command, providing a more natural grammar for your commands. Such commands often have scripts that do not even use the list variable. For example, the “Press Key” commands, such as

<PressKey> <Numeral>, do not use their first argument in their scripts. The PressKey list contains:

Press
Press Key
Keystroke
Type

The <PressKey> <Numeral> command script is:

```
SendSystemKeys _arg2
```

The script never uses the value of _arg1, the word recognized from the PressKey list. Users can start a command with any word in the Press Key list and get the same result. Thus, the command types “3” whether you say “Press 3”, “Type 3”, or “Keystroke 3”.

Using Dynamic Data Exchange

The DdeExecute and DdePoke commands send Windows Dynamic Data Exchange (DDE) messages to communicate with and control other applications. The following sections briefly describe these messages and discuss issues that are specific to their use in Dragon NaturallySpeaking scripts.

Your application’s technical documentation should provide information on its supported DDE messages, including the required values and formats for the DdeExecute and DdePoke command arguments.

DdeExecute Command

The scripting language DdeExecute command sends a command string to a DDE application. The Dragon NaturallySpeaking vocabularies often use this command to communicate with applications whose accelerator key assignments are not fixed, such as most word processors. The following scripting command will change the font size in Microsoft Word (6.0 through 97):

```
DdeExecute "WinWord", "System", "[ShrinkFont]"
```

The Dragon NaturallySpeaking commands for Microsoft Word 97 (the NaturalWord™ commands) use the DdeExecute command to run custom macros. For example, the Select That command for Word 97 uses the following script:

```
DdeExecute "WinWord", "System", "[ToolsMacro
.Name=""WCSelectThat"", .Run]"
```

Note

The DdeExecute command can run synchronously (the default) or asynchronously. It can be helpful to first use synchronous calls when you develop your commands and to convert to asynchronous calls (which return faster) if it is appropriate. See the DdeExecute reference on page 84 for more information.

DdeExecute versus SendKeys

While the SendKeys scripting command is often simpler to use than DdeExecute, there are good reasons to use DdeExecute.

Use DdeExecute in place of a SendKeys scripting command if you cannot be sure that a key or keystroke sequence has predictable results. Many applications let users redefine keys. Also, in some applications the same keystrokes produce different results in different situations. For example, when Microsoft Word 97 displays the Formatting toolbar, the {Ctrl+Shift+F} key combination puts the focus on the font name text box in the toolbar. When Word does not display the Formatting toolbar, {Ctrl+Shift+F} makes it display the Fonts dialog box.

Also, you may be able to use a simple DdeExecute command instead of a complex or confusing key sequence. For example:

```
DdeExecute "WinWord", "System", "[Grow Font]"
```

is easier to understand than:

```
SendKeys "{Ctrl+Shift+<}
```

DdePoke Command

The DdePoke command sets the value of a specific item in a DDE application, even if the application is not currently active (that is, even if it is minimized or does not have the focus). However, you get an error message if the application is not running.

This command is useful when you must send information to an application. For example, the following line sets to 10 the value of row 1, column 2 in the worksheet “sheet1” in Microsoft Excel, even if Excel is not currently active:

```
DdePoke "excel", "sheet1", "r1c2", "10"
```

Chapter 4 Using the Global.dvc File

This chapter describes the Global.dvc file. It includes the following topics:

- An overview of the file’s function and how you can use it
- How Dragon NaturallySpeaking manages and uses the file
- The file format, including descriptions of all file elements
- Rules and conventions for file contents

Introduction to the Global.dvc File

Dragon NaturallySpeaking keeps all voice commands in a file named Global.dvc. In the Deluxe and Professional editions it is a plain (8-bit ANSI) text file, and you can edit it manually. In all other editions it is encrypted and you cannot modify it. The New Command and Edit Command wizards only modify plain-text .dvc files.

You can create a large number of commands more quickly by editing the Global.dvc file than by using the New Command wizard. However, the command wizards check for common errors as you create and edit your commands. Therefore, you should not use this method until you are very familiar with creating commands.

Once you have created and tested your commands, you use the Global.dvc file to deliver them to another Deluxe or Professional edition user.

How Dragon NaturallySpeaking Uses the Global.dvc file

When you open a user, Dragon NaturallySpeaking loads voice commands from the Global.dvc file. (Dictation commands such as All Caps and Correct That are hard-coded and are not in Global.dvc.) If there is a Global.dvc file in the *NatSpeak*\Users\userName\Current directory, Dragon NaturallySpeaking loads that file. Otherwise, it loads the copy in the \NatSpeak\Data directory.

When you select Finish after using a command wizard to create or edit a command, Dragon NaturallySpeaking writes the command to the

NatSpeak\Users\Username\Current\Global.dvc file. If the file does not exist yet, it creates it with the contents of the master Global.dvc file plus your changes. Even if you do not save your speech files, any commands you add or change are saved.

Because Dragon NaturallySpeaking reads the Global.dvc file only when it loads a user, and because it writes to the file when you use the command wizards, you should *always* close Dragon NaturallySpeaking or open a different user before you edit your *Username*\Current\ Global.dvc file. When you next start Dragon NaturallySpeaking or open the user you edited, Dragon NaturallySpeaking will read your updated file and you can use your new commands.

Dragon NaturallySpeaking checks for syntax errors in the Global.dvc file as it loads the file. When it finds an error it displays a message indicating the line that contains the error and does not load any voice commands.

The .DVC File Format

The Global.dvc file has the following format:

```
MENU "appName" {
        STATE "stateName" [stateType] {
                COMMAND "commandName" {
                        SCRIPT | KEYS {
                            script or keystroke sequence
                        }
                }
                .
                .
                LIST "listName" {
                        "list entry"
                        .
                        .
                }
        .
        .
        }
    .
    .
}
.
.
```

The file consists of a number of hierarchically nested sections as follows:

- Each section consists of a section type keyword identifier, a section name, and the section contents, which is delimited by braces ({}).
- Each MENU section contains one or more STATE sections.
- Each STATE section contains one or more COMMAND definitions followed by zero or more LIST definitions.
- Each COMMAND definition contains either a SCRIPT definition or a KEYS definition.

The following sections document this structure in detail.

MENU

The MENU keyword specifies the beginning of a menu, which contains the commands for a particular application (or global commands). Each menu contains one or more states.

appName

The *appName* variable specifies the executable file name of the application without the .exe suffix. The *appName* is not case sensitive, but you should use all uppercase characters for consistency with names generated by Dragon NaturallySpeaking. The menu containing the standard Dragon NaturallySpeaking global commands has the *appName* Global Commands. For more information on global menus and states, see the stateType Keywords on page 36.

You do not always see the *appName* in the New and Edit Command wizards. Instead, these wizards often use more descriptive names. The Windows Registry \HKEY_LOCAL_MACHINE\Software\Dragon Systems\NaturallySpeaking\Professional 3.0\Applications key contains subkeys that associate the Command wizard descriptive names with the names used in the Global.dvc file. You should not edit the Registry entries, but you can view them for information about the mapping between the names. Use the Change Descriptive Names button on the New Command wizard to change the name that the wizard displays.

State

The STATE keyword specifies the beginning of a recognition state. The state contains the commands that are recognized when the specified window (of the menu application) is active. Each state contains one or more commands and lists.

Note that Dragon NaturallySpeaking has a few special states, such as the Asleep state and the Mouse Grid state that do not have corresponding windows. Also, the *stateType* keyword can specify special state behavior.

stateName

The *stateName* identifies the window and must appear somewhere in the active window's title bar. The name can have up to 32 characters and is case sensitive. It can be any portion of the title bar text and need not start or end with whole words. However, it should be detailed enough to uniquely identify the target window. For example, if the target window title bar looks like:

Turbo TextEdIt - C:\Proposals\NewWing.txt

you might use the following *stateName* values:

TextEdIt
If you want this state to apply to all members of the TextEdIt family of applications. (However, they must all have the same executable name.)

Turbo TextEdIt
If you only want the commands to apply to this particular version.

NewWing.txt
If you want the commands to apply to any document named NewWing.txt, independent of the file's location or the editor version.

Turbo TextEdIt - C:\Proposals\NewWing.txt
If you want the commands to apply to the Turbo TextEdIt edition only when the active window contains the NewWing.txt document from the C:\Proposals folder.

The *stateName* corresponds to the information you specify in response to the New Command wizard's "select the title of the target window or dialog box for this command" prompt.

stateType Keywords

The *stateName* can be followed by GLOBAL, NOGLOBALS, or SLEEPING. These special keywords control how Dragon NaturallySpeaking uses this state:

GLOBAL
This keyword specifies that the state is global and Dragon

NaturallySpeaking will recognize its commands in all applications and windows.

The Dragon NaturallySpeaking Edit Command wizard and New Command wizard display only one global state: the Global Commands state in the Global Commands menu. However, Dragon NaturallySpeaking recognizes commands from all global states. Therefore, you can create your own global state if you want to create a set of “hidden” global commands. You can use any unique *menuName* and *stateName* combination; the names do not have to correspond to an application or window.

In some cases you may have commands that you want to be global, but are really only useful if you have a particular application. For example, you might want to create a command that switches the focus to your application and then sends it keystrokes. In this case, you can create a state with an arbitrary *stateName* and a GLOBAL *stateType*, and put it in your application’s menu. This state is truly global—its commands are recognized in all applications, independent of the window title bar. It is also hidden from the command wizards.

NOGLOBALS
Tells Dragon NaturallySpeaking not to recognize any global commands when this state is active. The Dragon NaturallySpeaking Mouse Grid state uses this keyword to ensure that only MouseGrid commands get recognized while the MouseGrid is displayed.

SLEEPING
Tells Dragon NaturallySpeaking to recognize these commands in this state only when it is in sleep mode. Sleep mode is designed as a mode in which Dragon NaturallySpeaking will do nothing except wake up. As a result, Dragon NaturallySpeaking does not send keystrokes to applications in sleep mode.

The Global.dvc Asleep state uses this keyword. You should not create your own SLEEPING state. You may add additional words to the Asleep state if you wish to add your own aliases for the Wake Up command.

COMMAND

The COMMAND keyword identifies a voice command definition. This definition consists of the voice command name and either a script or a set of keystrokes, as identified by the SCRIPT or KEYS keyword. If the command name contains any list names (in angle brackets) the lists must be defined in the current state.

KEYS

The KEYS keyword identifies a keystroke command, which consists of one or more keystrokes, and cannot include scripting commands. The keystroke command must follow the rules for key definitions described on page 13, the scripting language conventions described on page 14, and the .dvc file conventions described starting on page 39.

For example, the following keystroke command is in the Dragon NaturallySpeaking state of the NATSPEAK menu. It is active when the Dragon NaturallySpeaking word processor window is active and lets you select a paragraph by voice. Note that because this is a keystroke command, the keystroke sequence is *not* in quotation marks.

```
COMMAND "Select Paragraph" {
  KEYS {
    {Ctrl+Down}{Shift+Ctrl+Up}
  }
}
```

SCRIPT

The SCRIPT keyword identifies a voice command script, which consists of one or more scripting commands. The script must follow the scripting command conventions described on page 14 and the .dvc file conventions described on page 39.

The following script command is in the Global Commands state of the Global Commands menu. It is active at all times and lets you change the active window to the Dragon NaturallySpeaking word processor or either the next or previous window in the Windows tab sequence.

```
COMMAND "Switch to <AppList>" {
  SCRIPT {
    if _arg1 = "NatSpeak" then AppBringUp "NatSpeak"
    if _arg1 = "NaturallySpeaking" then AppBringUp "NatSpeak"
    if _arg1 = "Next Window" then SendSystemKeys "{Shift+Alt+Tab}"
    if _arg1 = "Previous Window" then SendSystemKeys "{Alt+Tab}"
  }
}
```

LIST

The LIST keyword identifies a list of words or phrases for use in the variable part of a command name. The list can include entries in the written form\\spoken form format, such as a\\alpha. (Note that you must use a double-backslash in the Global.dvc file.)

The list is local to the state. The list name can only be used by commands in the current state. However, different states can have lists with identical names; these lists can have different contents.

By convention, the standard Dragon NaturallySpeaking Global.dvc file puts list definitions at the end of each state, following all command definitions. However, this organization is not required. For example, you could put a list that is used in a single command before or after that command.

The following list is at the end of the Microsoft Word state of the WINWORD 8.0 menu. It is used in several Natural Language commands for Microsoft Word 97.

```
LIST "thatItFontSelection" {
    "That"
    "This"
    "It"
    "Them"
    "Font"
    "Selection"
    "the Font"
    "the Selection"
}
```

DVC File Rules and Conventions

The following sections define the rules for writing DVC files.

General DVC File Rules and Conventions

The following rules apply to the overall structure and contents of the DVC file.

- Enclose the body of each MENU, STATE, COMMAND, SCRIPT, KEYS, or LIST section (following the name argument) in braces ({}).

- If a section consists of a single line you can enclose it in quotes instead of braces, for example, SCRIPT “WakeUp”.
- An opening brace ({) must be on the same line as the section keyword (such as SCRIPT).
- A closing brace (}) must be on a line by itself (preceded only by tabs or spaces and followed only by a line break or return character). This prevents ambiguity when key names of the form {*key-name*} are present in the text.
- Delimit all string arguments, including list entries and the section names that follow keywords, in quotation marks (").
- Do *not* put quotation marks around the keystrokes in a KEYS section.
- Put each keystroke sequence or scripting command on one line. However, you can continue a keystroke command onto subsequent lines by using the escape character (\), as described in the next section.
- White space and line breaks following an open brace ({) are ignored, so you can start the text on the next line and indent it.
- White space (tabs and/or spaces) at the beginning of each line is ignored.
- Trailing white space at the end of a line is retained.
- Remember that arguments are case sensitive. The keywords (MENU, STATE, etc.) are not case sensitive, but you should use all uppercase characters to ensure consistency.
- Use a double backslash (\\) to indicate a single backslash (\) character in a list, for example, between the written and spoken form of a word.
- Use \" to include a quotation mark in a list.
- Use the # character anywhere in a line to start a comment. You cannot use this character as the comment character in scripts, however. Use the ' character only for comments inside scripts.
- In HeardWord commands, use a single backslash before dictation commands that do not have a written form. For example, use HeardWord “\Caps”. To see a complete list of the words that require this treatment, open the Vocabulary Editor and scroll up. The words at the top of the list, which show a spoken form only, require a leading \ character. For more information on using dictation commands in HeardWord scripting commands, see page 91.

KEYS Section Escape Sequences

You can use the following C-standard escape sequences in keystroke sequences following the KEYS keyword. You cannot use them elsewhere in the .dvc file.

\t	tab character (0x9)
\n	newline character (0xa). The \n is equivalent to {Enter}, as is a line break.
\r	return (0xd)
\x	followed by 1 or 2 hexadecimal digits will be written out as a single byte containing the hexadecimal value. However, you should use two digits (for example \x02, to avoid including a subsequent character that happens to be a valid hexadecimal digit. For example, “\x1Arthur” translates to “\1a” followed by “rthur”.
\\	literal \ (backslash)
\}	literal } (close brace)
\"	literal " (double-quote)
\	(backslash space) a literal space. You can use this to force a leading space.
\	(backslash at end of line) discards the line break and wraps the current line onto the next line. However, the leading white space on the next line is still ignored, so you can indent as much as you like for readability. Word wrapping allows you to enter as long a line as you like and still have it readable in your text editor.

DVC file Sample

The following code illustrates sections of a customized Global.dvc file:

```
MENU "Global Commands" {
   STATE "Global Commands" GLOBAL {
# Note that the \n sequences are required to put the
# text on three different lines.
      COMMAND "Return Address" {
         KEYS {
            Dragon Systems \n
            320 Nevada Street \n
            Newton, MA 02460
         }
      }
      COMMAND "Go to Sleep" {
         SCRIPT "GoToSleep"
      }
   }
.
.
.
   # The Asleep state has only one command - Wake Up
   STATE "Asleep" GLOBAL SLEEPING {
         COMMAND "Wake Up" {
            SCRIPT "WakeUp"
         }
   }
}
.
.
.
# This is a comment outside a script
# The following menu defines application-specific
# commands for a system backup utility.

MENU "ARCSRV32" {
   STATE "ARCserve Agent" {
      COMMAND "Close Agent <NowOrLater>" {
         SCRIPT {
            If _arg1 = "Later" then Wait 2000
            SendKeys "{Alt+F4}"
         'This is a comment in a script.
            SendKeys "{Enter}"
         }
      }
      LIST "NowOrLater" {
         "Now"
         "Later"
      }
   }
}
```

Chapter 5 Scripting Language Reference

This chapter describes the elements of the Dragon NaturallySpeaking scripting language:

- Expressions
- Variables
- Operators
- Data manipulation functions
- Flow control statements
- Scripting commands
- Key names

Expressions

You can use expressions as arguments to scripting commands as long as the expressions can be converted to the argument data types, which must be strings or 16-bit integers (-32768–32767). The scripting language interpreter converts the value of an expression to the expected data type whenever possible and reports an error if it cannot do so.

Example

Many Dragon NaturallySpeaking command scripts use string expressions. For example, the Set Font <FontFace> script contains the following line:

```
SendKeys "{Alt+o}f{Alt+f}"+_arg1+"{Enter}"
```

This concatenates the keystrokes that open the Font dialog box and selects the Font field with the FontFace variable and the Enter keystroke that completes the operation.

Variables

The scripting language supports variables with the following BASIC programming language data types:

Ending Character	**Type**
$	String
%	A 16-bit (positive or negative) integer
&	A 32-bit (positive or negative) integer
#	A double-precision (8-byte) floating-point
(none)	Variant; can be any of the above (default)

Variable names conform to the name rules for Visual Basic®:

- Names are not case sensitive.
- The first character must be alphabetic (A–Z, a–z) or an underscore (_).
- Remaining characters can be alphabetic, numeric (0–9), or an underscore.
- The final character is an optional special type identifier character ($%&#). The variant type has no terminator.

Dragon NaturallySpeaking defines variables dynamically, when they are first referred to. If Dragon NaturallySpeaking uses a variable before it has been assigned a value, the variable value is 0 or an empty string.

You cannot use the same base variable name with multiple type identifiers. For example, you cannot use both the variable name “Color$” and the variable name “Color”, or partno% and partno&, in one voice command.

Notes

Dragon NaturallySpeaking automatically generates a variable name “_arg*n*”, for each list entry in a command that uses lists, where *x* corresponds to the position of the list name in the command. For example, the script for the Move <DirLeftRight> <2To20> Characters must use _arg1 for the variable from the DirLeftRight list and _arg2 for the variable from the 2To20 list.

You cannot use a scripting language keyword (such as a scripting command or a flow control statement) as a variable name.

The scripting language does not have a Dim statement or Global variables.

Scripting language commands only take strings or 16-bit integers as arguments. However, you can use other types in a script and explicitly convert their values to a variable of the required type. You can also let Dragon NaturallySpeaking automatically convert argument variables to the required type if you are sure that the conversion results in a valid value.

See Also

"List Variables in Scripts" section on page 27.

Operators

Dragon NaturallySpeaking supports these types of operators:

- Arithmetic
- Comparison
- Logical
- String concatenation

When expressions contain operators from more than one class, the arithmetic operators (and string concatenation) are evaluated first, followed by comparison operators and then logical operators. As with most programming languages, you can use parentheses to explicitly control the order of evaluation.

The following pages list operators in order of decreasing precedence.

Arithmetic Operators

Syntax	Operation
- a	Negation
a * b	Multiplication
a / b	Division (same precedence as multiplication)
a + b	Addition (+ is also used for concatenating strings)
a - b	Subtraction (same precedence as addition)

Examples

The following expression increments a 16-bit word counter:

```
Wordcount% = Wordcount% + 1
```

The following expression adds the Ones variable value to the product of the Tens variable and 10.

Result = Ones + Tens * 10

Comparison Operators

Syntax	Operation
a = b	Equal to
a <> b	Not equal to
a < b	Less than
a > b	Greater than
a <= b	Less than or equal to
a >= b	Greater than or equal to

All comparison operators have the same precedence. They are evaluated from left to right.

Example

The Delete <NextOrPrevious> Character command script compares the value of a list variable with string values in the list to determine the operation that is required:

```
COMMAND "Delete <NextOrPrevious> Character" {
  SCRIPT {
    if _arg1 = "Previous" then SendKeys "{Backspace}"
    if _arg1 = "Next"     then SendKeys "{Del}"
    if _arg1 = "Back"     then SendKeys "{Backspace}"
    if _arg1 = "Forward"  then SendKeys "{Del}"
    if _arg1 = "Last"     then SendKeys "{Backspace}"
  }
}the
```

Logical Operators

Syntax	Operation
NOT a	Logical NOT; performs negation
a AND b	Logical AND; true only if both are true
a OR b	Logical OR (inclusive OR); true if one or the other is true, or if both are true
a XOR b	Logical XOR (exclusive OR); true if one or the other is true, but not if both are true

Example

The following expression (from the <playPlaybackRead> <ReadText> command) compares the _arg2 variable from the ReadText list with both the string values “That” and “Selection”, and sends keystrokes if either comparison is true.

```
if _arg2 = "That" or _arg2 = "Selection" then
SendKeys "{Ctrl+"+key$+"+s}"
```

String Concatenation Operator

Syntax	Operation
“abc”+“def”	string concatenation (result is “abcdef”)

Example

The following command uses concatenation to combine string literal values and variables to create a single string.

```
SendKeys "{"+Direction+" "+Digits+"}"
```

The variables Direction and Digits are string variables. The concatenation operator combines the values of these variables with the string literals “{” , “ ” (space), and “}”. If the value of Direction is “down” and the value of Digits is “3”, the concatenation result is “{down 3}”.

See Also

“Using String Concatenation” section on page 24.

Data Manipulation Functions

The scripting language provides the following data manipulation functions. These functions are compatible with most BASIC languages.

Function	**Description**
CDbl	Converts to a double-precision floating-point number
Chr$	Returns the character specified by an ANSI (ISO 8859-1) code
CInt	Converts to a 16-bit integer
CStr	Converts to a string
Hex$	Converts to a string representation of a hexadecimal value
Instr	Returns the position of a character in a string
LCase$	Converts to all lowercase characters
Len	Returns the length of a string
Mid$	Selects part of a string
Str$	Converts an integer to a string
String$	Returns a string that repeats one character
UCase$	Converts to all uppercase characters
Val	Converts numbers in a string to an integer

CDbl

Syntax

`CDbl(`*`expression`*`)`

Description

Converts any valid string or numeric *expression* into a double-precision floating-point number. A string must represent a valid floating-point number. If the conversion fails, the system displays an error message.

Example

The following expression explicitly converts the value of the 16-bit variable sint% to a double-precision floating-point number and assigns its value to the variable dfloat#:

```
dfloat# = CDbl(sint%)
```

See Also

CInt
CStr

Chr$

Syntax

`Chr$(`*`integer%)`*

Description

Returns a string consisting of the character that corresponds to the specified ANSI code. For example, Chr$(9) returns a Tab character.

Example

The following line assigns the value of the ANSI newline character to the variable NL$. You can then use the NL$ variable to force new lines in message strings.

```
NL$ = Chr$(10)
```

See Also

Hex$
Str$
String$

CInt

Syntax

`CInt(`*`expression`*`)`

Description

Converts any valid string or numeric *expression* into a 16-bit integer. The string can only have digits (0–9), optionally preceded by a minus sign
(-). You cannot use a plus sign or a decimal point. The function rounds any floating-point number to the nearest integer. If the conversion fails, it displays an error message.

Examples

The following statement converts the string representation of the number 123 into an integer:

```
MyInt% = CInt("123")
```

The following example converts the results of a floating-point calculation into an integer (rounding to the nearest integer):

```
Circum% = CInt(Diam# * 3.14)
```

See Also

CDbl
CStr

CStr

Syntax

`CStr(`*`expression`*`)`

Description

Converts any valid *expression* (including numbers and strings) into a string representation of its value. If the conversion fails, the system displays an error message.

Examples

The following expression converts the expression 3 + 27 to the string “30”:

```
Strval$ = CStr(3 + 27)
```

The following script types 12.141:

```
afloat# = 3.141
anint% = 7
SendKeys CStr(afloat# + 2 + anint%)
```

See Also

CDbl
CInt

Hex$

Syntax

`Hex$(`*`expression)`*

Description

Converts a valid integer, string, or expression to a string representation of its hexadecimal value. Any string or string expression must represent a decimal integer. Therefore, the string must consist only of digits (0–9). Any expression must be convertible to an integer.

Example

The following statement sets the string variable Hexnum$ to “1b”:

```
intval% = 27
Hexnum$ = Hex$(intval%)
```

The following script types the letter “c”, the hexadecimal representation of 12:

```
afloat# = 3.141
anint% = 7
SendKeys Hex$(afloat# + 2 + anint%)
```

See Also

Chr$
Str$
String$

Instr

Syntax

`Instr([start%,] str1$, str2$)`

Description

Returns an integer representing the position of the first occurrence of string *str2$* within string *str1$*. (It returns the position of the first character of *str2$*.) The position of the first character in a string is 1. If you include a *start%* numeric parameter, the function begins checking at the *start*th character of *str1$*.

The function Instr returns 0, indicating an error condition if:

- The *start%* value is greater than the length of *str1$*
- The *str1$* variable is zero-length
- It cannot find *str2$* in *str$1*

Examples

If *str1$* is "Paragraph|Enter", the following statement sets the variable I to 10, which is the location of the "|" character in the string.

```
I = Instr(str1$, "|")
```

The scripts for the Dragon NaturallySpeaking "Press Key" commands use Instr statements and lists whose entries contain variable number of words to increase the flexibility of the grammar. For example, the <PressKey> <ShiftKey> <Alphabet> command script is:

```
COMMAND "<PressKey> <ShiftKey> <Alphabet>" {
  SCRIPT {
    tstr = "{"
   if Instr(_arg2, "control-key") > 0 then tstr = tstr +
        "Ctrl+"
    if Instr(_arg2, "alt-key") > 0 then tstr = tstr + "Alt+"
    if Instr(_arg2, "shift-key") > 0 then tstr = tstr +
        "Shift+"
    tstr = tstr + MID$(_arg3,0,1) + "}"
    SendSystemKeys tstr
   }
  }
```

The ShiftKey list contains all 15 possible combinations of "shift-key", "alt-key", and "control-key". The use of Instr enables the script to cover all these combinations with only three statements, one per key type.

See Also

Len
Mid$

LCase$

Syntax

`LCase$(`*`s$`*`)`

Description

Returns string *s$* in all lowercase characters.

Example

The following statement assigns the value "telephone" to the variable Phone$:

```
Phone$ = LCase$("TELEphone")
```

See Also

UCase$

Len

Syntax

`Len(`*`s$`*`)`

Description

Returns the number of characters in string *s$*.

Example

If *str1$* is “Paragraph|Enter”, the following statement sets the variable strlen% to 15:

```
strlen% = Len(str1$)
```

See Also

Instr
Mid$
Val

Mid$

Syntax

`Mid$(`*`s$, start%`*`[, `*`len%`*`])`

Description

Returns *len%* characters from string *s$*, starting with the character at the *start%* position. A *start%* value of 0 specifies the first character of the string. If you omit *len%*, the function returns all of the string from *start%* to the end.

Examples

The following statement sets the value of the variable Midstring$ to "is":

```
Midstring$ = Mid$("This is a test", 5, 2)
```

The <PressKey> <ICAlphabet> command lets you say "Press alpha" to type "a". The ICAlphabet list contains entries such as "a\alpha", which includes both the written form (a) and spoken form (alpha) of each word. The command script consists of the following line, which extracts the first character from the returned list variable value and sends the corresponding key to the system.

```
SendSystemKeys MID$(_arg2,0,1)
```

See Also

Instr
Len

Str$

Syntax

`Str$(`*`numericexpression`*`)`

Description

Returns a string representation of a numeric expression. This function is useful when you do integer arithmetic on a spoken number and want to use the result in a SendKeys command. Unlike the corresponding Microsoft Visual Basic function, Str$ does not reserve a leading space for the sign, so it does not put a space in front of an unsigned positive number.

Examples

The following statement sets the variable Fifteen$ to “15”:

```
Fifteen$ = Str$(15)
```

A Solitaire program vocabulary could include a Turn Over <Pile> command to let you say commands such as “Turn Over 7” to turn over the card in the seventh pile. Its script could be:

```
P1=Str$(7-Val(_arg1))
SendKeys "{End}{Left "+P1+"}{Space}{Esc}"
```

The first line of the script:

1. Uses the Val function to convert the pile number that you say into an integer value.
2. Subtracts the result from 7, to get the number of {Left} keystrokes that are required.
3. Uses the Str$ function to convert the resulting integer into a string, because the SendKeys command requires a string expression.

See Also

Chr$
Hex$
String$

String$

Syntax

`String$(`*`count%`*`, `*`code%`*`)`

`String$(`*`count%`*`, `*`s$`*`)`

Description

The first format returns a string of *count%* characters whose ANSI numeric code is *code%*.

The second format returns a string of *count%* copies of the first character in *s$*.

Examples

The following statement sets FiveNL$ to five newline characters:

```
FiveNL$ = String$(5, 10)
```

The following statement sets ThreeTs$ to “TTT”:

```
ThreeTs$ = String$(3, "Test")
```

See Also

Chr$
Hex$
Str$

UCase$

Syntax

`UCase$(`*`s$`*`)`

Description

Returns string *s$* in all uppercase characters.

Example

The following statement assigns the value “TELEPHONE” to the variable Phone$:

```
Phone$ = UCase$("TELEphone")
```

See Also

LCase$

Val

Syntax `Val(`*`s$`*`)`

Description Returns the numeric value of the string expression *s$*. The string expression must represent a valid integer or floating-point number. The function returns 0 if the expression does not start with a valid number. This function is useful if you wish to do arithmetic using spoken numbers.

Example The Scratch That <2To10> Times command has the following script, which uses the value of the <2To10> list variable as an index in a While loop. If you say “Scratch that 8 times”, Dragon NaturallySpeaking recognizes the word “8” from the 2To10 list, and the Val statement converts the string to an integer value. The While loop then runs 8 times.

```
loop& = Val(_arg1)
while ( loop& )
    HeardWord "Scratch","That"
    loop& = loop& - 1
wend
```

See Also Len

Flow Control Statements

The scripting language provides the following flow control statements:

- IF THEN
- IF THEN ELSE
- Labels and GOTO Statements
- DO UNTIL
- DO WHILE
- LOOP UNTIL
- LOOP WHILE
- WHILE

CAUTION

Any code that causes an infinite loop will hang Dragon NaturallySpeaking.

Note

The scripting language does not have a FOR loop. However, you can construct an equivalent by using the other looping commands and GOTO commands.

See Also

“Flow Control” section on page 26.

IF THEN

Syntax

`IF` *`expression`* `THEN` *`statement`*

Description

If *expression* is true, then Dragon NaturallySpeaking executes *statement*. Otherwise, it skips to the next statement in the script.

In this simple version of the IF THEN statement you can put only one statement after the THEN keyword, and you cannot have a newline character before the end of the statement.

Syntax

```
IF expression THEN
   statement
   .
   .
END IF
```

Description

This multiline form of the IF THEN statement lets you include multiple statements after the THEN keyword. In this form, the THEN keyword ends the IF line. Each statement must be on a separate line and an END IF keyword must be on its own line, after the last statement line. (Note that there is a space between END and IF.)

Dragon NaturallySpeaking executes all the statements between the THEN keyword and the END IF keywords when the expression is true.

Example

The Move <DirLeftRight> <One> Character command moves the insertion point one character to the left or right. It has the following script. (Note that the IF and THEN keywords are in lower case.)

```
if _arg1 = "Back"     then _arg1 = "Left"
if _arg1 = "Forward"  then _arg1 = "Right"
SendKeys "{" + _arg1 + "}"
```

The following example uses the multiline form of the IF THEN statement:

```
IF part1$ = "tele" THEN
part2$ = "phone"
Word = part1$+part2$
END IF
```

See Also

IF THEN ELSE statement

IF THEN ELSE

Syntax:

```
IF expression THEN
   statement
   .
   .
   .
ELSE
   statement
   .
   .
   .
END IF
```

Description

Each *statement* and the ELSE and END IF keywords must be on separate lines. Also, the THEN keyword must be at the end of a line.

Example

The following text from the Format That <Formatting> command for Microsoft Word 97 contains two nested IF THEN ELSE statements. Each IF statement compares the value of the Formatting list variable with a specific string. The second IF statement gets evaluated only if the first IF statement is false, that is, if you did not say “Format That With Hyphens”. Note that in this code the IF, THEN, and ELSE keywords are in lower case.

```
if _arg1 = "With Hyphens" then
  HeardWord "Hyphenate”, “That"
else
  if _arg1 = "Without Spaces" then
    HeardWord "Compound”, “That"
  else
  .
  .
  end if
end if
```

See Also

IF THEN statement

Labels and GOTO

Syntax

`label: [statement]`

Description

A label identifies a single line of code. It can be any combination of 40 or fewer characters that starts with a letter and ends with a colon. Any text after the colon is interpreted as a statement. Labels are not case sensitive and must be the first nonblank characters on a line.

A label serves as a potential target for a GOTO statement. GOTO statements have the form:

Syntax

`GOTO label`

Description

This statement causes execution to continue from the first statement or command following the matching *label*.

Note

You cannot use a scripting language keyword (such as a scripting command or a flow control statement) as a label.

Example

The following Who <Results> command tells you who won or who lost, but not both. The Results list has two words:

Won
Lost

The command script is:

```
IF _arg1 = "Lost" THEN GOTO Lost
SendKeys "Pete Won" + Chr$(10)
GOTO Endit
Lost: SendKeys "Ivan Lost" + Chr$(10)
Endit:
SendKeys "More scores will be available later."
```

Note that the Lost: label is on the same line as the code it labels, while the Endit label precedes the code line. Also, note that End is not a valid label name.

DO UNTIL

Syntax

```
DO UNTIL expression
   statement
   .
   .
   .
LOOP
```

Description

Dragon NaturallySpeaking evaluates *expression* first. If *expression* is false, it performs the *statement*(s), including the LOOP back to the *expression* evaluation. When *expression* is true, execution continues at the command or statement that follows the LOOP keyword.

Example

The voice command script for a Solitaire vocabulary Finish the Game command could use two DO UNTIL loops. The outer loop repeats the inner loop 10 times to account for all the cards that may be in a stack. The inner loop double-clicks on a stack and selects the next stack to the left. (In solitaire, the {End} key selects the rightmost stack of cards, and the {Left} key selects the next stack to the left.)

```
I=10
DO UNTIL I=0
   SendKeys "{End}"
   j=6
   DO UNTIL j=0
      ButtonClick 1,2
      SendKeys "{Left}"
      j=j-1
   LOOP
   ButtonClick 1,2
   I=I-1
LOOP
```

See Also

LOOP WHILE statement
LOOP UNTIL statement
DO WHILE statement

DO WHILE

Syntax

```
DO WHILE expression
   statement
   .
   .
   .
LOOP
```

Description

Dragon NaturallySpeaking evaluates *expression* first. If *expression* is true, it performs the *statement*(s), including the LOOP back to the *expression* evaluation. When *expression* is false, execution continues at the command or statement that follows the LOOP keyword.

Note

The DO WHILE/LOOP form is logically identical to WHILE/WEND, described on page 73.

Example

The following script is a version of the Scratch That <2To10> Times command (shown on page 73), rewritten to use DO WHILE instead of WHILE. The command uses the value of the number recognized from the 2To10 list as a loop counter.

```
loop& = Val(_arg1)
do while ( loop& )
  HeardWord "Scratch","That"
  loop& = loop& - 1
loop
```

See Also

LOOP WHILE statement
LOOP UNTIL statement
DO UNTIL statement

LOOP UNTIL

Syntax

```
DO
   statement
   .
   .
   .
LOOP UNTIL expression
```

Description

Dragon NaturallySpeaking does the *statement*(s) at least once. It evaluates the conditional expression at the end of the loop and looping continues as long as *expression* remains false. When *expression* is true, execution continues at the command or statement that follows *expression*. This type of loop is useful when you know that the operation must be done at least once.

Example

The following code double-clicks the left mouse button and sends the insertion point left by one unit. It repeats this operation 6 times. This code has the same result as the code in the LOOP WHILE example.

```
j = 6
DO
  ButtonClick 1,2
  SendKeys "{left}"
  j=j-1
LOOP UNTIL j=0
```

See Also

LOOP WHILE statement
DO WHILE statement
DO UNTIL statement

LOOP WHILE

Syntax

```
DO
   statement
   .
   .
   .
LOOP WHILE expression
```

Description

Dragon NaturallySpeaking performs the *statement(s)* at least once. It evaluates the conditional expression at the end of the loop and looping continues while *expression* remains true. When *expression* is false, execution continues at the command or statement that follows *expression.* This type of loop is useful when you know that the operation must be done at least once.

Example

The following code double-clicks the left mouse button and sends the insertion point left by one unit. It repeats this operation 6 times. This code has the same result as the code in the LOOP UNTIL example.

```
j = 6
DO
  ButtonClick 1,2
  SendKeys "{Left}"
  j=j-1
LOOP WHILE j>0
```

See Also

LOOP UNTIL statement
DO WHILE statement
DO UNTIL statement

WHILE

Syntax

```
WHILE expression
   statement
   .
   .
   .
WEND
```

Description

Dragon NaturallySpeaking evaluates the *expression*. If it is true, Dragon NaturallySpeaking performs the *statement*(s) down to WEND and control loops back to the *expression* evaluation. When *expression* is false, execution continues at the command or statement that follows WEND.

Note

This form is logically identical to DO WHILE/LOOP.

Example

The Scratch That <2To10> Times command has the following script, which uses the value of the number recognized from the 2To10 list as the loop counter.

```
loop& = Val(_arg1)
while ( loop& )
  HeardWord "Scratch","That"
  loop& = loop& - 1
wend
```

See Also

LOOP WHILE statement
LOOP UNTIL statement
DO WHILE statement
DO UNTIL statement

Scripting Commands

This table lists the scripting commands by functional area and provides a brief description of each command. Full command descriptions follow, in alphabetical order.

Command	**Description**
Application Control	
AppBringUp	Starts or switches to an application.
AppSwapWith	Swaps the current application with another.
ClearDesktop	Minimizes all applications.
DdeExecute	Sends a DDE command.
DdePoke	Sets an item value in a DDE application.
DllCall	Calls a function in a DLL.
ShellExecute	Loads an application.
WinHelp	Runs Windows Help.
Dragon NaturallySpeaking Control	
GoToSleep	Puts Dragon NaturallySpeaking in sleep mode.
SetMicrophone	Turns the microphone on or off.
WakeUp	Resumes normal recognition.
Mouse/Insertion Point Control	
ButtonClick	Clicks the specified mouse button.
DragToPoint	Drags the mouse to the current point.
MouseGrid	Uses MouseGrid to position the mouse pointer.
RememberPoint	Records the current mouse pointer position.
SetMousePosition	Places the mouse pointer at a specified position.

(Continued)

(Continued)

Command	**Description**
Recognition Control	
HeardWord	Invokes one or more word actions.
Sound Control	
Beep	Plays Windows beep sound.
PlaySound	Plays a .wav file.
Script Execution Control	
MsgBoxConfirm	Asks a user to confirm an action.
RunScriptFile	Executes a specified script.
Wait	Makes the command script pause.
Text Entry Control	
SendKeys	Sends keystrokes to the active window.
SendSystemKeys	Sends system keystrokes to Windows.
Text-to-Speech Control	
TTSPlayString	Sends a request to the text-to-speech utility to play text.
Window Control	
ControlPick	Selects a control such as a button or prompt.
MenuCancel	Cancels the current menu.
MenuPick	Selects a menu or menu item.

AppBringUp

Syntax

```
AppBringUp "applicationName"[, "commandLine"[,
windowStyle[, "directory"]]]
```

Description

Starts or makes active an application. The command operation depends on the active applications and the command parameters, as follows:

1. If *applicationName* was used in a previous AppBringUp or AppSwapWith command and a copy of the application is running, the command tries to make that copy the active application. If AppBringUp cannot activate it, the command starts a second copy.

2. If there is no *commandLine* argument, Dragon NaturallySpeaking does the following:

   a. If the *applicationName* is the executable name of an unnamed instance of a program that is already loaded, Dragon NaturallySpeaking brings up the instance and assigns it *applicationName*. (Programs that are not started by AppBringUp or AppSwapWith are unnamed.)

   b. Otherwise, Dragon NaturallySpeaking runs any executable named *applicationName* that it locates in your PATH, the Windows directory, the Windows/System directory, or the *NatSpeak/*Program directory.

3. If there is a *commandLine* argument, Dragon NaturallySpeaking looks for a running copy of the program specified by this argument.

   a. If it finds a running application that can support only one instance, it brings up the existing instance and gives it the name *applicationName*.

   b. If it finds a running copy of the application that supports multiple instances, it brings up a new copy and gives it the name *applicationName*.

   c. If it does not find a running copy of the program, it starts the program specified by the *commandLine* argument and assigns it *applicationName*.

The Dragon NaturallySpeaking Switch to <AppList> command uses this scripting command.

Arguments

applicationName	The name by which Dragon NaturallySpeaking identifies the program to bring up. The application is assigned the name the first time you use it in this command. This name is meaningful to Dragon NaturallySpeaking only and does not affect the application title bar.
commandLine	The application program and any arguments such as a document file or command switches. You do not have to include the application's .EXE extension. If the *commandLine* does not specify a full file path, the application must be in the current, Windows, Windows\System, or Dragon NaturallySpeaking directory, or in a directory listed in the PATH variable. The default value for *commandLine* is the same as the value specified in *applicationName.*
windowStyle	A number specifying how the window appears when you activate the application:
directory	Specifies the working directory for the application. The default is the directory containing the program's executable file.

windowStyle values:

1, 5, or 9	Normal, active (default)
2	Minimized, active
3	Maximized, active
4 or 8	Normal, inactive
6 or 7	Minimized, inactive

If the application is already running, this value has no effect.

Examples

Example 1

This example starts or reactivates WordPad, opens the Dragon NaturallySpeaking Readme file in a maximized window, and sets the working folder to C:\My Documents.

```
AppBringUp "Readme", "WordPad C:\Natspeak\Help\
Readme.rtf", 3, "C:\My Documents"
```

This second instruction reactivates the specified application (it must be running) and file by using the same *applicationName* value.

```
AppBringUp "Readme"
```

Example 2

This example copies text from Dragon NaturallySpeaking to a different application when you say “Copy All to Word” or “Copy All to WordPad”, for example.

```
SendKeys "{Ctrl+a}{Ctrl+c}"
if arg1 = "Word" then AppBringUp "WinWord"
if arg1 = "WordPad" then AppBringUp "WordPad"
SendKeys "{Ctrl+v}"
SendSystemKeys "{Alt+Tab}"
```

See Also

AppSwapWith command
ShellExecute command

AppSwapWith

Syntax

`AppSwapWith "`*`applicationName`*`"`

Description

Switches the currently active application with *applicationName*.

1. If *applicationName* was used in a previous AppBringUp or AppSwapWith command and a copy of the application is running, the command tries to make that copy the active application. If AppSwapWith cannot activate it, the command starts a second copy.
2. Otherwise, If the *applicationName* is the executable name of an unnamed instance of a program that is already loaded, Dragon NaturallySpeaking brings up the instance and assigns it *applicationName*.
3. Otherwise, Dragon NaturallySpeaking runs any executable named *applicationName* that it locates in your PATH, the Windows directory, the Windows/System directory, the Program Files/Accessories directory, or the *NatSpeak/*Program directory.

Argument

applicationName	Specifies the name of the application to run. Do not include .EXE in the name.

Example

The following command swaps the currently active application with WordPad:

```
AppSwapWith "WordPad"
```

See Also

AppBringUp command

Beep

Syntax

```
Beep
```

Description

Generates the Windows default beep sound by calling the Windows API MessageBeep() function. The exact result of this call depends on both the operating system and the system configuration. On some systems with half-duplex sound systems, you must turn off the microphone before using this command for users to hear the beep.

Example

The following script turns off the microphone, plays the beep sound, waits for ½ second to ensure that the sound finishes playing, and turns the microphone back on:

```
SetMicrophone 0
Beep
Wait 500
SetMicrophone 1
```

See Also

PlaySound command

ButtonClick

Syntax

`ButtonClick [`*`button`*`[, `*`click`*`]]`

Description

Clicks the specified mouse button at the current mouse location. The command generates a single or double click. You can generate triple or quadruple clicks by using two ButtonClick commands in a row.

The Dragon NaturallySpeaking global Mouse <MouseAction> command uses this scripting command.

Arguments

button A number that specifies which mouse button(s) to click, as follows:

1	Left button (default)
2	Right button
4	Middle button

Add the above numbers to click multiple buttons simultaneously.

click A number that specifies the number of clicks, as follows:

1	Single click (default)
2	Double click

Example

The following command double-clicks the left mouse button:

```
ButtonClick 1,2
```

See Also

MouseGrid command
RememberPoint command
SetMousePosition command

ClearDesktop

Syntax	`ClearDesktop`
Description	Minimizes all open applications.
See Also	AppBringUp command

ControlPick

Syntax

`ControlPick "`*`caption`*`"`

Description

Selects the control. Command buttons, option buttons, check boxes, group boxes, list boxes, and text fields are examples of controls. Only controls within the currently active window are available.

Generally, toolbar buttons, property sheet tabs, and taskbar buttons cannot be selected using ControlPick.

Some Windows applications do not always use standard controls to draw a seemingly ordinary prompt, push button, and so on. When this is the case, Dragon NaturallySpeaking cannot find a control that matches the caption, and ControlPick fails. If this happens, you may be able to pick the control by using the SendKeys command to press an accelerator key.

Argument

caption	Identifies the selected control. The required value depends on the control and application. For many controls, the caption is the control’s label (for example, the text on a button). This argument is case sensitive. If the label includes an ampersand (&) character to identify the control’s accelerator key, do not include the ampersand in this argument. If *caption* matches a “static” control, such as a prompt or a list box, the control must have an accelerator key. (The accelerator key is identified by an underlined character in the caption.)

Note

The ControlPick command does not work with some Microsoft Office 97 controls. If a control has an accelerator key, you can use the SendKeys command to send the keystroke that selects the control.

Example

The following command presses the OK button in the currently active dialog box:

```
ControlPick "OK"
```

See Also

ButtonClick command
MenuPick command

DdeExecute

Syntax

```
DdeExecute "app", "topic", "commandString"[,
async]
```

Description

Sends a command or series of commands to the specified dynamic data exchange (DDE) application. See the application’s documentation to find the application name and topics to which the application responds.

Arguments

app	Specifies the application to which you are sending a DDE command string. This value is normally the name of the application’s .EXE file without the file name extension. The application must be running.
topic	Specifies the DDE topic. Many DDE applications recognize a topic named “System”, which is always available. Each open document may also be a separate topic.
commandString	Specifies the command or series of commands to send to the DDE application. Most applications require that each command be enclosed in square brackets (for example, “[commandString]”). Do not include spaces between bracketed commands.
async	A number that indicates a synchronous or asynchronous call, as follows: 0 Synchronous call (the default). DdeExecute waits until the DDE application acknowledges completion or a timeout occurs. 1 Asynchronous call. DdeExecute returns immediately without waiting for the application to acknowledge completion.

Set *async* to 1 if you are calling a DDE command that does not return immediately. For example, you may request a command that opens a dialog box and does not return until you close the dialog box. If you set *async* to 0 (or omit the argument) and the DDE command does not return quickly, DdeExecute fails with a timeout error and Dragon NaturallySpeaking displays a message box indicating a DdeExecute error.

Example

The following command tells Microsoft Excel to open the file Amortize.xls. Note that the file name must be enclosed in two sets of quotation marks:

```
DdeExecute "excel", "system",
"[open(""amortize.xls"")]"
```

See Also

DdePoke command
SendKeys command
“Using Dynamic Data Exchange” section on page 31

DdePoke

Syntax

`DdePoke "`*`app`*`", "`*`topic`*`", "`*`item`*`", "`*`value`*`"`

Description

Sets the value of a specific item in a dynamic data exchange (DDE) application. The application name, topic, and item name depend on the application. See the documentation for that application.

Arguments

app	The name of the DDE application to which you are sending a DDE poke command. The application must be running.
topic	Application topic to which you are sending a poke command.
item	The name of the item that you are changing.
value	The new value for the item.

Example

The following command sets to 10 the value of row 1, column 2 in the worksheet “sheet1” in Microsoft Excel:

```
DdePoke "excel", "sheet1", "r1c2", "10"
```

See Also

DdeExecute command

DllCall

Syntax

`DllCall "`*`lib`*`", "`*`function`*`", `*`"stringArg"`*

Description

Calls a function in an application’s dynamic link library (DLL). The function must be a PASCAL function that takes a single LPCSTR (long pointer to a constant string) argument and has no return value. C or C++ functions must have prototypes equivalent to:

```
extern "C" __declspec(dllexport) void __stdcall
fn(LPCTSTR szParam );
```

This command is useful for adding your own extensions to the Dragon NaturallySpeaking scripting language. The DllCall Script command works even if the DLL file name extension is not “.dll”.

Arguments

lib	The name of the DLL file you are calling. The file extension does not have to be .dll, and you must include the extension name in the argument.
function	Specifies either a string containing the function name or the ordinal number of the entry point in the DLL. If you are using an ordinal number, do not enclose it in quotation marks. This argument must be a constant or a variable. It cannot be an expression.
stringArg	Use this argument to pass information to your function. The *stringArg* argument has no meaning to Dragon NaturallySpeaking. This argument can be a constant, variable, or expression.

Notes

Do not use this command to call Windows system DLLs. However, you can write a DLL with functions that convert the contents of the *stringArg* from a single string into the appropriate arguments for a Windows DLL call and then make the required Windows calls.

This command loads and releases the DLL each time you use it. If you make frequent DLLCalls to the same DLL, you can have another process load the DLL (and not unload it) to ensure that the DLL stays in memory.

Example

The following command is equivalent to a PressButton(“bold”) C function call, where the PressButton function is defined as void far PASCAL PressButton (LPCSTR).

```
DllCall "wpx.dll", "PressButton", "bold"
```

See Also

RunScriptFile command

DragToPoint

Syntax

`DragToPoint [`*`buttons`*`]`

Description

Performs a mouse drag. The drag starts at the location specified by the last RememberPoint command and ends at the current mouse pointer location. If you have not set the RememberPoint, an error message appears. Shift keys that are in effect before this operation stay in effect during the operation. The command does not work with applications that do not support mouse dragging.

The Dragon NaturallySpeaking <MouseAction> command uses this scripting command.

Argument

buttons A number that specifies which buttons to press and hold while the drag takes effect:

1	Left mouse button (default)
2	Right mouse button
4	Middle mouse button

You can add the numbers together to press more than one button at a time.

Example

The following command presses the left and right mouse buttons and drags the mouse from the location set by the last RememberPoint command to the current mouse pointer location:

```
DragToPoint 3
```

See Also

ButtonClick command
MouseGrid command
RememberPoint command
SetMousePosition

GoToSleep

Syntax

`GoToSleep`

Description

Puts Dragon NaturallySpeaking in sleep mode. In this mode NaturallySpeaking normally recognizes only the words in the Asleep state of the Global Commands Menu (which typically includes only the Wake Up command).

The Dragon NaturallySpeaking Go to Sleep command uses this scripting command.

See Also

WakeUp command

HeardWord

Syntax

`HeardWord "`*`wordName`*`"[, "`*`wordName`*`"]...`

Description

Causes the action associated with the *wordName* arguments to occur as if you said these words to Dragon NaturallySpeaking as a phrase. Dragon NaturallySpeaking resumes normal execution after the *wordName* action completes.

Argument

wordName	Specifies the word to recognize, as if you were saying it to Dragon NaturallySpeaking. This argument is case sensitive. You can specify up to eight *wordName* arguments, separated by commas. Each *wordName* must be a single word or list item (which can be more than one word) in the active vocabulary.

Notes

If any of the words specified in the command are not in the vocabulary, Dragon NaturallySpeaking displays an error message.

If a word in a list has different written and spoken forms, use the entire list entry, with both forms for the *wordName* argument. (You can use either the written form or the written\spoken form for dictation words.)

Use a single backslash before dictation commands that do not have a written form. For example, use HeardWord “\Caps”. To see a complete list of the words that require this treatment, open the Vocabulary Editor and scroll up. (The editor does not show the top of the list when it opens.) The words at the top of the list that show a spoken form only require a leading \ character. Note that Dragon NaturallySpeaking treats these words as dictation words, not as commands. As a result they are only available during dictation. They do not work, for example, if you press the Ctrl key to force recognition of words as commands.

Some dictation vocabulary words, such as “Academy Awards”, consist of more than one dictionary word. In these cases, put the complete vocabulary word in a single argument.

Examples

The following example types a closing remark in Times New Roman font at the end of a Dragon NaturallySpeaking document. Note that “Times New Roman” is a single argument to the HeardWord command because it is a list entry. Also, the Wait 100 command ensures that the font gets set before Dragon NaturallySpeaking sends the keystrokes.

```
HeardWord "Go", "to", "Bottom"
HeardWord "Set", "Font", "Times New Roman"
Wait 100
SendKeys "{Enter 2}" + "This document was dictated
using Dragon NaturallySpeaking."
```

The following code from the NaturallySpeaking Global.dvc file enables users to say commands such as “Scratch that 5 times” to delete the last 5 utterances. (An utterance is one or more words spoken as a single unit, without pauses.)

```
COMMAND "Scratch That <2To10> Times" {
            SCRIPT {
                loop& = Val(_arg1)
                while ( loop& )
                HeardWord "Scratch","That"
                loop& = loop& - 1
                wend
            }
        }
```

See Also

“Using HeardWord in Commands with Lists” section on page 30.

MenuCancel

Syntax	`MenuCancel`
Description	Cancels the current menu and all parent menus used to reach it.
See Also	MenuPick command

MenuPick

Syntax

`MenuPick "menuItem"`

Description

Finds a currently visible menu or menu item labeled *menuItem* in the active window and outputs the keystroke sequence to select it.

Argument

menuItem	Specifies the desired menu or menu item. This argument is case sensitive. If the menu name contains an ellipsis (…), do not include it in the menuItem argument.

Notes

The MenuPick command does not work with Microsoft Office 97 and Microsoft Windows 98 menus. You can use the SendKeys command to send keystrokes to select these menus.

You may need to insert a Wait command between consecutive MenuPick commands to ensure that the first menu appears before the next MenuPick command tries to select one of its entries.

Example

The following script displays the current application’s File menu and selects its Save As item.

```
MenuPick "File"
MenuPick "Save As"
```

See Also

MenuCancel command
ControlPick command

MouseGrid

Syntax `MouseGrid [`*`state`*`[, `*`gridbox`*`]]`

Description Displays or shrinks the MouseGrid utility for positioning the mouse pointer on the screen or active window.

Arguments

state MouseGrid state:

0 Turns MouseGrid off or leaves it off.

1 (Default) If the grid is off and there is no *gridbox* argument, fills the whole screen with the grid. If there is a *gridbox* argument, the command also shrinks the grid relative to the whole screen.

If the grid is on, the command shrinks the current grid as specified by *gridbox*. If there is no *gridbox* argument, the command:

- Expands the grid to its previous size
- Expands a grid that covers a window to cover the screen
- Leaves a grid that already covers the full screen

2 If the grid is off and there is no *gridbox* argument, the command fills the active window with the grid. If there is a *gridbox* argument, the command also shrinks the grid relative to the active window.

If the grid is on, the command shrinks the current grid as specified by *gridbox*. If there is no *gridbox* argument, the command expands the grid to its previous size.

gridbox Indicates how the grid shrinks. If *gridbox* is 0, the command undoes the most recent shrinkage. If *gridbox* is 1 through 9, it shrinks the grid in the indicated direction based on:

1	2	3
4	5	6
7	8	9

When the grid gets so small that it cannot display properly if it gets smaller, it no longer shrinks. Instead, it moves in the indicated direction.

Example

If MouseGrid is off, the following command puts a grid in the upper-left 1/9th of the active window. If the MouseGrid is on, it shrinks the grid so that it covers only the upper-left 1/9th of its initial area.

```
MouseGrid 2, 1
```

See Also

ButtonClick command
RememberPoint command
SetMousePosition command

MsgBoxConfirm

Syntax

`MsgBoxConfirm "msg", type, "title"`

Description

Displays a standard Windows message dialog box with the appearance defined by the arguments.

When the message box displays two buttons and you choose the second, the script containing the MsgBoxConfirm command terminates.

Arguments

msg — Specifies the message to display in the dialog box. The message is truncated after the 1024th character. Messages longer than 255 characters with no intervening spaces are truncated after the 255th character. Messages automatically wrap from line to line to fit the box width.

type — A number that specifies the buttons and icons to display in the message box. This number is the sum of your choices from each of the following three sections.

1. Select one number from this list:

0	Displays the OK button
1	Displays the OK and Cancel buttons
4	Displays the Yes and No buttons
5	Displays the Retry and Cancel buttons

2. Add it to one number from this list:

16	Displays the Critical Message icon (a stop sign)
32	Displays the Warning Query icon (a question mark). The Microsoft *Windows Interface Guidelines for Software Design* discourages using this icon.
48	Displays the Warning Message icon (an exclamation mark)
64	Displays the Information Message icon (a lowercase i)

3. Add that to one number from this list:

0	First button is the default
256	Second button is the default

title	Specifies the caption for the dialog box

Note

Dragon NaturallySpeaking editions through 3.01 do not support using the second button as a default.

Example

The following script displays a message box with Yes and No buttons. The default is Yes, and there is a Warning query (?) icon. If you answer Yes or press Enter the script sends the {Ctrl+s} key sequence. If you choose No it terminates with no action.

To calculate the *type* value, add:

	4	Displays the Yes and No buttons
+	48	Displays the Warning Message icon
+	0	Sets the first button as the default
	52	Total

The script is:

```
MsgBoxConfirm "Keep your results?", 52, "Keep Results?"
SendKeys "{Ctrl+s}"
```

PlaySound

Syntax

`PlaySound "`*`waveFile`*`"`

Description

Plays the *waveFile*. This command turns the microphone off while the file plays and then restores the microphone to its previous state. The *waveFile* completes before continuing to the next command.

Argument

waveFile	Specifies the sound input file. If you do not specify an absolute path name, the command searches for the file in the current, Windows, Windows system, and Dragon NaturallySpeaking directories and all directories in your PATH environment variable. You do not have to specify the .wav file extension.

Example

The following command plays the standard Windows 95 “Tada” sound:

```
PlaySound "C:\Windows\Media\Tada.wav"
```

See Also

Beep command

RememberPoint

Syntax

```
RememberPoint
```

Description

Records the current position of the mouse pointer. Use this command to specify the starting point of a DragToPoint mouse drag. The remembered point is reset to the upper left corner of the screen when Dragon NaturallySpeaking restarts.

The global Mouse <MouseAction> voice command uses this command when you say “Mouse Mark”.

See Also

ButtonClick command
DragToPoint command
MouseGrid command
SetMousePosition command

RunScriptFile

Syntax

```
RunScriptFile "fileName"
```

Description

Runs a Dragon NaturallySpeaking script contained in the specified file. The file must be in ANSI text format and must contain only valid Dragon NaturallySpeaking scripting language commands. You can nest RunScriptFile commands (that is, call one script file from another).

Argument

fileName	Specifies the name of an ANSI text file to read.

Note

Dragon NaturallySpeaking releases through Version 3.01 will only run a script file that is in the *NatSpeak*/Program directory or is specified by an absolute path name such as C:\Myapps\Myscript.txt.

Example

The following command runs the script contained in the Commands.txt file:

```
RunScriptFile "Commands.txt"
```

See Also

DllCall command
HeardWord command

SendKeys

Syntax

`SendKeys "`*`keystroke text`*`"`

Description

Sends keystrokes to the currently active window. Use this command for most key sequences that you include inside voice commands. Use the SendSystemKeys command in place of this command only where this command *does not* work, such as in sending the {ctrl+esc}, {alt+esc}, and {alt+tab} key sequences to Windows 95.

Arguments

keystroke text	Specifies which keystrokes to send to the currently active window. Send nonprinting keys by enclosing their names in braces ({}). To repeat a key, enclose the key name, a space, and the number of repetitions in braces. Specify a shift-key combination by enclosing the modifier key name (such as Alt), a plus sign (+), and the shifted key in braces, without any spaces.

Notes

Do not include spaces between the names of keys pressed simultaneously, such as Alt+Tab.

Any modifier keys (Shift, Ctrl, Alt) pressed while the SendKeys command executes do not affect the command action.

You should test the use of SendKeys under varying conditions to avoid unpredictable results.

Use a DdeExecute instruction, rather than SendKeys, if you cannot be sure a key sequence always produces the same result.

In Dragon NaturallySpeaking 3.*x* you can use 0 to indicate zero repetitions of a key. This feature is useful for scripts where a keystroke may be used 0 or more times, depending upon a variable value. This feature is not supported in Dragon NaturallySpeaking 2.*x*.

Examples

The following command sends three Ctrl+Right keystrokes to the application. In most word-processing applications, this moves the text insertion point right by three words:

```
SendKeys "{Ctrl+Right 3}"
```

The Dragon NaturallySpeaking word processor Set Font <FontFace> command includes the following scripting command. It sends the keystrokes needed to bring up the font change dialog box, followed by the contents of the command's _arg1 variable (the name of the typeface), followed by the Enter key. The + signs preceding and following the variable name concatenate the three parts of the argument into a single text string.

```
SendKeys "{Alt+o}f{Alt+f}"+arg_1+"{Enter}"
```

See Also

SendSystemKeys command
DdeExecute command
Key Name List on page 116
"Key Names in Voice Commands" section on page 13
"Using Dynamic Data Exchange" section on page 31

SendSystemKeys

Syntax

`SendSystemKeys "`*`keystroke text`*`"[, `*`speed`*`]`

Description

Sends keystrokes such as {ctrl+esc} to the Windows operating system. Only use this command for short system key sequences where the SendKeys command *does not* work. It is particularly useful for sending the {Ctrl+Esc}, {Alt+Esc}, and {Alt+Tab} key sequences and hot keys (such as the key to bring up the Correction dialog box).

The keyboard Alt, Ctrl, and Shift keys affect this command. For example, if you press the Shift key and run “SendSystemKeys "{Alt+Esc}"”, Windows gets a Shift Alt Esc key sequence. The command sends keystrokes more slowly than the SendKeys command. Also, if the system responds slowly, it is possible to overflow the system keyboard buffer by using a long string with this call.

Arguments

keystroke text	Specifies the system keystroke combination to send to Windows, in standard keystroke string format. Do not include spaces between the names of keys pressed simultaneously.
speed	Specifies the speed at which to generate the keystroke events as a percentage of the default speed. This argument must be a positive integer in the range 1 through 32767, where 100 is the default speed, 50 is half the default speed, and so on. In general, you should only use this argument if you have trouble getting the keystrokes to work correctly with slow systems and applications. You can also speed the keystrokes by setting this argument above 100, if experience proves it works reliably.

Example

The following command sends the Alt+Esc key combination to the Windows system:

```
SendSystemKeys "{alt+esc}"
```

See Also

SendKeys command
“Key Names in Voice Commands” section on page 13

SetMicrophone

Syntax

`SetMicrophone [`*`onOff`*`]`

Description

Turns the microphone on or off.

The Dragon NaturallySpeaking Microphone Off command uses this scripting command.

Arguments

onOff Specifies whether the microphone is on or off, as follows:

0	Off
nonzero	On

If you do not specify *onOff*, the microphone setting toggles.

Example

The following command turns on the microphone:

```
SetMicrophone 1
```

See Also

GoToSleep command
WakeUp command

SetMousePosition

Syntax

```
SetMousePosition relativeTo, xPos[, yPos]
```

Description

Positions the mouse pointer using absolute or relative coordinates. This command lets you put the pointer on items on the screen that are always at the same location and do not have accelerator keys. You can follow the SetMousePosition command with a ButtonClick 1,1 command to single-click the item.

The Dragon NaturallySpeaking global Mouse <Direction> <1To10> command uses this scripting command.

Arguments

relativeTo Indicates the meaning of the *xPos* and *yPos* coordinate arguments as follows:

0	Coordinates are relative to the top left of the screen.
1	Coordinates are relative to the top left of the active window.
2	Coordinates are relative to the current pointer position.
3	*xPos* is a map position relative to the screen.
4	*xPos* is a map position relative to the active window.
5	Coordinates are relative to the upper left corner of the client area of the active window. (The client area is the interior of the window that does not include the title bar, menu, and borders.)
6	*xPos* is a map position relative to the client area of the active window.

xPos, yPos The position of the mouse.

If *relativeTo* is 0, 1, 2, or 5, *xPos* and *yPos* specify a positive or negative number of pixels, with *xPos* increasing to the right and *yPos* values increasing downward.

If *relativeTo* is 3, 4, or 6, *xPos* must be an integer from 0 through 8. Do not specify a *yPos* value, or the command will cause an error message. The command puts the pointer in the location that corresponds to the number on the following map:

8	1	5
4	0	2
7	3	6

If *relativeTo* is 3, the command puts the pointer just inside the indicated screen corner or edge. The pointer is set back from the edge so it remains visible. Therefore, if *xPos* is 5, SetMousePosition puts the pointer just inside the upper right corner of the screen.

If *relativeTo* is 4, the command puts the pointer over the indicated edge of the active window, usually on a sizing border. Therefore, if *xPos* is 3, the command puts the pointer over the middle of the lower edge of the active window.

If *relativeTo* is 6, the command puts the pointer over the indicated edge of the client area of the active window.

Notes

If *relativeTo* is 0, 1, 2, or 5, the position is in pixels. You must know the screen resolution to be sure of the command results. For example, if the screen resolution is 640x480, and you specify coordinates of 680x200, the coordinates are off screen.

If the requested coordinates are off screen, and *relativeTo* is 0, 1, or 5, the system reports an error. In other cases, the command puts the pointer as close to the requested position as the screen allows.

Example

The following command positions the mouse pointer 300 pixels to the right and 200 pixels down from the upper left corner of the active window:

```
SetMousePosition 1, 300, 200
```

See Also

ButtonClick command
MouseGrid command

ShellExecute

Syntax

`ShellExecute "`*`commandLine`*`"[, `*`windowStyle`*`[, "`*`directory`*`" ["`*`application name`*`"]]]`

Description

Loads an application. If a copy of the application is already running, it loads a new instance of the application.

Arguments

commandLine	Specifies the executable file to run and any command line arguments. You do not have to include the application's .exe extension. If *commandLine* does not specify a full file path, the application must be in the current, Windows, Windows\System, or *NatSpeak*\Program directory, or in a directory that is listed in the PATH variable. You cannot include spaces in the executable path. If any directory name includes spaces, use its short name.
windowStyle	Controls how the window appears when you activate the application. Must be one of the following:

1, 5, or 9	Normal (with focus, default)
2	Minimized (with focus)
3	Maximized (with focus)
4 or 8	Normal (without focus)
6 or 7	Minimized (without focus)

directory	Assigns a working directory for the application. The default is the directory that contains the executable file.
applicationName	Lets you name the application. This name is used by the AppBringUp and AppSwapWith commands. The application name must be unique. ShellExecute reports an error if the name is already in use.

Note

In Dragon NaturallySpeaking releases through Version 3.01 the *commandLine* argument must contain only the executable name or path.

If you include any command line arguments, Dragon NaturallySpeaking reports an error.

Example

The following command starts an instance of WordPad with its window maximized and sets the working directory to C:\DOCS. This instance of WordPad has the name “letter,” and you can now use the AppBringUp scripting command with the argument “letter” to activate the window while it is open.

```
ShellExecute "WordPad", 3, "c:\docs", "letter"
```

See Also

AppBringUp command

TTSPlayString

Syntax

`TTSPlayString ["`*`text string`*`"[, "`*`switches`*`"]]`

Description

Sends a request to the text-to-speech utility. If the microphone is on, this command turns it off during playback. You can stop the text-to-speech engine during playback by turning on the microphone by mouse or by keyboard.

Arguments

text string	The text string to be read back by the text-to-speech utility. If this string is empty or not specified, the command uses the contents of the Windows clipboard.
switches	A series of values specifying the text-to-speech output characteristics. Switches are optional; if you omit a switch, Dragon NaturallySpeaking uses the default text-to-speech engine settings specified on the Options dialog box Text-to-Speech tab. Switches can be used in any order. The valid switches are:

/sXXX	Sets the playback speed in arbitrary units in the range 0–255. The default is 127.
/pXXX	Sets the voice pitch on a low-to-high scale. The value is in arbitrary units in the range 0–255. The default is 127.
/lXX	Sets the playback volume. This setting works in relation to the speaker volume set by the Audio Setup wizard. The value must be in the range 0–255. The default is 255.
/e	Tells the text-to-speech utility to ignore e-mail headers and other common e-mail separators (like a line of angle brackets (>) during playback.

Note

You can optionally install text-to-speech on Dragon NaturallySpeaking Deluxe and Professional editions. If you run this command on an edition that does not have text-to-speech installed, Dragon NaturallySpeaking displays an error message. This message includes a check box that lets you tell Dragon NaturallySpeaking not to display the message in the future.

Examples

The following command uses the default settings of the text-to-speech engine to read the string:

```
TTSPlayString "Click the microphone button to start."
```

The following command tells Dragon NaturallySpeaking to read the string at a moderately slow pace with a lower-pitched voice and a volume of about 2/3 of the current setting:

```
TTSPlayString "Your journey is about to begin.",
"/s100 /p50 /l170"
```

Wait

Syntax

`Wait` *`milliseconds`*

Description

Causes Dragon NaturallySpeaking to wait the specified time before it runs the next script command (or exits the script, if Wait is the last or only command in the script). The Wait command disables all user input while it executes. You can use the Wait command to ensure that an application has enough time to load before the script continues.

Note

In many cases, this command stops recognition. However, it does enable recognition as appropriate, for example, when the previous command is a MsgBoxConfirm command (to allow spoken responses to the message box).

Argument

milliseconds	Specifies how long, in milliseconds, to pause before running the next script command or exiting. This number must be in the range 0–32767. If you need to specify a wait longer than 32 seconds, use a loop to repeat the Wait command.

Example

The following command causes the script to pause100 milliseconds (1/10 of a second) before it executes the next command:

```
Wait 100
```

WakeUp

Syntax `WakeUp`

Description Causes Dragon NaturallySpeaking to exit sleep mode and resume normal recognition.

The Dragon NaturallySpeaking Wake Up command uses this scripting command.

See Also GoToSleep command

WinHelp

Syntax

`WinHelp "helpFile", cmd[, topic]`

Description

Activates the Windows Help application and specified Help (.HLP) file, and then displays the specified window or topic.

The Dragon NaturallySpeaking What Can I Say and Give Me Help commands use this scripting command.

Arguments

helpFile The name of the Help (.HLP) file to activate. If no path is specified, Dragon NaturallySpeaking looks for the file in the Natspeak\Help directory. The Help file name can be followed by a window identifier in the following form:

>*WindowName*

cmd A value telling the Help engine what to do, as follows:

1 Display the Help topic identified by the topic argument value.

2 Close the Windows Help application.

3 Display the default Help Contents topic. This topic is defined either in the Help project (.hpj) file or by a previous WinHelp command with a *cmd* value of 5. If there is no Contents topic, WinHelp displays the first topic in the Help file.

4 Display Help for Windows Help.

5 Set the specified *topic* as the Help Contents topic. Does not display the Help file.

8 Display in a popup window the Help topic identified by the topic argument value.

9 Switch to the specified Help file. Open it with the first topic if the file is not already open. (This is useful if another Help file has the focus.)

11 Display the Help Topics dialog box. The tab used most recently is displayed.

topic	Identifies the Help topic to use. This value must be the decimal number value of a Help context identifier defined in the [MAP] section of the .HPJ file (that is, the numeric value assigned to a specific topic ID). This value is required if the *cmd* value is 1, 5, or 8.

Example

This instruction opens Dragon.hlp in a window named “main” and displays the topic with the context ID 1033096 (Hex 207E8). (In Dragon NaturallySpeaking 3.0 this number corresponds to the “Selecting the type of command action” topic.)

```
WinHelp "C:\Natspeak\Help\Dragon.hlp>main", 1,
1033096
```

Key Name List

Dragon NaturallySpeaking supports both the base keys supported on all keyboards and the extended keypad keys that may not be available on some systems.

Base Keys

All PC keyboards have the following keys:

Esc	Tab	Num Key 0
F1	Caps Lock	Num Key.
F2	Shift *or* LeftShift	NumKey* *or* Asterisk *or* Mult
F3	Ctrl *or* LeftCtrl	NumKey– *or* Minus
F4	Alt *or* LeftAlt	NumKey+ *or* Plus
F5	Space	Home
F6	BackSpace	Up
F7	Enter	PgUp
F8	RightShift	Left
F9	NumKey7	Center
F10	NumKey8	Right
F11	NumKey9	End
F12	NumKey4	Down
SysReq	NumKey5	PgDn
PrtSc	NumKey6	Ins
ScrollLock	NumKey1	Del
Pause	NumKey2	
Break	NumKey3	

Extended Keypad Keys

The 101-key “Extended” keyboards used on most desktop computers have the following additional keys.

On the right side of the main keyboard:

RightAlt
RightCtrl

Between the main keyboard and the numeric keypad:

ExtIns	ExtHome	ExtPgUp
ExtDel	ExtEnd	ExtPgDn
	ExtUp	
ExtLeft	ExtDown	ExtRight

On the numeric keypad:

NumLock
NumKey/ *or* Slash
NumKeyEnter *or* KeyPadEnter

Notes

Always precede the Break key with Ctrl.

The NumKey0–9 and NumKey. entries send the codes generated by pressing the numeric pad keys with NumLock on. The Home, Up, PgUp, and other such key names send the codes generated by pressing the numeric pad keys with NumLock off.

Center is the non-NumLock version of NumKey5.

See Also

“Key Name in Voice Commands” section on page 13.

Glossary

A

accelerator key	A keyboard key or key combination that invokes a particular application command, such as Ctrl+s to save a file.
acoustic model	A numeric representation characterizing the basic sound units of a **word** or **phrase**.
active application	The application that currently has the Windows focus. The active application receives any keystrokes. See **active window**.
active vocabulary	The set of **words** that an application recognizes at any one time. See **resident vocabulary**, **total vocabulary**.
active window	The window in which you are currently working or directing input. Its title bar is highlighted.
adaptation	The process of improving existing speech **models** to better reflect your speech. Dragon NaturallySpeaking adapts models when you use the Correction dialog box to correct a **recognition** error. See **training**.
alpha-bravo spelling	Using the words in the **International Communication Alphabet** to represent letters and spell words.
ANSI character set	The set of 8-bit western-language characters defined by the American National Standards Institute. Also referred to as the ISO 8859-1 or Latin-1 character set. The ANSI character set is a superset of the 7-bit ASCII character set. For example, ANSI (decimal) code 153, the Trademark symbol (™), is not in the ASCII character set.

B

backup dictionary	A **dictionary** containing the **total vocabulary**, with the exception of any user-defined **words**. This dictionary resides on disk, and its words are used only during error correction and when adding new words and **phrases**. The Dragon NaturallySpeaking backup dictionary contains over 230,00 words.
built-in command	A standard Dragon NaturallySpeaking **command** as opposed to user-written **voice commands**. A built-in command can be either a **dictation command** or a **voice command** written by Dragon Systems.

C

caret	See **insertion point**.
click	To position the **pointer** over an object and then press and release a **mouse** button. The act of clicking. See **press**.
command	An **utterance** (**word** or **phrase**) that causes Dragon NaturallySpeaking to take some action other than type the spoken text. “Command” is normally used for **built-in commands,** including **dictation commands**, while “voice command” is used for user-written **voice command**s.
command and control	The use of speech to control a computer application, the computer system environment, or equipment. Contrast with **dictation**.
confusable words	**Words** that sound alike (or very similar) to a speech-recognition system. Shorter words are generally more confusable than longer words.
continuous speech	**Words** spoken fluently and rapidly without pauses between individual words, as in conversational speech. Contrast with **discrete speech**.
continuous speech recognition	Recognition of **continuous speech**. Recognition where the **recognizer** treats each **utterance** as a stream and breaks it into words recognized individually.
control	A Microsoft Windows object that enables user interaction or input, often to initiate an action, display information, or set values.

cursor	A generic term used for the indicator on the computer screen that shows where user interaction occurs. Often used for the **insertion point** or **pointer**.

D

DDE	See **Dynamic Data Exchange**.
dialog box	A secondary window that gathers additional information from a user.
dictation	Using speech to enter text into a document.
dictation command	A **command** (such as Caps On) that you can say without pause during dictation. Dictation commands are built into the Dragon NaturallySpeaking system and cannot be added. Contrast with **voice command**.
dictation word	A **word** that Dragon NaturallySpeaking enters directly into an application. Contrast with **command**.
dictionary	The **total vocabulary** of a speech-**recognition** system. The file or files that contain the spellings and phonetic information for all the words that the system can recognize. See **backup dictionary**.
discrete speech	Speech where you must pause between **words**. Contrast with **continuous speech**.
Dynamic Link Library (DLL)	A set of routines that can be dynamically linked with an application program.
.DVC file	A file that defines Dragon NaturallySpeaking voice commands.
Dynamic Data Exchange (DDE)	A protocol for communicating between Windows applications.

E

enrollment	An initial process intended specifically for making representative speaker-dependent **models** for a **recognizer**. Also called **training**. See **adaptation**.
environmental sounds	Extraneous sounds that may be confused with speech. Environmental sounds include noises, coughing, telephone rings, and other nonspeech sounds.

I

insertion point	The location where text or graphics are inserted (also referred to as the **cursor** or caret).
International Communications Alphabet (ICA)	Also called the alpha-bravo words. A set of **words** (such as “lima”) each representing a single letter of the English alphabet. The ICA is less likely to be recognized incorrectly than the sounds of the letter names.

K

keystroke command	A **voice command** that only generates keystrokes. Contrast with **script command**.

L

language model	Data about the use of **words** in a language, such as word-frequency information. Dragon NaturallySpeaking uses language model information to improve **recognition** accuracy.
literal	A word, symbol, or number that defines itself; contrast with variable, which represents some other value.

M

Macro	See **voice command**

menu	(1) The section of a Global.dvc file that defines the commands that apply to a particular application. There is also a Global commands menu. (2) A list of commands that you can choose in a Windows user interface. For example, most Windows programs have a File menu.
model	A description or representation of observed behavior. Dragon NaturallySpeaking has **acoustic models** for speech sounds and **language models** for word use.
mouse	An input (pointing) device with one or more buttons. The mouse typically moves the **pointer**.
MouseGrid	A mechanism in Dragon NaturallySpeaking that lets you position the **pointer** by speech.
mouse pointer	See **Pointer**

N

navigation	Moving between or within elements of a graphical user environment such as Microsoft Windows.

O

out-of-vocabulary word	A **word** not in the **active vocabulary**, so it cannot be recognized immediately.

P

phrase	A multiple-word **utterance**.
pointer	A graphic screen representation of the pointing device location. Often referred to as the mouse pointer or **cursor**. See **mouse**, **insertion point**.
press	To press and release a keyboard key. Also used for activating a visual representation of a button. See **click**.

pronunciation	A representation of how a **word** is pronounced. A word can have more than one pronunciation.

R

recognition	The process of identifying the contents of spoken material.
recognizer	A software component that takes speech as input and determines what is spoken.
resident vocabulary	The set of **words** that are in computer memory. The resident vocabulary can have user-defined words that are not in the **backup dictionary**. See **active vocabulary**, **total vocabulary**.

S

SAPI	Speech Application Programming Interface, a programming interface defined by Microsoft, Inc., for the interaction between application programs and speech recognition software.
script	A Dragon NaturallySpeaking scripting language program.
script command	A **voice command** that runs one or more **scripting commands** (a **script)**. Contrast with **keystroke command.**
scripting command	A **command** in the Dragon NaturallySpeaking scripting language. Scripting commands control Dragon NaturallySpeaking, applications, and the Windows environment.
scripting language	The programming language for Dragon NaturallySpeaking **voice commands**.
select	To identify one or more objects on which to perform an operation.
Short-cut key	See **accelerator key.**

sleep mode	A special **mode** in which Dragon NaturallySpeaking only responds to words in Asleep state in the Global Commands menu, which typically contains only the Wake Up command.
speech-aware application	An application specifically designed to respond to speech. Dragon NaturallySpeaking can work with almost any application that is not itself speech-aware.
state	(1) The condition of a system, such as an executing computer program, with regard to phase, form, or structure. (2) The section of a Global.dvc file that defines the commands that can be recognized when a particular window is active. (3) The condition of a speech recognizer that corresponds to the state(s) from which it can recognize. Synonymous with **mode**.
string	(1) A sequence of spoken **word**s or **phrases**, such as a zip code or a telephone number, intended as single, useful input to a **recognizer**. (2) A one-dimensional array of characters delimited by quotation marks (") in the Dragon NaturallySpeaking **voice command** language.
syntax	The structure of **strings** (sentences) in a language. The rules defining the valid sequence of elements in a language.

T

total vocabulary	The set of all **words** that a **recognizer** can understand, usually used to mean the **resident vocabulary** plus all words in the **backup dictionary**. See **active vocabulary**.
training	The process of making representative **acoustic models** for a **recognizer**. You typically go through a training procedure when you first use a speech-recognition product. Also called **enrollment.** See **adaptation**.

U

user file (USR)	A file that contains the **acoustic model** information for a particular user.
utterance	One or more **words** spoken as a single unit, without pauses.

V

vocabulary	**Words** and **phrases** that the **recognizer** can understand. See **resident vocabulary**, **active vocabulary**, **total vocabulary**.
vocabulary file (VOC)	A Dragon NaturallySpeaking file that defines a **vocabulary**. The VOC file includes word spellings, pronunciations, and other information.
voice command	A Dragon NaturallySpeaking **command** consisting of a **voice command name** and a **voice command action**. Types of voice commands include **built-in command**, **keystroke command**, and **script command**. Contrast with **dictation command**.
voice command action	The action that results when you say a **voice command name**. Defined by a keystroke sequence or one or more scripting language commands.
voice command name	The c**ommand** that you say to run a **voice command**.
voice command script	The scripting language commands that define a **voice command action**.

W

word	The basic unit of speech that a **recognizer** acknowledges. May be smaller or larger than a word in a language. For example, in Dragon NaturallySpeaking,“Wake Up” is a word. See **dictation word, dictation phrase, phrase.**

Index

CORPORATE HEADQUARTERS

Dragon Systems, Inc.
320 Nevada Street
Newton, Massachusetts 02460
USA

Tel: +1 (617) 965-5200
Fax: +1 (617) 527-0372
E-mail: info@dragonsys.com

www.naturalspeech.com

DRAGON SYSTEMS UK Ltd.

Dragon Systems UK Ltd.
Millbank, Stoke Road
Bishops Cleeve
Cheltenham, Glos. GL52 4RW
UK

Tel: +44 (0)1242 678575
Fax: +33 (0)1242 678301
E-mail: info@dragon.co.uk

DRAGON SYSTEMS GmbH

Dragon Systems GmbH
Messerschmittstrasse 4
D-80992 Munich
Germany

Tel: +49 (0) 89-1430-5062
Fax: +49 (0) 89-1430-5536
E-mail: deutsch@dragonsys.com