Talking Tools™

Computer Generated Speech for Your Apple® IIGS

Barbara Allred

Byte Works, Inc.
4700 Irving Blvd. NW, Suite 207
Albuquerque, NM 87114
(505) 898-8183

Master Set 1.0.1.1

Table of Contents

Chapter 1 – Introduction

Welcome to Talking Tools, a complete speech synthesizer for your Apple IIGS computer. In just a few minutes, your computer will be chattering away. You'll quickly learn everything you need to know about this easy-to-use software, from generating speech to controlling the way it sounds! Before diving in, please take some time to skim Chapter 1 and then use the demo program on your Talking Tools disk while you read Chapter 2.

Who Should Use Talking Tools

Talking Tools has been designed for computer programmers. You need not be an advanced programmer, since the package contains sample programs that will guide you through using Talking Tools, and which can be used as blueprints for your own programs. The sample programs and interface files for the speech tools are written in assembly, Pascal, and the C programming languages. It is expected that you are familiar enough with one of these languages that you can write simple text programs containing more than one subroutine, and that you understand all of your language's basic data types. In addition to the samples given on the disk, the manual also explains programming with the speech toolkit, and provides an in-depth reference to the toolkit's available functions.

To make the most out of your experiences with programming Talking Tools, you should probably own the Apple IIGS reference manuals needed to program the tools and the operating system. The sample programs show you how to start, call, and shut down the toolkit, but having the reference manuals on hand will enhance the discussion. The reading list at the end of this chapter gives the manuals you should have and how they can be purchased.

What You Should Have Received

The Talking Tools package consists of one 3.5-inch disk labeled Talking Tools System Disk; one 3.5-inch disk labeled Talking Tools Program Disk; a warranty registration card; and this manual. Be sure to return the registration card; it entitles you to receive software update information, as well as special product offers from the Byte Works. The registration card is also your proof of purchase. We cannot replace defective product components or send upgrades unless we receive it.

The Talking Tools System Disk contains the GS/OS operating system, Apple's tools, the Finder, and the SmoothTalker speech tools. Please notice that the Talking Tools demonstration programs require version 5.0 or greater of GS/OS in order to run, as well as the speech tool files. The Talking Tools System Disk that comes with Talking Tools has the version of the operating system and tools that it needs.

The Talking Tools Program Disk contains the interface files for the speech tools and sample programs. There may be a file named ReleaseNotes; you should always read this file to obtain important information not included in the manual. At the root level of the disk are the demo programs and any files that are common to the demos, such as Rez source files used in the desktop samples. The interface files and source code for the sample programs are contained in the folders named Pascal, Asm, and CLang, corresponding to the three programming languages Pascal, assembly, and C.

Back Up Your New Disks First!

Whenever you purchase new software, one of the first things you should do is to make back-up copies of the original disks. (Of course, sometimes this isn't possible when the disks are copy-protected.) The Talking Tools disks are not copy-protected, so make back-ups of the disks now. You can use whatever disk-copying utility you prefer to create the back-ups. You can also use the Talking Tools System Disk that came with Talking Tools to perform the back-up, since this disk boots into Apple's program launcher, the Finder.

Installing the Speech Tools

The Talking Tools software comes in two distinct parts. The first of these parts is the executable programs and demonstration source code that we will cover in this manual; these don't require any special treatment or installation. You can run the programs or compile the source on copies of the disks we sent, or you can move them to some other disk. The second part of the software is the speech tools themselves; like Apple's tools, these must be installed in the System/Tools folder of your system disk, or they cannot be used.

If you will be booting from the system disk we sent with the Talking Tools, you don't need to do any special installation – the speech tools are already installed on that disk.

To install the speech tools on another floppy disk or on a hard disk, run Apple's Installer; you can find a copy on the Talking Tools Program Disk. You will see a script in the left-hand list labeled "Talking Tools." Select that script. The next step is to move to the right-hand list item and select the tools folder where you want to install the speech tools. If you are installing the speech tools on your hard disk, select the SYSTEM folder on your boot partition, then select the TOOLS folder that you find inside of the SYSTEM folder. To set up a bootable floppy-disk, you would do the same thing, selecting the TOOLS folder from within the SYSTEM folder on your floppy disk. With the TOOLS folder and "Install Speech Tools" script selected, click on the Install button.

Required Hardware

You will need the following hardware to run Talking Tools' software:

- An Apple IIGS computer, or an Apple //e computer with an installed Apple IIGS upgrade.

- A minimum of 1 megabyte of RAM.

- At least one 3.5 inch disk drive.

The following hardware is highly recommended, especially if you intend to do multiple-language development or to develop large programs:

- With ROM version 01, 1.25M of RAM. With ROM version 03, 1.125M of RAM.

- A second disk drive. While a hard disk is ideal for serious programming, a 5.25-inch or second 3.25-inch disk to hold files while running your compiler on your 3.5-inch disk drive will work, too.

- A printer.

Required Software

There is no extra software that you'll need to use Talking Tools. To call the speech tools from a computer program, you'll need a language that can call the toolbox and produce stand-alone programs.

About This Manual

Chapter 2 provides a general introduction to the speech tools and should be read by all users of the package. Chapter 3 shows how to call the speech tools from a program, with Chapter 4 giving a complete call-by-call reference. Appendix A gives the complete source code listings for some of the samples on the disk. Appendix B explains how you can license the speech tools.

Visual Cues

In order to distinguish between information that this manual provides and characters that you type or characters that appear on your computer screen, special type faces are used. When you are to enter characters, the type face

```
looks like this.
```

The demo program's menu names, commands, and buttons and options within its dialog boxes and alerts

```
     look like this.
```

Additional Reading and Reference

Apple IIGS Toolbox Reference Volume I
Apple IIGS Toolbox Reference Volume II
Apple IIGS Toolbox Reference Volume III
These volumes provide essential information on how tools work; they are absolutely essential in mastering the source code for desktop programs on the disk.

Programmer's Reference for System 6.0
This colume covers changes to the tools sinse System Disk 5.0, plus new information about GS/OS and the Finder. This book is available from the Byte Works.

Human Interface Guidelines: The Apple Desktop Interface
Provides a complete reference for writing standard desktop applications on the Apple IIGS and Macintosh computers. Also contains important information for the users of desktop programs.

Apple IIGS Hardware Reference
Apple IIGS Firmware Reference
These manuals provide information on how the Apple IIGS works.

Programmer's Introduction to the Apple IIGS
Provides programming concepts for the Apple IIGS.

Technical Introduction to the Apple IIGS
A good basic reference source for the Apple IIGS.

GS/OS Reference
Documents the Apple IIGS disk operating system and program loader. This book contains all of the information you need to make calls to Apple's disk operating system.

GS/OS Reference, Volume 2, Beta Draft
For those who write code on the bare metal. Covers writing device drivers, interrupt handlers, signal processors.

Chapter 2 – Converting Text to Speech

In this chapter, we'll look at how the Talking Tools work, in a general way. We'll discuss how text is converted to speech, how text is converted to the phonetic language of the tool kit, and the kind of control you have over the processes. To guide you through the various aspects of the tool kit, we'll use a simple desktop program named SpeakIt, located on the Talking Tools disk.

How Do Computers Speak?

There are two main methods used to generate speech from computers. The first method works more or less like a tape recorder: you digitally record the text to be spoken, and then play the recording back. Obtaining the recording generally requires special hardware and software that is capable of sampling the speech at high frequencies in order to quantize the wave forms, volume, pitch, and other aspects of the speech. The quality of the speech produced with this method is very high. An obvious disadvantage of this method is that the computer only "speaks" pre-recorded messages – you cannot have it "say" arbitrary text on the fly. Another disadvantage is that digital recordings require massive amounts of space: using the recordings in an application means you'll need to pay special attention to memory requirements, as well as disk space. Yet one more disadvantage is that you'll need special equipment, such as a microphone, to record the original speech.

The other main method of generating speech, and the one used by Talking Tools, is to translate text into basic sounds, and then "speak" the sounds. The advantages to this method include the ability to "say" any text and to limit the amount of memory and disk space needed. The major disadvantage to this method is that the quality of the speech is not as high as that produced by a recording. The speech is still clear and understandable; coupled with the ability to customize the spoken words, as you can with Talking Tools, synthesized speech offers a very versatile way to make your computer talk.

The speech synthesizer included with Talking Tools was developed by First Byte, Inc. It represents years of intensive research to bring software-based speech to all major computer systems, at minimal cost to the consumer. You are free to use the speech tools in your own non-commercial programs; if you wish to use the software for commercial products, please consult the licensing information in Appendix B.

Getting Started

The Talking Tools package contains two disks, one labeled Talking Tools System Disk and the other named Talking Tools Program Disk. The system disk is used to boot your computer; it boots into the Finder, Apple's program launcher. Since the Finder is a desktop program, you can use the mouse to do just about everything: pull down menus, select menu items, open files and folders. You can use the Finder to initialize, rename, and copy disks; create folders; delete files;

and copy files and folders. In the instructions that follow, it is taken for granted that you know how to launch applications, initialize disks, create folders, and copy files.

Before launching our demo program, let's take a moment to look at the Talking Tools System Disk. In the SYSTEM folder is a folder named TOOLS that contains the RAM-based tools. Talking Tools' speech tool kit actually consists of four separate tool files, named Tool050, Tool051, Tool052, and Tool053. Tool052 contains the tool kit's front-end module, the parser that converts English text to phonetics. Tool050 and Tool051 are the tool kit's back-end modules, the adult male voice and adult female voice, respectively. Tool053 provides an interface to the latest operating system on the Apple IIGS. If you prefer using a system disk other than that which came with your Talking Tools package, be sure to copy these four tool files to the SYSTEM/TOOLS folder of your system disk. You can do this with Apple's Installer, as described in the installation instructions in Chapter 1.

Before using the speech tools, you'll want to make sure two settings in your Control Panel are correct (use the 3-key code -CONTROL-ESC to reach the Control Panel). First access the Sound area, using the arrow keys to set the Volume control so that the speech will be audible – we suggest starting with a fairly high volume setting. Next enter the System Speed area and set the speed to Fast; a speed of Normal will cause the synthesizer to stutter.

Overview of SpeakIt

SpeakIt is a simple talking editor. You can type text into a window, edit the text, save the window to disk, and print the window. You can also have the text spoken and translated to phonetics. Access to the speech tool kit's exceptions dictionary is supported as well.

Start the SpeakIt program by first booting your computer with the Talking Tools System Disk. Double-click on the SpeakIt icon to launch the demo program.

SpeakIt is a standard desktop program. That is, it contains a menu bar with pull-down menus and windows. Commands are executed by selecting them from menus. It is "standard" in the sense that its menu bar is organized like other desktop programs you'll see on the Apple IIGS, and the commands that it shares with other desktop programs all function in pretty much the same way. If you are not familiar with desktop programs, you should read Chapter 3 of the *Apple IIGS Owner's Guide*, which came with your computer.

Speaking English Text

SpeakIt's menu bar contains the Apple, File, Edit, and Speech menus. The Apple, File, and Edit menus are common to most desktop programs, while the Speech menu is specific to SpeakIt. Let's dive right in and start exploring. Pull down the File menu and select the New command. This creates a new text window on the desktop. Type in something to say, such as **Hello, World!!** then select the Speak command from the Speech menu. You can make your computer speak individual words or phrases by using the mouse to select the text to be spoken, and then issuing the Speak command.

This command illustrates the first function of the speech tool kit that we'll explore, the ability to speak arbitrary English phrases. In order to speak, the tool kit converts the English text internally into phonetics, then speaks the phonetics. The process of translating English text into phonemes, the individual pieces of the tool kit's phonetic language, is called parsing. The tool that performs the parsing is referred to as the front-end of the speech synthesizer. The tool sets that emit sounds from the phonemes are collectively called the back-end of the synthesizer.

There is much more to converting English to speech than merely emitting a series of sounds. The front-end analyzes sentence structure, applying over 1200 English grammar rules, then encodes stress, pitch, and inflection parameters with the phonemes. Built into the front-end is the ability to recognize abbreviations, such as etc, Dr., Ms., Mr., Mrs., and St., monetary figures, such as $2.39, numbers, dates, times, and mathematical symbols.

Experiment with different phrases to see how the speech tool kit says them. The table below shows some example pronunciations:

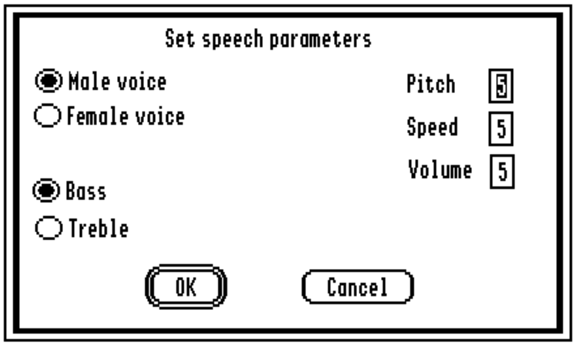
Text to Say	Talking Tools' Pronunciation
Dr. and Mr. Rootabaker	Doctor and Mister Root-a-baker
$32.45	Thirty-two dollars and forty-five cents
1 + 1 = 6	One plus one equals six
3 5/8	Three and five eighths

Controlling Speech Parameters

The speech tool kit defines five parameters that control the way in which text is spoken: tone, pitch, speed, volume, and gender. The basic pitch of the voice, either treble or bass, is set by the tone parameter. The pitch parameter determines the frequency of the emitted sound: the higher the pitch, the higher the sound. The speed parameter sets the speed at which the sounds are spoken. Volume refers to the loudness of the speech. The gender parameter controls the type of voice that speaks, and can be set to either adult male or adult female.

The parameters are defined both globally (the default settings for all spoken text) and locally (the settings for the next group of phonemes only). During the process of translating English to phonetics, the front-end embeds local parameters into the phonetic representation to reflect the structure of the phrase. When no parameters are embedded, the global values are used.

Talking Tools allows you to alter the global speech values. Pull down the `Speech` menu and select the `Set parameters...` command. It brings up this dialog box:



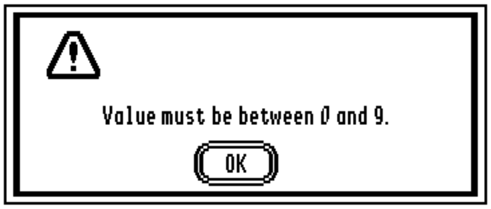
Set speech parameters

● Male voice
○ Female voice

● Bass
○ Treble

Pitch 5
Speed 5
Volume 5

The two radio buttons at the top left let you specify whether the default voice should be adult male or adult female. The radio buttons below the voice buttons let you choose the default tone, which can be either bass or treble. On the right side of the dialog is a sequence of three edit-line boxes that control the default settings for pitch, speed, and volume. You can use the TAB key to move from one edit-line box to another. The boxes accept a digit from 0 to 9. If you enter something other than a digit, this alert will appear:



Value must be between 0 and 9.

Simply click the OK button in the alert and you will be returned to the Set parameters dialog.

The OK button at the bottom of the dialog causes your choices to take effect immediately. Choosing the Cancel button exits the dialog with the original selections still in place.

To see how changing the global parameters affects speech, let's choose the voice to be female, the tone to be treble, a speed of 6, a pitch of 7, and a volume of 9. Click the OK button, then issue the Speak command (T) for your text window. You should hear your computer say, "Hello, World!" (or whatever text you've selected in your window) in a loud, fast, high-pitched female voice.

Talking Tools also provides a mechanism for controlling local speech parameters. You code a special string, called an embedded string, that is passed to the Speak command. The embedded string is formed by typing two less-than symbols (<<), immediately followed by any desired speech parameters, at least one space, the text to be spoken, and ending with two greater-than symbols. For example, type this line into your text window, then execute the Speak command:

Hi! <<P2V9 Low pitch, volume = 9>> Bye!

Assuming you had previously set the global voice to female and the global pitch to 7, you should have heard your computer say "Hi!" in a high-pitched female voice, followed by "Low pitch volume equals 9" in a moderately loud, deep voice, and ending with "Bye!" in a normal female voice. The P stands for pitch, and the V stands for volume. The table below summarizes the codes you can use to control local speech parameters:

Parameter	Embedding Code	Range of Values	Default
Pitch	P or a number by itself	0 (low) to 9 (high)	5
Speed	S	0 (slow) to 9 (fast)	5
Volume	V	0 (soft) to 9 (loud)	5
Tone	B (bass) or T (treble)	no range is used	B
Phonetics	~	no range is used	

You can use either uppercase or lowercase letters for the parameter codes. Strings can be embedded within one another up to a depth of nine strings. For example:

<<V9 loud voice <<V1 softer voice >> back to normal voice>>

The parameters can appear in any order, but you need to be careful when specifying bass or treble tone. If more than one tone parameter is given, the last one in the parameter list will be used.

Notice that the speed, volume, and pitch parameters are specified with two characters, the code letter followed by a digit. You can omit the P code for the pitch parameter, and just use a single digit. For example,

<<B9 pitch of 9, bass tone>>

The actual values used by Talking Tools for the speed, pitch, and volume parameters are a computed average of the global (default) setting and the local value. For example, the string **<<V9 fairly loud>>**, with a default volume setting of 5, would be spoken with an actual volume value of 7: (9 + 5) / 2 = 7. Nested parameters also affect the relative values of the parameters. Returning to a previous example, with a default volume setting of 5:

<<V9 loud voice <<V1 softer voice >> back to normal voice>>

The first volume parameter would be set to 7 ((9 + 5) / 2), the second would be set to 4 ((1 + 7) / 2), while the last one would be set to 6 ((9 + 4) / 2).

The last "parameter" in the table above, the tilde (~), informs Talking Tools that the text to be spoken is a phonetic string. It must be the first symbol following the embedding (<<) symbol. We'll examine the phonetic language in the next section.

Translating Text to Phonetics

The next Talking Tools function we'll look at is the process of translating an English string into its phonetic representation. Shrink your text window to make room on the desktop for another window, then type this sentence into your text window: **Hello, World!** Select this sentence (a quick way to select an entire line is to triple-click on it), pull down the `Speech` menu, and select the `Show phonetics` (H) command. A new window will appear on the desktop, entitled `Phonetics 1`, containing the text `9hEHl8OW w9ERld!` If you followed the discussion in the last section about speech parameters, some of the phonetic text probably looks familiar. The digits represent pitch levels, while the characters are the phonemes. Just to assure ourselves that the English sentence sounds the same as the phonetics, click on the `Phonetics` window then pull down the `Speech` menu and select the `Speak` command. Now click on your text window and issue the `Speak` command. The two pronunciations will be identical.

Of course, `9hEHl8OW w9ERld!` just doesn't have the same familiar look as `Hello, World!` There must be a pretty good reason to go to all of the trouble of creating a phonetic language, and of course there is. The problem with English text is that it doesn't match up exactly with the sounds we make when we read the text. To see this, try a simple experiment. Say the word boo out loud. Now say the word book aloud. OK, so what does oo sound like? Our phonetic language comes to the rescue. It knows that the oo sound differs in different words; it translates boo as bUW, and book as b3UHk.

Let's look at Talking Tools' phonetic language, summarized in the table below:

	Phonetic Code	Pronunciation
Vowels		
	AE	Short "a" as in "l**a**st"
	EH	Short "e" as in "b**e**st"
	IH	Short "i" as in "f**i**t"
	AA	Short "o" as in "c**o**t"
	AH	Short "u" as in "c**u**p"
	EY	Long "a" as in "**a**ce"
	IY	Long "e" as in "b**ee**t"
	AY	Long "i" as in "**i**ce"
	OW	Long "o" as in "d**o**se"
	UW	Long "u" as in "l**u**te"
	AO	Intermediate "o" as in "c**au**ght"
	UH	Intermediate "u" as in "b**oo**k"
	AX	schwa sound as in "**a**gainst"
	OY	diphthong in "n**oi**se"
	AW	diphthong in "l**ou**d"
	ER	"ur" as in f**ur**ther

Consonants

b	"**b**ig"
CH	"**ch**ild"
d	"**d**ig"
f	"**f**ig"
g	"**g**ood"
h	"**h**ave"
j	"**J**im"
k	"**c**ave"
l	"**l**ove"
m	"**m**an"
n	"**n**o"
NG	"**r**ing"
p	"**p**ig"
r	"**r**isk"
s	"**s**ave"
SH	"**sh**ip"
t	"**t**op"
TH	"**th**ing"
DH	"**th**at"
v	"**v**ery"
w	"**w**ater"
WH	"**wh**y"
y	"**y**onder"
z	"**Z**iggy"
ZH	"trea**s**ure"

Pitch control

/	Increase pitch by 1 step
\	Decrease pitch by 1 step
[	Says the following phoneme a little faster
]	Says the following phoneme a little slower

While exploring phonetics can be fun, you needn't know anything about the phonetic language to make effective use of the speech tools. The material is presented here for completeness, and to give you the necessary information to customize the tool kit for your own needs.

Ignoring that caveat for the moment, let's experiment with the phonetic language a bit more. There are four main uses for phonetic strings. First of all, you can embed a phonetic string within English text. This gives you even more control over how the string will be pronounced than simply using parameters. An embedded phonetic string is indicated by placing a tilde (~) immediately after the opening embedding symbol (<<). The string must follow the rules for forming valid phonetic strings, which we'll look at shortly. To illustrate passing phonetic strings to the speech tools, type this line into your text window, and then issue the `Speak` command:

<<~9hEHl8OW w9ERld>>

You should have heard your computer say, "Hello, World!"

Phonetic strings can also be passed directly to the back-end of the speech tool kit. You may recall that when we ask Talking Tools to speak an English string, it first converts the string to phonetics (parsing, using the front-end), and then sends the phonetics to the back-end, which uses the machine's sound hardware to speak. To bypass the front-end, we need to signal that we're sending phonetics only. This is handled in our demo program with two different types of windows, one for text and the other for phonetics. We looked at creating phonetics windows at the beginning of this section: type some English in a text window, then issue the `Show phonetics` (H) command. A good way to become familiar with the phonetics language it to enter short sentences into your text window, use the `Show phonetics` command to translate them to phonetics, and then use the `Speak` command to speak selected parts of the phonetics. Once you see how the language works, it's fun to interject speech parameters to change the stress of certain words and substitute different vowels to change the pronunciation.

The last use for phonetic strings is in customizing the exceptions dictionary, our final topic in this chapter.

Building Valid Phonetic Strings

When you were experimenting with the phonetic language, you may have noticed that the `Speak` command refused to speak certain phonetic strings. As an example, if you were using a phonetic string of **9hEHl8OW** for "Hello," but only selected the characters **9hE** to pass to the `Speak` command, nothing would happen. This is because the vowel phoneme of EH was split, and the tool kit doesn't know what a capital E by itself means. The first rule for forming valid phonetic strings, then, is that you must use complete phonemes. The tool kit also won't handle special characters inside of a phonetic string; the special characters must be converted to their equivalent phonetic codes.

The next point we need to make about phonetic strings is that they are similar in structure to the embedded English strings we looked at in an earlier section. There are some important differences, however. Pitch levels in a phonetic string are encoded as digits, without the preceding P code, and can also be set with the special symbols given at the end of the phonetic language table. Parameter codes can be placed anywhere within the string, not just at the beginning. Parameter codes must always appear in uppercase letters, to distinguish them from phonemes.

A parameter that can be used in a phonetic string that is not available in an embedded English string is delay, which causes a pause before speaking. The delay is coded as a capital D, followed by a digit in the range 0 to 9. For example, **S8hEY D2S3yUW**, for "hey you," would speak "hey" with a relative speed of 8, then pause before speaking "you" with a relative speed of 3. The table below gives the actual pauses for the possible delay values:

Delay value	Pause, in seconds
0	No delay
1	0.25
2	0.50
3	1
4	2
5	3
6	4
7	6
8	8
9	10

After wading through that wealth of information, let's look at a sample to help firm up the ideas. In the `Text` window on your desktop, enter the string

```
<<v9s7p6  Hey!>>
```

Select the string, then issue the `Speak` command. You should hear your computer say "Hey!" in a loud, fast, relatively high-pitched voice. Now let's use phonetics to say the same thing. In the `Phonetics` window on your desktop, enter the string

```
6V9S7hIY!
```

Once again, select the string, then issue the `Speak` command. The string should sound the same as that said in your `Text` window.

The table below summarizes the differences between how speech parameters are used in embedded text strings and those used in phonetic strings:

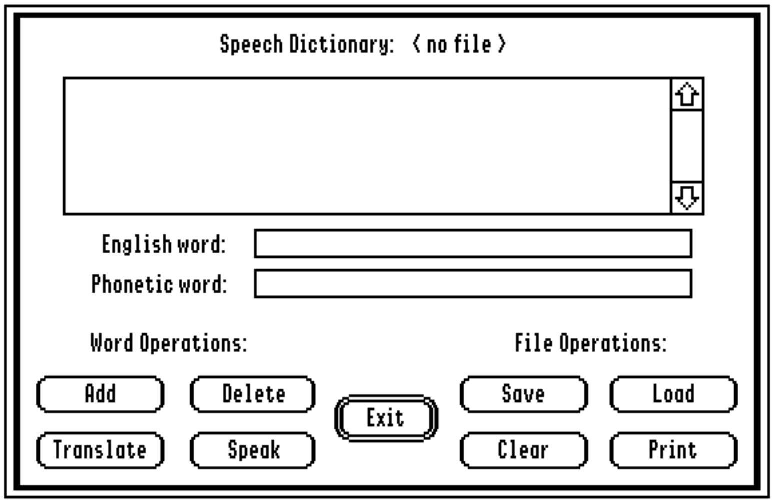
Parameter	English Code	Phonetics Code
Pitch	P, p or single digit	single digit or /, \, [,]
Speed	S, s	S
Volume	V, v	V
Tone	B (bass) or T (treble)	B (bass) or T (treble)
Phonetics	~	Not used
Delay	Not used	D

Parameters must appear immediately after embedding symbol (<<) in an English string, but can be used anywhere in a phonetic string.

The Exceptions Dictionary

The last two commands in SpeakIt's `Speech` menu, `Dictionary...` and `Dictionary on/off`, control Talking Tools' on-line exceptions dictionary. The dictionary is used to change the way in which words would normally be spoken. A dictionary entry consists of an English word and its phonetic translation; by altering the phonetics, you will change the way the word is spoken.

Let's try a sample to see how this works. Pull down the `Speech` menu and select the `Dictionary` command. This brings up the dictionary editor:



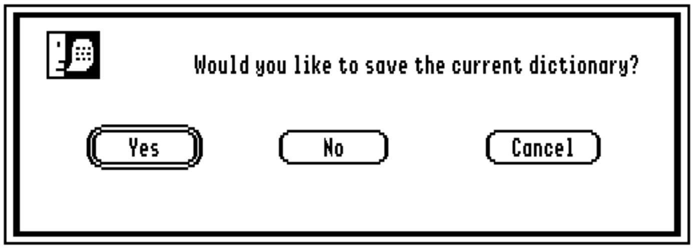
At the top of the window is the message `Speech Dictionary`, followed by the name of the file from which the current dictionary is derived. If the dictionary was not loaded from a file, the message will be `Speech Dictionary: < no file >`. When the SpeakIt program starts up, it looks in the System folder of the boot disk for a special file named SpeechDict. If it finds the file, it uses it to initialize the dictionary. If it doesn't find the file, the dictionary is initialized to empty.

Beneath the message is a scrollable list of the English words in the current dictionary. This list is limited to 16,383 entries.

Below the list are edit boxes labeled `English word` and `Phonetics word`, where you can make new dictionary entries. If you select an English word from the list, the boxes will be filled with the entry corresponding to the one you selected.

The bottom left portion of the dictionary window contains four buttons used for word operations. The `Add` button inserts a new entry into the dictionary, while the `Delete` button removes an entry. The entry itself is defined by what appears in the `English` and `Phonetics` edit boxes. The `Translate` button converts the English word in the `English` edit box to its phonetic equivalent, placing the result in the `Phonetics` edit box. The `Speak` button speaks what is in the `Phonetics` edit box. When the dictionary window is front-most, the `Speak` command in Speech menu performs the same function as the Speak button, and the Show phonetics command operates identically to the Translate button.

The buttons on the right are used for file operations. The Save button brings up a standard save-file dialog, and is used to save the current dictionary to disk. The Load button brings up a standard open-file dialog, and is used to load a new dictionary from disk. If the current dictionary has changed, you will be asked whether you want to save it before loading in a new dictionary.



Would you like to save the current dictionary?

The Clear button is used to remove the current dictionary from memory. As with the Load button, you will be asked whether you want to save the current dictionary, if it has changed, before removing it.

The Print button is used to print the current dictionary, one entry per line. It uses standard print dialogs.

When the dictionary window is front-most, the Save as command in the File menu performs the same function as the Save button, the Open... command operates identically to the Load button, and the Print... command is identical to the Print button.

You can close the dictionary window by clicking the Exit button or with the Close command in the File menu. When the dictionary window is front-most, selecting the Dictionary command in the Speech menu will also close the window. If the dictionary window is open on the desktop, but other windows are on top of it, choosing the Dictionary command will bring it to the front.

Back to playing with the dictionary. Enter the word **cat** in the English word edit box, click the Translate button, then click the Speak button. You should hear your computer say, "cat." Now replace the English word **cat** with **dog**, then click the Add button. The list will scroll to show DOG in your dictionary. Now use the New command from the File menu to create a new text window. Type in the sentence **My dog has fleas** then issue the Speak command (T). You should hear your computer say, "My cat has fleas." It should be obvious what is happening: when the parser sees the word dog, it substitutes the phonetic word 3kAEt, which sounds like the English word "cat."

This brings us to our final command in the Speech menu, Dictionary off. Pull down the Speech menu and select this command, then use the Speak command to say the "My dog has fleas" phrase. This time you should hear your computer say the word dog, rather than cat. That's because we've temporarily disabled the dictionary. Pull down the Speech menu again, and you'll see that the name of the last command has changed to Dictionary on. Selecting the command again will turn the dictionary back on, and will change the menu name to Dictionary off.

Chapter 3 – Speaking from a Computer Program

A computer program that uses Talking Tools' speech tool kit is written in much the same way as any other program that uses the Apple IIGS toolbox: the required tools are started up during the program's initialization phase, various tool calls are made in the body of the program, and finally the tools are shut down at the end of the program. In this chapter, we'll discuss in detail a "plain vanilla" program that demonstrates all of the available tool calls in the tool kit. We will concentrate on the Pascal version of our sample program, since Pascal is similar enough to English to be understood by programmers familiar with at least one high-level language. The source code listings for all three versions of the sample is given in Appendix A. You will find a complete reference for the tool kit in Chapter 4.

Accessing the Interface Files

Before writing a program that uses the speech tool kit, you will need to move the proper interface file to the correct folder for your compiler or assembler. For ORCA/Pascal, the interface file is named SpeechTools.Int, and is located in the Pascal folder on the Talking Tools disk. You'll want to copy this file to the Libraries/ORCAPascalDefs folder of your ORCA/Pascal program disk. For ORCA/C, the interface file is named speech.h, and is located in the CLang folder on the Talking Tools disk. You should copy this file to the Libraries/ORCACDefs folder of your ORCA/C program disk. For ORCA/M, the interface file is named M16.Speech, and is located in the Asm folder on the Talking Tools disk. You'll want to copy this file to the Libraries/AInclude folder on your ORCA/M disk.

As mentioned in Chapter 1, you'll need to use a boot disk that has the speech tools installed in the Tools folder of the System folder. The speech tools consist of four separate tool sets: tool number 50, the male speech tool; tool number 51, the female speech tool; tool number 52, the English text to phonetics translator; and tool number 53, the interface between the speech tools and GS/OS. You can copy these files manually, or use the installer to move the files, as described in Chapter 1.

As you read this chapter, you will see various chunks of source code in Pascal. The complete source listing for the executable program described in this chapter is also in Appendix A, only there you will find it not only in Pascal, but in C and assembly language as well. Of course, the source code is also in the Talking Tools disk, again in all three languages.

The Speak Program at the Global Level

At the beginning of the program, we declare the interface files that we'll use. These include Common, the Tool Locator, the Integer Math Tool Set, and the speech tools themselves:

```
uses Common, ToolLocator, MemoryMgr, IntegerMath, SpeechTools;
```

Following the interface files are our global variables. We'll take a closer look at these variables as they're used. For now, you will notice that our global variables can be divided into two main groups: those used in a general way, and those which deal specifically with speech.

```
var
   answer: string;                      {user's response to queries}
   done: boolean;                       {true if user wants to exit pgm}
   toolRec: toolTable;                  {table of tools we need to start}

   voice: Gender;                       {current global voice setting}
   basePitch: Tone;                     {current global tone setting}
   speed,                               {current global speed setting}
   pitch,                               {current global pitch setting}
   volume: ParmRange;                   {current global volume setting}
```

The global variables are followed by the program's subroutines. They are in alphabetical order by procedure name. We'll look at each subroutine below. For now, let's skip down to the main program.

The first five `writeln`s comprise a simple "splash screen," introducing the user to our program and letting him know that we'll be loading some tools, which could take some time.

```
{----------------------------------------------------------------}
{                                                                }
{  Main program - Display "main menu" and call appropriate      }
{       function until user selects Quit.                        }
{                                                                }
{----------------------------------------------------------------}

begin
{Splash screen.}
writeln;
writeln('Speak - A demonstration of the Talking Tools');
writeln;
writeln('Please wait while we load the tools.');
writeln;
```

After the introduction, the program calls our `Init` procedure, which loads the tools we need and initializes our global variables. We use a global boolean named `done` to control our main loop. If the `Init` procedure encounters any errors, it sets `done` to true in order to avoid entering the main loop.

```
{Initialize the program.}
Init;
```

Our main loop consists of bringing up a menu of choices for the user, reading the user's selection, and then calling the appropriate routine. If the user enters **Q** (for quit), we set `done` to true, which will bring us out of the main loop.

```
{Main loop: bring up menu; get user's selection; handle selection.}
while not done do begin
   writeln('Enter desired function:  S to speak English string');
   writeln('                         P to speak phonetic string');
   writeln('                         C to convert to phonetics');
   writeln('                         G to set global speech parameters');
   writeln('                         A to activate dictionary');
   writeln('                         T to deactivate dictionary');
   writeln('                         D to display dictionary');
   writeln('                         I to insert word into dictionary');
   writeln('                         R to remove word from dictionary');
   writeln('                         L to load dictionary from disk file');
   writeln('                         W to write dictionary to disk file');
   writeln('                         Q to quit program');
   write('                         ');
   readln(answer);
```

Our main menu reflects the available functions of the speech tool kit: speaking an English string; speaking a phonetic string; converting an English string to its phonetic representation; and the dictionary functions to insert, delete, and display entries, as well as to activate, deactivate, and clear the dictionary. We dispatch a subroutine to handle the selected speech function or set `done` to true if **Q** is entered:

```
   case answer[1] of
      'S', 's': SpeakText;
      'P', 'p': SpeakPhonetics;
      'C', 'c': ConvertToPhonetics;
      'G', 'g': SetSpeechGlobals;
      'A', 'a': DictActivate(1);
      'T', 't': DictActivate(0);
      'D', 'd': DisplayDict;
      'I', 'i': InsertWord;
      'R', 'r': DeleteWord;
      'L', 'l': LoadDict;
      'W', 'w': WriteDict;
      'Q', 'q': done := true;
      otherwise: begin
                 writeln('Please enter one of S, P, C, G, A, T, D, I, R, L, W, or Q...');
                 writeln;
                 end;
      end; {case}
end; {while}
```

Following the main loop, we call our `ShutDown` routine to shut down the tools we've started.

```
{Shut down the program.}
ShutDown;
end.
```

The Init Procedure

Our `Init` procedure attempts to start the tools we need. If it is successful, it then initializes our global variables and `done` is set to false. If unsuccessful, `done` is set to true.

Since our final program will be an EXE file, we do not need to start any of the tools we'll be using that are guaranteed to be started by the shell. For our program, these include the Tool Locator and Integer Math Tool Set. (These tool sets are described in volumes one and two of the *Apple IIGS Toolbox Reference* manuals.) We'll be calling the Tool Locator to load our speech tools, and we'll be using the Integer Math Tool Set to format error codes as hexadecimal numbers, in order to report errors in the form used by the system's toolbox.

We begin our Init procedure by loading the speech tools, which are RAM based, into memory. The speech tools are loaded by adding them to our tool record data structure and then making the Tool Locator's LoadTools call, passing it the tool record we've constructed. If the LoadTools call is successful, we then make a start up call for each speech tool that we loaded.

{The following code is from the Tool Locator interface file}

```
type
   (* Table of tools to load from the TOOLS directory in the SYSTEM folder *)
   toolSpec = record
       toolNumber: integer;
       minVersion: integer;
       end;

   (* Change array size for your application. *)
   ttArray = array [1..20] of toolSpec;

   toolTable = record
       numToolsRequired:  integer;
       tool:              ttArray;
       end;
```

{The following code is from the Speak program}

```
{---------------------------------------------------------------}
{                                                               }
{  Init - Load the tools we need and initialize our data        }
{       structures.                                             }
{                                                               }
{---------------------------------------------------------------}

procedure Init;

var
   errNum: integer;                     {error number to report to user}
   errString: packed array[1..5]   of char; {error number as a hex string}

begin {Init}
errString[1] := '$';                    {return error codes as hex numbers}
with toolRec do begin
   numToolsRequired := 4;
   with tool[1] do begin
      toolNumber := maleToolNum;
      minVersion := 0;
      end; {with}
   with tool[2] do begin
      toolNumber := femaleToolNum;
      minVersion := 0;
      end; {with}
   with tool[3] do begin
      toolNumber := parserToolNum;
      minVersion := 0;
      end; {with}
   with tool[4] do begin
      toolNumber := speechToolNum;
      minVersion := 0;
      end; {with}
   end; {with}
LoadTools(toolRec);                     {load the tools}
errNum := toolError;
if errNum <> 0 then begin               {report any error returned}
   Int2Hex(errNum, pointer(@errString[2]), 4);
   writeln('Unable to load tools: Error = ', errString);
   done := true;
   end {if}
else begin                              {start the tools}
   ParseStartUp(userID);
   MaleStartUp;
   FemaleStartUp;
   SpeechStartUp;
```

The speech tool kit defines five global speech parameters: voice, either male or female; tone, either bass or treble; and speed, pitch, and volume, each specified on a scale of 0 to 9, 9 the highest value. Our global speech parameters are assigned the default values used by the speech tool kit:

```
{The following code is from the SpeechTools interface file}

type
   pString32    = packed array [0..32] of char;
   pString32Ptr = ^pString32;

   Gender = ( Male, Female );
   Tone   = ( Bass, Treble );

   ParmRange = 0..9;
```

```
{The following code is from the global variables section of the Speak program}

  voice:        Gender;                      {current global voice setting   }
  basePitch:    Tone;                        {current global tone setting    }
  speed,                                     {current global speed setting   }
  pitch,                                     {current global pitch setting   }
  volume:       ParmRange;                   {current global volume setting  }
```

```
{The following code is from the Init subroutine of the Speak program}

   done := false;                      {initialize globals}
   voice := male;                      {these are the default settings for}
   basePitch := bass;                  {the global speech parameters}
   speed := 5;
   volume := 5;
   pitch := 5;
   end; {else}
end; {Init}
```

Speaking an English String

The first speech tool call we'll look at is named Say. Say accepts one parameter, a pointer to an English string. It first converts the string to its phonetic representation, and then speaks the string. Say is useful for speaking random strings; that is, when it cannot be known in advance what is to be spoken.

The call to the Say procedure is shown in bold in the source code below. The string we pass Say is a "Pascal-style" string. Since Pascal strings start with a length byte, the total number of characters you can pass is limited to 255.

Our SpeakText procedure, shown below, is very simple: we allow the user to Say as many strings as he wants; the user signals he's ready to stop Saying by entering a null string (i.e., by hitting the RETURN key right away).

```
{-------------------------------------------------------------}
{                                                             }
{  SpeakText - Speak as many non-empty lines of English text as }
{       the user wants.                                       }
{                                                             }
{-------------------------------------------------------------}

procedure SpeakText;

var
   sayString: packed array[0..255] of char; {English text to speak or parse}
   stop: boolean;                      {true if user wants to exit}
```

```
begin {SpeakText}
stop := false;
repeat
   writeln;
   writeln('Enter string to speak.  Press RETURN to exit.');
   readln(sayString);
   if length(sayString) = 0 then
      stop := true
   else
      Say(sayString);
until stop;
writeln;
end; {SpeakText}
```

Converting an English String to Phonetics

The Say procedure provides the easiest way to generate speech. Another method of generating speech is to first translate the string to phonetics, and then call the male or female voice to speak the phonetics. This method works well when the same string will be repeated over and over, since the process of converting the string from text to phonetics is only done once, as opposed to translating the text to phonetics each time the string is spoken.

The tool call to perform the translation from English to phonetics is named Parse, shown in bold in the source code below. The parameters we pass to Parse include a pointer to the English string to be translated, a pointer to the string to receive the phonetics, and an integer that tells the Parse function how many characters to skip before it starts to translate the text into phonetic characters. Parse returns an integer, the position in the string where the conversion stopped. A character's position in the string is counted from one. Be careful! Parse expects the phonetic string to be 255 characters long. If your string is less than 255 characters, Parse could overwrite whatever is beyond the end of the string, without warning.

Since the output string is limited to 255 characters, it is possible that Parse will not be able to translate all of the text. Parse returns a displacement into the text string; if it finished, then this value will be equal to the length of the input text. If the returned value is smaller than the length of the string, then Parse didn't have enough room to store all of the phonetic characters, and you will have to call it again. On the next call, you would pass the value returned by Parse+1 as the displacement into the text string, and Parse would pick up where it left off.

In the following sample program, you can see one way to handle these long input strings. Once again, to leave this subroutine, press the RETURN key right away to enter a null string.

```
{-------------------------------------------------------------}
{                                                             }
{  ConvertToPhonetics - Convert English text to phonetic      }
{      representation.                                        }
{                                                             }
{-------------------------------------------------------------}

procedure ConvertToPhonetics;

var
   phString: packed array[0..255] of char; {phonetic string}
   sayString: packed array[0..255] of char; {English text to speak or parse}
   size: integer;                      {length of string}
   start: integer;                     {position in English string to begin}
                                       { conversion                       }
   stop: boolean;                      {true if user wants to exit}
```

```
begin {ConvertToPhonetics}
stop := false;

{Outer loop lets user enter next string to convert.}
{Entering null string signals it's time to exit.}
repeat
   writeln;
   writeln('Enter string to translate to phonetics.  Press RETURN to exit.');
   readln(sayString);
   size  := length(sayString);
   start := 0;

   {Inner loop is necessary in the event that the }
   { complete English string wasn't converted.    }
   if size > 0 then begin
      repeat
         start := start+1;
         start := Parse(sayString, phString, start);
         write(phString);
      until start = size;
      writeln;
      end {if}
   else
      stop := true;
until stop;
end; {ConvertToPhonetics}
```

Speaking a Phonetic String

The next two speech tool calls we'll look at are used to generate speech from a phonetic string. The functions are named MaleSpeak and FemaleSpeak, and invoke the back-end's male and female voices, respectively. The parameters we pass the two routines are identical, and include three integers that specify the volume, speed, and pitch values that are to be applied to the speech, and a pointer to the phonetic string to speak. As always, the values you pass for volume, speed and pitch should be in the range 0 to 9. The phonetic string itself is a p-string.

```
{--------------------------------------------------------------}
{                                                              }
{  SpeakPhonetics - Speak as many non-empty lines of phonetic  }
{       text as the user wants.                                }
{                                                              }
{--------------------------------------------------------------}

procedure SpeakPhonetics;

var
   phString: packed array[0..255] of char; {phonetic string}
   stop: boolean;                       {true if user wants to exit}
```

```
begin {SpeakPhonetics}
stop := false;
repeat
   writeln;
   writeln('Enter phonetic string to speak.  Press RETURN to exit.');
   readln(phString);
   if length(phString) = 0 then
      stop := true
   else begin
      if voice = male then
         MaleSpeak(volume, speed, pitch, phString)
      else
         FemaleSpeak(volume, speed, pitch, phString);
      end; {else}
until stop;
writeln;
end; {SpeakPhonetics}
```

Back in Chapter 2, you saw that an illegal phonetic string would stop the speech output, and the same thing is true with this call. For example, suppose parsing some English text that included the word "hat" resulted in the two strings: hA and Et, both of which are to be spoken one after the other. Both calls to the Speak function would fail, since the vowel AE has been split. You can avoid this problem in your own programs by checking the last two characters in the phonetic string returned by the Parse function. If both characters are capital letters or if the next-to-the-last character only is a capital, you don't need to do anything – you've found a complete phoneme. On the other hand, if only the last character is a capital letter, then you're splitting a phoneme. To remedy the situation, you'll need to decrement the start variable and the length of the string. The sample code below, adapted from our ConvertToPhonetics procedure, demonstrates the technique:

```
  readln(sayString);
  size := length(sayString);

  {Inner loop is necessary in the event that the}
  {complete English string wasn't converted.    }
  if size > 0 then begin
     repeat
        start := start+1;
        start := Parse(sayString, phString, start);
        phSize := length(phString);
        if (phString[phSize] >= 'A') and (phString[phSize] <= 'Z') then
           if (phString[phSize-1] < 'A') or (phString[phSize-1] > 'Z') then begin
              start := start-1;
              phString[0] := phString[0]-1;
           end; {if}
        write(phString);
     until start = size;
     writeln;
     end; {if}
```

Setting Global Speech Parameters

When English is spoken using the Say procedure, the current global values of voice gender, tone, pitch, speed, and volume are applied. The speech tool kit provides a mechanism for setting the global speech parameters, in a procedure named SetSayGlobals. The procedure accepts five integer parameters which set the voice, tone, pitch, speed, and volume global values. For our readers who are Pascal purists, you'll notice that the voice and tone parameters to the call are actually enumerated types. This actually works out fine, since all Pascal compilers use ordinal values for enumerations that start with zero and count up from there, and all Pascal compilers on the Apple IIGS represent integers and enumerated values internally the same way.

Our program's SetSpeechGlobals procedure, shown below, uses a short function to make sure that the number the user types is in an acceptable range. This subroutine also handles reprompting when the value is bad. Once all of the values have been obtained, the SetSayGlobals routine is called to set the global speech parameters.

```
{----------------------------------------------------------------}
{                                                                }
{  SetSpeechGlobals - Set global speech parameters.              }
{                                                                }
{----------------------------------------------------------------}

procedure SetSpeechGlobals;


   function GetValue (min,max: integer): integer;

   { Get a value, making sure it is in the given range          }
   {                                                            }
   { Parameters:                                                }
   {    min - lowest allowed value                              }
   {    max - highest allowed value                             }
   {                                                            }
   { Returns: Value read                                        }

   var
      value: integer;                  {value read}

   begin {GetValue}
   repeat
      readln(value);
      if (value < min) or (value > max) then begin
         writeln('Please enter a value from ', min:1, ' to ', max:1, '.');
         writeln;
         write('  Value: ');
         end; {if}
   until (value >= min) and (value <= max);
   GetValue := value;
   end; {GetValue}


begin {SetSpeechGlobals}
write('Voice = ');                        {Read new global voice setting}
if voice = male then
   writeln('male ')
else
   writeln('female ');
writeln('Enter 0 to change voice to male, 1 to change voice to female.');
if GetValue(0,1) = 0 then
   voice := male
else
   voice := female;
writeln;                                  {Read new global tone setting}
```

```
write('Tone = ');
if basePitch = bass then
   writeln('bass ')
else
   writeln('treble ');
writeln('Enter 0 to change tone to bass, 1 to change tone to treble.');
if GetValue(0,1) = 0 then
   basePitch := bass
else
   basePitch := treble;
writeln;                                {Read new global volume setting}
write('Volume = ', volume:1, '  ');
volume := GetValue(0,9);
writeln;                                {Read new global speed setting}
write('Speed = ', speed:1, '  ');
speed := GetValue(0,9);
writeln;                                {Read new global pitch setting}
write('Pitch = ', pitch:1, '  ');
pitch := GetValue(0,9);
                                        {set the globals}
SetSayGlobals(voice, basePitch, pitch, speed, volume);
writeln;
end; {SetSpeechGlobals}
```

The Exceptions Dictionary

While it is being run, the speech tool kit maintains an on-line dictionary containing words that are to be pronounced in a special way. Access to the tool kit's dictionary is provided by five tool calls: DictInit, which initializes the dictionary; DictDump, which returns the next entry in the dictionary; DictInsert, used to add an entry to the dictionary; DictDelete, which removes an entry from the dictionary; and DictActivate, which enables or disables the current dictionary. The dictionary is initially empty; entries are added to it with the insert function and are removed from it with delete calls. Notice that the dictionary resides only in memory – the tool kit knows nothing about dictionary files on disk. Our program includes subroutines to handle the disk I/O; you can use these subroutines in your own programs to load and save dictionaries.

Displaying the Dictionary

Our DisplayDict procedure displays the contents of the dictionary, one entry at a time. As you may recall from Chapter 2, a dictionary entry consists of an English word and the phonetics to be used for the word whenever the parser is called (either with the Say procedure or the Parse function). Both words are restricted to a maximum of 32 characters each. The entries are stored in alphabetical order, based on the English words. The front-end maintains a pointer to the current dictionary entry. Every time the tool kit's DictDump function is called, the pointer is advanced to the next entry. When the end of the dictionary is reached, the pointer is set to a null string. The speech tool kit provides a routine named DictInit to reset the pointer into the dictionary. DictInit accepts one integer parameter, a flag that tells the parser how to reset the pointer. A flag of zero causes the pointer to be set to the first entry, while a flag of one removes the dictionary from memory and sets the pointer to a null string. This is the purpose of the first part of our DisplayDict procedure; we give the user the option of resetting the dictionary pointer:

```
{---------------------------------------------------------------}
{                                                               }
{  DisplayDict - Displays current exceptions dictionary, one    }
{       entry at a time.                                        }
{                                                               }
{---------------------------------------------------------------}

procedure DisplayDict;

var
   answer: string;                      {user's response to queries}
   flag: integer;                       {dict. initialization flag}
   noErr: boolean;                      {true if no error has occurred}
   stop: boolean;                       {true if user wants to exit}
   word1, word2: pString32;             {dictionary entry}
   wordPtr: pString32Ptr;               {pointer returned by DictDump function}

begin {DisplayDict}
writeln;

{Before displaying the dictionary, let user initialize it.}
repeat
   noErr := true;
   writeln('Before displaying the dictionary, lets initialize it.');
   writeln('Enter 0 to reset dictionary to beginning.');
   writeln('Enter 1 to delete current dictionary.');
   writeln('Enter 2 to NOT initialize dictionary.');
   writeln;
   readln(flag);
   if (flag < 0) or (flag > 2) then begin
       noErr := false;
       writeln;
       writeln('Please enter either 0, 1, or 2.');
       writeln;
       end; {if}
until noErr;

if flag <> 2 then
   DictInit(flag);
```

The second part of our DisplayDict procedure is a loop to show the dictionary entries, one after another until the user signals he's ready to stop or the end of the dictionary is reached. The parameters we pass the DictDump function are pointers to strings to receive the English word and phonetic word. The call returns a pointer to the current phonetic word entry.

```
{While there are still entries in the dictionary, }
{ get and then display the next entry.            }
stop := false;
repeat
   wordPtr := DictDump(word1, word2);
   if length(word1) = 0 then
      stop := true
   else begin
      writeln('Next entry:   ', word1, '   ', word2, '   Continue? (Y or N)');
      readln(answer);
      if (answer[1] = 'N') or (answer[1] = 'n') then
         stop := true;
      end; {else}
until stop;
writeln;
end; {DisplayDict}
```

Adding Entries to the Dictionary

New words are added to the dictionary with the speech tool kit's DictInsert procedure. The call accepts two parameters, a pointer to the English word and a pointer to the phonetics to be used whenever the English word is encountered during parsing. Notice that the English word is case-insensitive; that is, all English words in the dictionary are stored in all uppercase letters, so that if you first added the word "have" and then inserted the word "Have," only the second addition would appear in the new dictionary.

As with our other procedures in our sample program, our InsertWord procedure contains a loop to allow the user to insert as many words into the dictionary as he wants. The user signals he's finished inserting words by entering a null string for the two words that comprise a dictionary entry.

```
{---------------------------------------------------------------}
{                                                               }
{  InsertWord - Insert new entries into exceptions dictionary.  }
{                                                               }
{---------------------------------------------------------------}

procedure InsertWord;

var
   stop: boolean;                      {true if user wants to exit}
   word1, word2: pString32;            {dictionary entry}

begin {InsertWord}
stop := false;
writeln;
repeat
   writeln('Press RETURN for the dictionary entries to exit function.');
   write('Enter English word to add to dictionary:  ');
   readln(word1);
   write('Enter phonetic representation of word to add to dictionary:  ');
   readln(word2);
   if (length(word1) = 0) or (length(word2) = 0) then
      stop := true
   else
      DictInsert(word1, word2);
   writeln;
until stop;
writeln;
end; {InsertWord}
```

Removing Entries from the Dictionary

Words are deleted from the dictionary with the speech tool kit's DictDelete procedure. The call accepts one parameter, a pointer to the English word to be removed. As with the DictInsert procedure, the English word is case-insensitive; this means that you can mix any combination of uppercase and lowercase letters in the word that you pass to DictDelete.

Our DeleteWord procedure contains a loop to allow the user to delete as many words from the dictionary as wanted. The user signals he's finished by entering a null string for the word to delete.

```
{----------------------------------------------------------------}
{                                                                }
{  DeleteWord - Removes entries from the current exceptions      }
{       dictionary.                                              }
{                                                                }
{----------------------------------------------------------------}

procedure DeleteWord;

var
   stop: boolean;                       {true if user wants to exit}
   word: pString32;                     {dictionary entry}

begin {DeleteWord}
stop := false;
writeln;
repeat
   writeln('Press RETURN for the dictionary entry to exit function.');
   write('Word to delete from dictionary?  ');
   readln(word);
   if length(word) = 0 then
      stop := true
   else
      DictDelete(word);
   writeln;
until stop;
writeln;
end; {DeleteWord}
```

Loading A Dictionary From Disk

Loading a dictionary from a disk file is very easy: we get the path name of the file to load, open the file, and then read entries from the file, inserting them into the dictionary, until we reach the end of the file. There are a number of ways to read the entries from the file, based on how the file was created. In our sample program, we created the file as a sequence of variable-length p-strings, so we need to read them back in the same format:

```
{----------------------------------------------------------------}
{                                                                }
{  LoadDict - Load dictionary file from disk.                    }
{                                                                }
{----------------------------------------------------------------}

procedure LoadDict;

label 99;

var
   ch: char;                            {character from the file}
   errNum: integer;                     {error number to report to user}
   errString: packed array[1..5]   of char; {error number as a hex string}
   f: text;                             {file variable}
   i: integer;                          {loop/index variable}
   pathname: string[255];               {name of the file}
   word1, word2: pString32;             {dictionary entry}

begin {LoadDict}
{Get pathname of dictionary to open.}
write('Enter pathname of dictionary to open:  ');
readln(pathname);
if length(pathname) = 0 then
   goto 99;
```

```
{Open the file for reading.}
reset(f, pathname);
errNum := toolError;
if errNum <> 0 then begin               {report any error returned}
   Int2Hex(errNum, pointer(@errString[2]), 4);
   writeln('Unable to open file:  Error = ', errString);
   goto 99;
   end; {if}

{Build the dictionary from the file.}
DictInit(1);                            {clear current dict from memory}
while not (eof(f)) do begin             {Loop:}
   if eoln(f) then
      readln(f)
   else begin
      read(f, ch);                      {read English word}
      word1[0] := ch;
      for i := 1 to ord(ch) do
         read(f, word1[i]);
      read(f, ch);                      {read phonetic word}
      word2[0] := ch;
      for i := 1 to ord(ch) do
         read(f, word2[i]);
      DictInsert(word1, word2);         {insert entry into dict}
      end; {else}
   end; {while}
close(f);
DictInit(0);                            {reset dict to top}

99:
end; {LoadDict}
```

Saving A Dictionary To Disk

Saving the current dictionary to a disk file is as simple as loading one: we get the name of the file to hold the dictionary, open it, and then write entries to the file until we reach the end of the dictionary. Perhaps the easiest way to create a dictionary file using the Pascal language is to declare our file variable as being of type file of pString32, and then write the entries with a single write statement:

```
var
   f: file of pString32;

begin
{Get pathname...}
rewrite(f, pathname);
{Write the dictionary to the file.}
DictInit(0);                               {set dictionary to top}
repeat                                        {Loop:}
   stop := false;
   tmp := DictDump(word1, word2);          {get next dict entry}
   if length(word1) = 0 then
      stop := true
   else
      write(f, word1, word2);              {write entry to file}
until stop;
```

When the file is created, each word will occupy 33 characters, with zeroes appearing at the end of words that are shorter than 33 characters. To save disk space, we want our entries to take up only as much space as they really need. This is accomplished by writing the characters individually:

```
{---------------------------------------------------------------}
{                                                               }
{  WriteDict - Write dictionary to disk file.                   }
{                                                               }
{---------------------------------------------------------------}

procedure WriteDict;

label 99;

var
   errNum: integer;                     {error number to report to user}
   errString: packed array[1..5]   of char; {error number as a hex string}
   f: text;                             {file variable}
   i: integer;                          {loop/index variable}
   pathname: string[255];               {name of the file}
   stop: boolean;                       {true if user wants to exit}
   tmp: pString32Ptr;                   {pointer returned by DictDump}
   word1, word2: pString32;             {dictionary entry}

begin
{Get pathname for dictionary file.}
write('Enter pathname for dictionary file:  ');
readln(pathname);
if length(pathname) = 0 then
   goto 99;

{Open the file for writing.}
rewrite(f, pathname);
errNum := toolError;
if errNum <> 0 then begin               {report any error returned}
   Int2Hex(errNum, pointer(@errString[2]), 4);
   writeln('Unable to open file:  Error = ', errString);
   goto 99;
   end; {if}

{Write the dictionary to the file.}
DictInit(0);                            {set dictionary to top}
stop := false;
repeat                                  {Loop:}
   tmp := DictDump(word1, word2);       {get next dict entry}
   if length(word1) = 0 then
      stop := true
   else begin
      for i := 0 to length(word1) do    {write English word}
         write(f, word1[i]);
      for i := 0 to length(word2) do    {write phonetic word}
         write(f, word2[i]);
      end; {else}
until stop;
close(f);
DictInit(0);                            {reset dict to top}

99:
end; {WriteDict}
```

The ShutDown Procedure

After we're through using the speech tool sets, we must shut them down before leaving the program. The tool shut down calls are very simple, with none of them requiring parameters.

Our ShutDown procedure, shown below, shows how to make these tool calls. Notice that the order in which we shut down our tools differs from the standard way tools are shut down on the

Apple IIGS. Normally, tools are shut down in the reverse order of the way they were started. The speech tools require that the Speech tool set be started last, and also it should be shut down last.

```
{---------------------------------------------------------------}
{                                                               }
{  ShutDown - Shut down the tools we started; do any necessary  }
{       clean-up before exiting.                                }
{                                                               }
{---------------------------------------------------------------}

procedure ShutDown;

begin {ShutDown}
FemaleShutDown;                         {shut down speech tools}
MaleShutDown;
ParseShutDown;
SpeechShutDown;
end; {ShutDown}
```

Chapter 4 – Talking Tools Tool Set Reference

Talking Tools' speech tool kit is a set of four different tool sets that work together to generate high-quality speech on the Apple IIGS computer. Each tool set performs a separate function in generating speech. The Parser tool set is the "front-end" module; it converts English text to a phonetic representation, using the current values of the five global speech parameters and its exceptions dictionary to create the phonemes. The Male Speech and Female Speech tool sets form the "back-end" modules; they speak phonetic strings with the computer's ENSONIQ sound chip. The Speech tool set provides an interface between the other three tool sets and the latest operating system on the machine.

When using the speech tool kit you'll need to make sure that the appropriate tools are installed in the SYSTEM/TOOLS folder of the boot disk. The speech tool kit's tools are: Tool050, the Male Speech tool set; Tool051, the Female Speech tool set; Tool052, the English text to phonetics translator tool set; and Tool053, the interface to the GS/OS operating system. To install them on your boot disk, simply copy the four tool files from the SYSTEM/TOOLS folder of the system disk that came in your Talking Tools package to the SYSTEM/TOOLS folder of your boot disk.

For readability, this chapter is organized along the lines of the *Apple IIGS Toolbox Reference* manuals. Each tool set is described in its own section, and the individual calls for a tool set are first organized with the housekeeping functions of boot initialization, start up, shut down, version, reset, and status, followed by the other calls in alphabetical order. Stack diagrams are included with each call, and source code for each of the programming languages Pascal, C, and assembly is shown. In the cases of Pascal and C, the function declarations used in their respective interface files is given. For assembly language, the sample source code shows what parameters to push (if any), followed by a call to the appropriate macro, and ending with pulling return values (if any). In the sections below, we'll look at the steps needed to use Talking Tools with each of these languages.

Using Talking Tools With Pascal

The interface file that is to be used when programming in ORCA/Pascal is named SpeechTools.Int. It is located in the Pascal folder of the Talking Tools disk. The simplest way to access this file is to move it to the LIBRARIES/ORCAPascalDefs folder of your ORCA/Pascal disk. The source code for the interface file is also in the Pascal folder on the Talking Tools disk, in a file named SpeechTools.pas. You may want to copy this file to the TOOL.INTERFACE folder on the ORCA/Pascal Extras disk, where the source code for the other interface files is found.

If you moved the interface file to your ORCAPascalDefs folder of your libraries prefix, you can access the interface file in a program with a simple USES statement:

```
uses SpeechTools;
```

Unlike the interfaces to Apple's toolbox, where each tool has a separate interface file, this one interface file contains the interface for all four of the speech tools.

If the interface file is not in your libraries prefix, you can use the LibPrefix directive to tell the compiler where the file resides, followed by a USES statement. In the example below, assume that the interface file resides in the same folder as your program:

```
{$LibPrefix '0/'}
uses SpeechTools;
```

Throughout this chapter, you will see references to "pString32" and "pString255" data structures. These are "Pascal-style" strings (contain a leading length byte) of 32 and 255 characters, respectively. The pString types are defined in the SpeechTools interface file. You may recall that in ORCA/Pascal, these kinds of strings are declared as a packed array of characters, indexed from 0 to the maximum number of characters. You will also see references to the data types Gender, Tone, and ParmRange. Gender and Tone refer to enumerated types, while ParmRange refers to an integral subrange. For your convenience, the type definitions from the SpeechTools interface file are shown here:

```
type
   pString32 = packed array[0..32] of char;
   pString32Ptr = ^pString32;

   Gender = ( Male, Female );
   Tone = ( Treble, Bass );

   ParmRange = 0..9;
```

Using Talking Tools With C

The interface file that is to be used when programming in ORCA/C is named speech.h. It is located in the CLang folder of the Talking Tools disk. The simplest way to access this file is to move it to the LIBRARIES/ORCACDefs folder of your ORCA/C disk.

You can access the interface file from a program with an include statement:

```
#include <speech.h>        /* interface file in libraries prefix */
#include "/path/speech.h"  /* interface file in path prefix      */
```

Unlike the interfaces to Apple's toolbox, where each tool has a separate interface file, this one interface file contains the interface for all four of the speech tools.

In the C language source code in this chapter, you will see references to "pString32Ptr" and "pString255Ptr" data types. These are pointers to "Pascal-style" strings (containing a leading length byte) of 32 and 255 characters, respectively. You will also see the data types of Gender and Tone, which are enumerations. These data types are defined in the speech.h interface file, reprinted below for your convenience:

typedef char pString32 [33], *pString32Ptr;

```
typedef char pString255 [256], *pString255Ptr;
typedef enum Gender { Male, Female } Gender;
typedef enum Tone   { Treble, Bass } Tone;
```

Using Talking Tools With Assembly Language

The macros that can be used to call the speech tool kit are in a file named M16.SPEECH, located in the Asm folder on the Talking Tools disk. You may want to copy this file to the LIBRARIES/AInclude folder on your ORCA/M disk, where all of the other macro files are located. In order to use the speech tool kit from assembly language, you would typically use the MacGen utility in ORCA/M to create a custom macro file for your program, having it scan the M16.SPEECH file for the tool kit's macros. At the beginning of the program, you would then include an MCOPY directive to access the macros:

```
        mcopy    myMacros
```

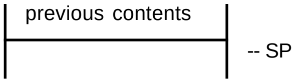
Speech Tool Set

The speech tool set provides an interface between the speech tools provided by First Byte, Inc., and the latest version of the GS/OS operating system, correcting minor incompatibilities that crop up when the First Byte tools are used alone.

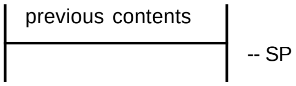
$0135 SpeechBootInit

Called by the Tool Locator to initialize the tool. An application should never make this call.

Stack before call



Stack after call



Pascal: `procedure SpeechBootInit;`

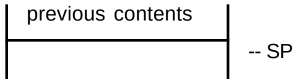
C: `extern pascal void SpeechBootInit (void);`

Assembly: `_SpeechBootInit`

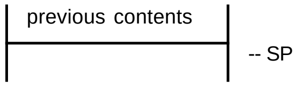
$0235 SpeechStartUp

Initializes the Talking Tools speech tool sets. This call should be made before any other tool calls to the Speech tool set, and the call should be made only once. The call should be made after starting the other Talking Tools tool sets.

Stack before call



Stack after call



Pascal: `procedure SpeechStartUp;`

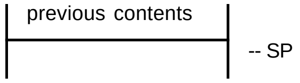
C: `extern pascal void SpeechStartUp (void);`

Assembly: `_SpeechStartUp`

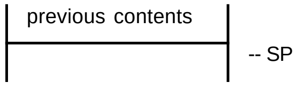
$0335 SpeechShutDown

Shuts down the Talking Tools tool sets. This call should be made after all other Speech tool calls have been made, and should be made only once in a program. The call should be made after shutting down the other Talking Tools tool sets.

Stack before call



Stack after call



Pascal: `procedure SpeechShutDown;`

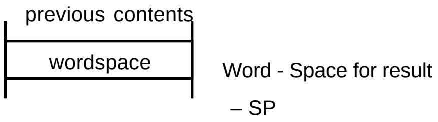
C: `extern pascal void SpeechShutDown (void);`

Assembly: `_SpeechShutDown`

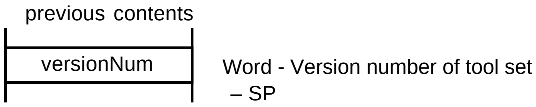
$0435 SpeechVersion

Returns the version number of the Speech tool set installed in the TOOLS folder of the SYSTEM folder on the boot disk.

Stack before call



Stack after call



Pascal: `function SpeechVersion: integer;`

C: `extern pascal int SpeechVersion (void);`

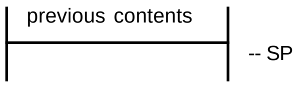
Assembly:

```
pha
_SpeechVersion
pl2    versionNum
```

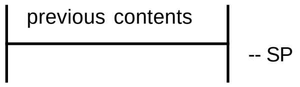
$0535 SpeechReset

Reinitializes the Speech tool set to its original state.

Stack before call



Stack after call



Pascal: `procedure SpeechReset;`

C: `extern pascal void SpeechReset (void);`

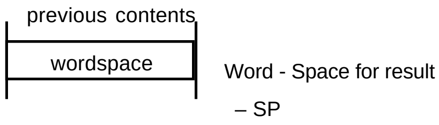
Assembly:

```
_SpeechReset
```

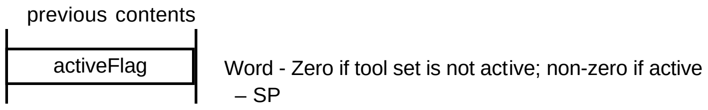
$0635 SpeechStatus

Indicates whether the Speech tool set is currently active.

Stack before call



Stack after call



Pascal: `function SpeechStatus: integer;`

C: `extern pascal int SpeechStatus (void);`

Assembly:

```
pha
_SpeechStatus
pla
beq     NotActive
bne     Active
```

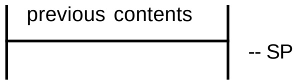
Parser Speech Tool Set

The Parser tool handles the exceptions dictionary, and can convert English text to phonetics, either returning the phonetic string or passing it on to one of the other tools to say the text right away.

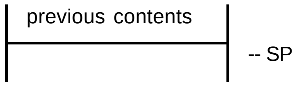
$0134 ParseBootInit

Called by the Tool Locator to initialize the tool. An application should never make this call.

Stack before call



Stack after call



Pascal: `procedure ParseBootInit;`

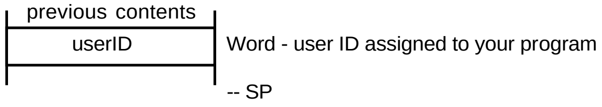
C: `extern pascal void ParseBootInit (void);`

Assembly: `_ParseBootInit`

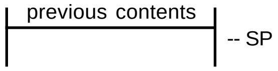
$0234 ParseStartUp

Initializes the Talking Tools front-end, its parser module. This call should be made before any other tool calls to the Parser tool set, and the call should be made only once. The global speech parameters are set as follows: voice gender is set to male; tone is set to bass; speed, volume, and pitch are each set to 5. The exceptions dictionary is initialized to empty.

Stack before call



Stack after call



Pascal:

```
procedure ParseStartUp (myUserID: integer);
```

C:

```
extern pascal void ParseStartUp (int myUserID);
```

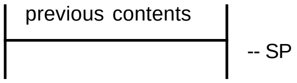
Assembly:

```
ph2     myUserID
_ParseStartUp
```

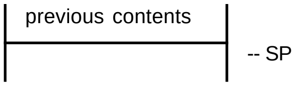
$0334 ParseShutDown

Shuts down the Parser module. This call should be made after all other Parser tool calls have been made, and should be made only once in a program. The call causes all memory allocated by the parser to be released.

Stack before call



Stack after call



Pascal: `procedure ParseShutDown;`

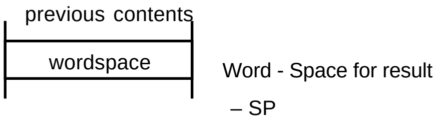
C: `extern pascal void ParseShutDown (void);`

Assembly: `_ParseShutDown`

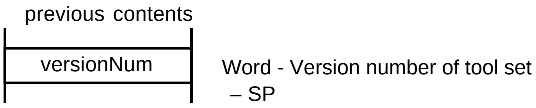
$0434 ParseVersion

Returns the version number of the Parser tool set installed in the TOOLS folder of the SYSTEM folder on the boot disk.

Stack before call



Stack after call



Pascal:

```
function ParseVersion: integer;
```

C:

```
extern pascal int ParseVersion (void);
```

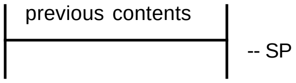
Assembly:

```
pha
_ParseVersion
pl2    versionNum
```

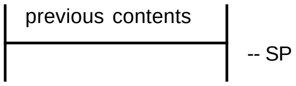
$0534 ParseReset

Reinitializes the parser module to its original state.

Stack before call



Stack after call



Pascal: `procedure ParseReset;`

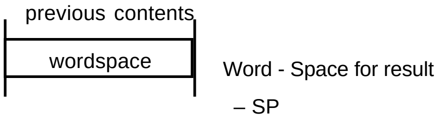
C: `extern pascal void ParseReset (void);`

Assembly: `_ParseReset`

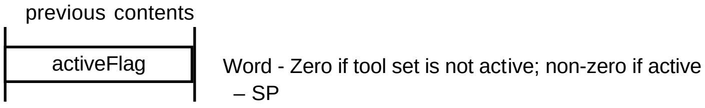
$0634 ParseStatus

Indicates whether the Parser tool set is currently active.

Stack before call



Stack after call



Pascal: `function ParseStatus: integer;`

C: `extern pascal int ParseStatus (void);`

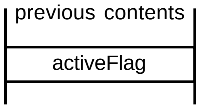
Assembly:

```
pha
_ParseStatus
pla
beq     NotActive
bne     Active
```

$1034 DictActivate

Activates or deactivates the dictionary. A parameter value of 0 turns off the exceptions dictionary, so that subsequent calls to Say and Parse will not consult the dictionary for special pronunciations. A parameter of 1 turns the exception dictionary back on.

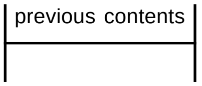
Stack before call



Word - activate or deactivate the dictionary?

-- SP

Stack after call



-- SP

Pascal:

```
procedure DictActivate (activeFlag: integer);
```

C:

```
extern pascal void DictActivate (int activeFlag);
```

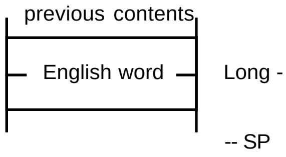
Assembly:

```
ph2    #activeFlag
_DictActivate
```

$0B34 DictDelete

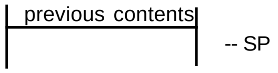
Removes an entry from the parser's on-line dictionary. Subsequent Say and Parse calls will use the phonetic translation generated by the parser for the deleted English word, rather than that which had been associated with the word in the dictionary.

Stack before call



Long - pointer to English word to remove from dictionary

Stack after call



Pascal: `procedure DictDelete (EnglishWord: pString32);`

C: `extern pascal void DictDelete (pString32Ptr EnglishWord);`

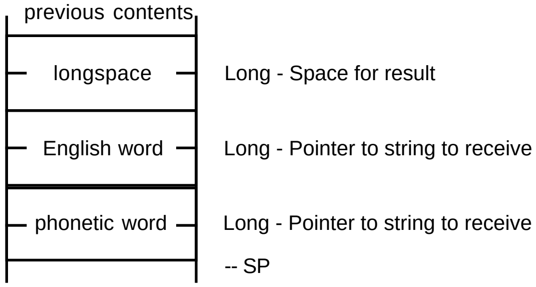
Assembly:

```
ph4    #EnglishWord
_DictDelete
```

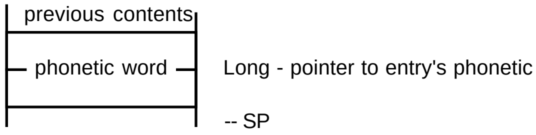
$0C34 DictDump

Returns the next dictionary entry, advancing the pointer into the dictionary. A dictionary entry consists of an English word and a phonetic representation that is to be used for the word whenever the parser is called. The dictionary is stored as a list of entries in memory; the entries are in alphabetical order. The parser maintains a pointer to the current dictionary entry. After DictDump has returned the last entry in the list, subsequent calls will return a null string. To start the process over, you can call DictInit, passing it a flag of zero, to reset the dictionary pointer to the beginning of the dictionary. You should pass DictDump pointers to two 33-byte (first byte reserved for length) arrays in which it will place the English word and its phonetic representation. The call returns a pointer to the entry's phonetic representation.

Stack before call



Stack after call



Pascal:

```
function DictDump (var EnglishWord, phoneticWord: pString32):
       pString32Ptr;
```

C:

```
extern pascal pString32Ptr DictDump (pString32Ptr EnglishWord,
       pString32Ptr phoneticWord);
```

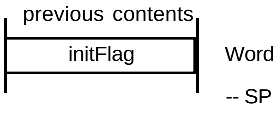
Assembly:

```
pha
pha
ph4    #EnglishWord
ph4    #phoneticWord
_DictDump
pl4    entryPointer
```

$0E34 DictInit

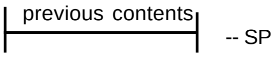
Initializes the parser's on-line dictionary. DictInit accepts an integer parameter; a value of zero resets the entry pointer to the first entry in the dictionary, while a value of one deletes the current dictionary from memory. Calls to the DictDump function, after clearing the dictionary from memory, will return the null string.

Stack before call



Word - Flag specifying how to initialize dictionary: 0 = set to top; 1 = clear from memory

Stack after call



Pascal: `procedure DictInit (flag: integer);`

C: `extern pascal void DictInit (int flag);`

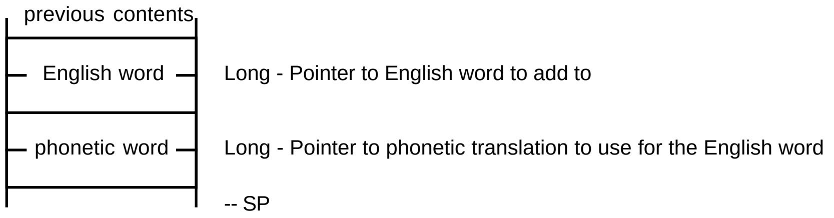
Assembly:

```
ph2     flag
_DictInit
```

$0A34 DictInsert

Adds a new entry to the current dictionary, inserting it in the correct place in order to preserve the alphabetical ordering of the dictionary. Subsequent Say and Parse calls will use the dictionary's phonetic representation for the new English word rather than the one which would normally be generated by the parser.

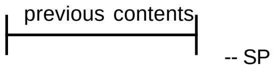
Stack before call



Long - Pointer to English word to add to

Long - Pointer to phonetic translation to use for the English word

Stack after call



Pascal:

```
procedure DictInsert (EnglishWord, phoneticWord: pString32);
```

C:

```
extern pascal void DictInsert (pString32Ptr EnglishWord,
       pString32Ptr phoneticWord);
```

Assembly:

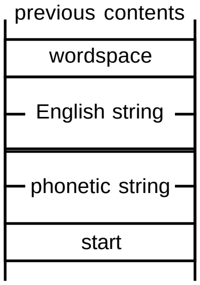
```
ph4    #EnglishWord
ph4    #phoneticWord
_DictInsert
```

$0934 Parse

Generates a sequence of phonemes for a given string. The string can contain English text consisting of letters, numbers, and abbreviations recognized by the parser, as well as a command string of <<COMMANDS text>> or a phonetic string encoded as <<~phonetics>>. Refer to Chapter 2 for more information about the phonetics generated and for the syntax of the command string.

Parse returns the position of the last character in the English string that it processed, counting the characters from position 1. You can check to see if all of the English string was parsed by checking whether the returned value is equal to the length of the English string. In the event that characters remain to be parsed, you should increment the start parameter and then call Parse again to obtain the next group of phonetics.

Stack before call



Word - Space for result

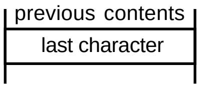
Long - Pointer to English string to convert to phonetics

Long - Pointer to string to receive phonetics

Word - Character position in English string to begin conversion, counting from 1

-- SP

Stack after call



Word - Position of last character in English string to be processed

-- SP

Pascal:

```
function Parse (EnglishString: pString255;
       var phoneticString: pString255; start: integer): integer;
```

C:

```
extern pascal int Parse (pString255Ptr EnglishString,
       pString255Ptr phoneticString, int start);
```

Assembly:

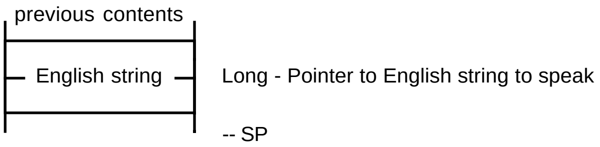
```
pha
ph4    #EnglishString
ph4    #phoneticString
ph2    startPos
_Parse
pl2    startPos
```

$0F34 Say

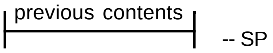
Speaks an English string by first calling the parser to generate phonetics for the string, and then passes the phonetics with the current global speech parameters to the currently active male or female voice in order to generate speech. You can use the SetSayGlobals tool call to change the global speech values used by Say.

The input string can contain English text consisting of letters, numbers, and abbreviations recognized by the parser, as well as a command string of <<COMMANDS text>> or a phonetic string encoded as <<~phonetics>>. Refer to Chapter 2 for more information about the phonetics generated and for the syntax of the command string.

Stack before call



Stack after call



Pascal: `procedure Say (EnglishString: pString255);`

C: `extern pascal void Say (pString255Ptr EnglishString);`

Assembly:

```
ph4     #EnglishString
_Say
```

$0D34 SetSayGlobals

Sets the global speech parameters applied during parsing.

Stack before call

previous contents	
voice	Word - Say voice to use, either adult male (0) or adult female (1)
tone	Word - Tone for voice, either bass (0) or treble (1)
pitch	Word - Pitch of voice, from 0 (low) to 9 (high)
speed	Word - Speed of speech, from 0 (slow) to 9 (fast)
volume	Word - Volume of speech, from 0 (soft) to 9 (loud)
	-- SP

Stack after call

previous contents	
	-- SP

Errors: $3403 Parameter out of range: invalid value passed to function

Pascal:

```
procedure SetSayGlobals (voice: Gender; basePitch: Tone;
       pitch, speed, volume: ParmRange);
```

C:

```
extern pascal void SetSayGlobals (Gender voice, Tone basePitch,
       int pitch, int speed, int volume);
```

Assembly:

```
ph2    voice
ph2    tone
ph2    pitch
ph2    speed
ph2    volume
_SetSayGlobals
```

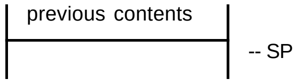
Male Speech Tool Set

The Male Speech tool takes phonetic strings as input and creates sound with a male voice.

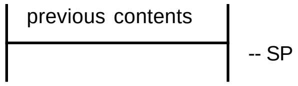
$0132 MaleBootInit

Called by the Tool Locator to initialize the tool. An application should never make this call.

Stack before call



Stack after call



Pascal: `procedure MaleBootInit;`

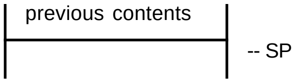
C: `extern pascal void MaleBootInit (void);`

Assembly: `_MaleBootInit`

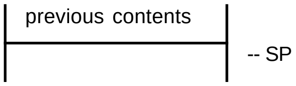
$0232 MaleStartUp

Initializes the Male Speech tool set. This call should be made before any other calls to the Male Speech tool set, and should only be made once in a program.

Stack before call



Stack after call



Pascal: `procedure MaleStartUp;`

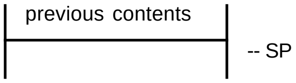
C: `extern pascal void MaleStartUp (void);`

Assembly: `_MaleStartUp`

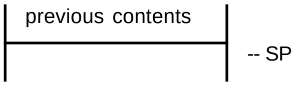
$0332 MaleShutDown

Shuts down the Male Speech tool set. This call should be made after all other Male Speech tool calls have been made, and should be made only once in a program.

Stack before call



Stack after call



Pascal: `procedure MaleShutDown;`

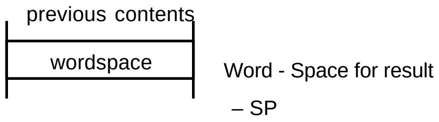
C: `extern pascal void MaleShutDown (void);`

Assembly: `_MaleShutDown`

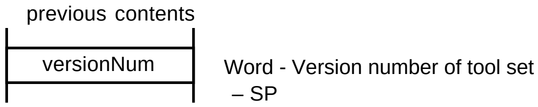
$0432 MaleVersion

Returns the version number of the Male Speech tool set currently installed in the SYSTEM/TOOLS folder of the boot disk.

Stack before call



Stack after call



Pascal:

```
function MaleVersion: integer;
```

C:

```
extern pascal int MaleVersion (void);
```

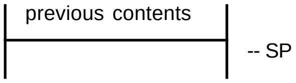
Assembly:

```
pha
_MaleVersion
pl2    versionNum
```

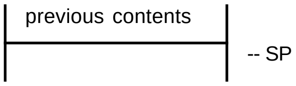
$0532 MaleReset

Restores the Male Speech tool set to its original state.

Stack before call



Stack after call



Pascal: `procedure MaleReset;`

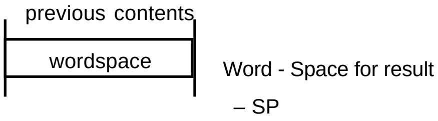
C: `extern pascal void MaleReset (void);`

Assembly: `_MaleReset`

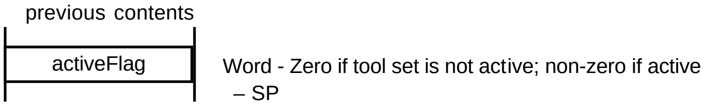
$0632 MaleStatus

Indicates whether the Male Speech tool set is currently active.

Stack before call



Stack after call



Pascal: `function MaleStatus: integer;`

C: `extern pascal int MaleStatus (void);`

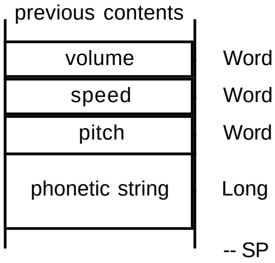
Assembly:

```
pha
_MaleStatus
pla
beq     NotActive
bne     Active
```

$0932 MaleSpeak

Uses the Apple IIGS' ENSONIQ chip to generate speech from a phonetic string. The voice used is adult male, and the passed speech parameters of volume, speed, and pitch are applied to the speech as well.

Stack before call



Word - Volume of speech, from 0 (soft) to 9 (loud)

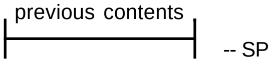
Word - Speed of speech, from 0 (slow) to 9 (fast)

Word - Pitch of voice, from 0 (low) to 9 (high)

Long - Pointer to phonetic string to speak

-- SP

Stack after call



-- SP

Pascal:

```
procedure MaleSpeak (volume, speed, pitch: ParmRange;
       phoneticString: pString255);
```

C:

```
extern pascal void MaleSpeak (int volume, int speed, int pitch,
       pString255Ptr phoneticString);
```

Assembly:

```
ph2    volume
ph2    speed
ph2    pitch
ph4    #phoneticString
_MaleSpeak
```

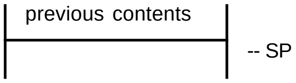
Female Speech Tool Set

The Female Speech tool takes phonetic strings as input and creates sound with a female voice.

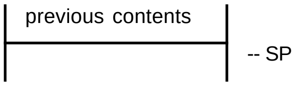
$0133 FemaleBootInit

Called by the Tool Locator to initialize the tool. An application should never make this call.

Stack before call



Stack after call



Pascal: `procedure FemaleBootInit;`

C: `extern pascal void FemaleBootInit (void);`

Assembly: `_FemaleBootInit`

$0233 FemaleStartUp

Initializes the Female Speech tool set. This call should be made before any other calls to the Female Speech tool set, and should only be made once during a program.

Stack before call

previous contents

-- SP

Stack after call

previous contents

-- SP

Pascal: `procedure FemaleStartUp;`

C: `extern pascal void FemaleStartUp (void);`

Assembly: `_FemaleStartUp`

$0333 FemaleShutDown

Shuts down the Female Speech tool set. This call should be made after all other Female Speech tool calls have been made, and should be made only once in a program.

Stack before call

previous contents

-- SP

Stack after call

previous contents

-- SP

Pascal: `procedure FemaleShutDown;`

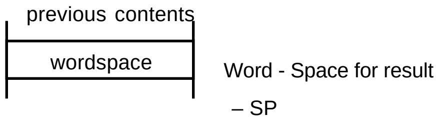
C: `extern pascal void FemaleShutDown (void);`

Assembly: `_FemaleShutDown`

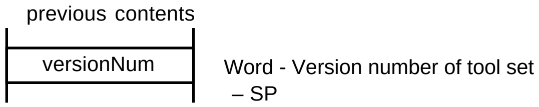
$0433 FemaleVersion

Returns the version number of the Female Speech tool set currently installed in the SYSTEM/TOOLS folder of the boot disk.

Stack before call



Stack after call



Pascal:

```
function FemaleVersion: integer;
```

C:

```
extern pascal int FemaleVersion (void);
```

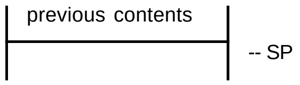
Assembly:

```
pha
_FemaleVersion
pl2    versionNum
```

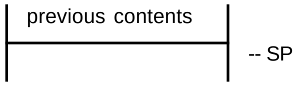
$0533 FemaleReset

Restores the Female Speech tool set to its original state.

Stack before call



Stack after call



Pascal: `procedure FemaleReset;`

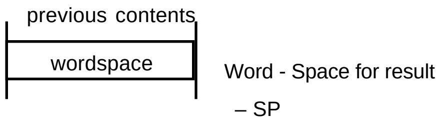
C: `extern pascal void FemaleReset (void);`

Assembly: `_FemaleReset`

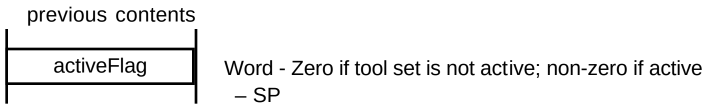
$0633 FemaleStatus

Indicates whether the Female Speech tool set is currently active.

Stack before call



Stack after call



Pascal: `function FemaleStatus: integer;`

C: `extern pascal int FemaleStatus (void);`

Assembly:

```
pha
_FemaleStatus
pla
beq    NotActive
bne    Active
```

$0933 FemaleSpeak

Uses the Apple IIGS' ENSONIQ chip to generate speech from a phonetic string. The voice used is adult female, and the passed speech parameters of volume, speed, and pitch are applied to the speech as well.

Stack before call

previous contents

volume	Word - Volume of speech, from 0 (soft) to 9 (loud)
speed	Word - Speed of speech, from 0 (slow) to 9 (fast)
pitch	Word - Pitch of voice, from 0 (low) to 9 (high)
phonetic string	Long - Pointer to phonetic string to speak

-- SP

Stack after call

previous contents

-- SP

Pascal:

```
procedure FemaleSpeak (volume, speed, pitch: ParmRange;
       phoneticString: pString255);
```

C:

```
extern pascal void FemaleSpeak (int volume, int speed,
       int pitch, pString255Ptr phoneticString);
```

Assembly:

```
ph2    volume
ph2    speed
ph2    pitch
ph4    #phoneticString
_FemaleSpeak
```

Appendix A - Complete Speak Program

This appendix contains the complete source code for the Speak program described in Chapter 3. The complete source code for the program can also be found on the Talking Tools disk. Please keep in mind that it is easier for us to change the disk than it is for us to change the manual. As the speech tools mature and the product improves, there may be minor differences between the source code presented here and that which appears on the disk.

Speak In Pascal

```
{---------------------------------------------------------------}
{                                                               }
{  Speak                                                        }
{                                                               }
{  A "plain vanilla" program that demonstrates all of the calls }
{  available in the Speech Tool kit.                            }
{                                                               }
{  By Barbara Allred                                            }
{                                                               }
{  Copyright 1991 by Byte Works, Inc.                           }
{  Copyright 1987 - 1991 by First Byte, Inc.                    }
{                                                               }
{---------------------------------------------------------------}

{$keep 'Speak'}

program Speak (input, output);

uses Common, ToolLocator, IntegerMath, SpeechTools;

var
   answer: string;                      {user's response to queries}
   done: boolean;                       {true if user wants to exit pgm}
   toolRec: toolTable;                  {table of tools we need to start}

   voice: Gender;                       {current global voice setting}
   basePitch: Tone;                     {current global tone setting}
   speed,                               {current global speed setting}
   pitch,                               {current global pitch setting}
   volume: ParmRange;                   {current global volume setting}


{---------------------------------------------------------------}
{                                                               }
{  ConvertToPhonetics - Convert English text to phonetic        }
{       representation.                                         }
{                                                               }
{---------------------------------------------------------------}

procedure ConvertToPhonetics;

var
   phString: packed array[0..255] of char; {phonetic string}
   sayString: packed array[0..255] of char; {English text to speak or parse}
   size: integer;                       {length of string}
   start: integer;                      {position in English string to begin}
                                        { conversion                        }
   stop: boolean;                       {true if user wants to exit}

begin {ConvertToPhonetics}
stop := false;
```

```
{Outer loop lets user enter next string to convert.}
{Entering null string signals it's time to exit.}
repeat
   writeln;
   writeln('Enter string to translate to phonetics.  Press RETURN to exit.');
   readln(sayString);
   size  := length(sayString);
   start := 0;

   {Inner loop is necessary in the event that the }
   { complete English string wasn't converted.    }
   if size > 0 then begin
      repeat
         start := start+1;
         start := Parse(sayString, phString, start);
         write(phString);
      until start = size;
      writeln;
      end {if}
   else
      stop := true;
until stop;
end; {ConvertToPhonetics}


{-------------------------------------------------------------}
{                                                             }
{  DeleteWord - Removes entries from the current exceptions   }
{       dictionary.                                           }
{                                                             }
{-------------------------------------------------------------}

procedure DeleteWord;

var
   stop: boolean;                      {true if user wants to exit}
   word: pString32;                    {dictionary entry}

begin {DeleteWord}
stop := false;
writeln;
repeat
   writeln('Press RETURN for the dictionary entry to exit function.');
   write('Word to delete from dictionary?  ');
   readln(word);
   if length(word) = 0 then
      stop := true
   else
      DictDelete(word);
   writeln;
until stop;
writeln;
end; {DeleteWord}


{-------------------------------------------------------------}
{                                                             }
{  DisplayDict - Displays current exceptions dictionary, one  }
{       entry at a time.                                      }
{                                                             }
{-------------------------------------------------------------}

procedure DisplayDict;

var
   answer: string;                     {user's response to queries}
   flag: integer;                      {dict. initialization flag}
   noErr: boolean;                     {true if no error has occurred}
   stop: boolean;                      {true if user wants to exit}
   word1, word2: pString32;            {dictionary entry}
   wordPtr: pString32Ptr;              {pointer returned by DictDump function}
```

```
begin {DisplayDict}
writeln;

{Before displaying the dictionary, let user initialize it.}
repeat
   noErr := true;
   writeln('Before displaying the dictionary, lets initialize it.');
   writeln('Enter 0 to reset dictionary to beginning.');
   writeln('Enter 1 to delete current dictionary.');
   writeln('Enter 2 to NOT initialize dictionary.');
   writeln;
   readln(flag);
   if (flag < 0) or (flag > 2) then begin
       noErr := false;
       writeln;
       writeln('Please enter either 0, 1, or 2.');
       writeln;
       end; {if}
until noErr;

if flag <> 2 then
   DictInit(flag);

{While there are still entries in the dictionary, }
{ get and then display the next entry.            }
stop := false;
repeat
   wordPtr := DictDump(word1, word2);
   if length(word1) = 0 then
      stop := true
   else begin
      writeln('Next entry:   ', word1, '   ', word2, '   Continue? (Y or N)');
      readln(answer);
      if (answer[1] = 'N') or (answer[1] = 'n') then
         stop := true;
      end; {else}
until stop;
writeln;
end; {DisplayDict}


{---------------------------------------------------------------}
{                                                               }
{  Init - Load the tools we need and initialize our data        }
{       structures.                                             }
{                                                               }
{---------------------------------------------------------------}

procedure Init;

var
   errNum: integer;                     {error number to report to user}
   errString: packed array[1..5]   of char; {error number as a hex string}

begin {Init}
errString[1] := '$';                    {return error codes as hex numbers}
with toolRec do begin
   numToolsRequired := 4;
   with tool[1] do begin
      toolNumber := maleToolNum;
      minVersion := 0;
      end; {with}
   with tool[2] do begin
      toolNumber := femaleToolNum;
      minVersion := 0;
      end; {with}
   with tool[3] do begin
      toolNumber := parserToolNum;
      minVersion := 0;
      end; {with}
   with tool[4] do begin
      toolNumber := speechToolNum;
      minVersion := 0;
      end; {with}
   end; {with}
LoadTools(toolRec);                     {load the tools}
errNum := toolError;
if errNum <> 0 then begin               {report any error returned}
   Int2Hex(errNum, pointer(@errString[2]), 4);
   writeln('Unable to load tools: Error = ', errString);
   done := true;
   end {if}
else begin                              {start the tools}
   ParseStartUp(userID);
   MaleStartUp;
   FemaleStartUp;
   SpeechStartUp;
   done := false;                       {initialize globals}
   voice := male;                       {these are the default settings for}
   basePitch := bass;                   {the global speech parameters}
   speed := 5;
   volume := 5;
   pitch := 5;
   end; {else}
end; {Init}


{---------------------------------------------------------------}
{                                                               }
{  InsertWord - Insert new entries into exceptions dictionary.  }
{                                                               }
{---------------------------------------------------------------}

procedure InsertWord;

var
   stop: boolean;                       {true if user wants to exit}
   word1, word2: pString32;             {dictionary entry}

begin {InsertWord}
stop := false;
writeln;
repeat
   writeln('Press RETURN for the dictionary entries to exit function.');
   write('Enter English word to add to dictionary:  ');
   readln(word1);
   write('Enter phonetic representation of word to add to dictionary:  ');
   readln(word2);
   if (length(word1) = 0) or (length(word2) = 0) then
      stop := true
   else
      DictInsert(word1, word2);
   writeln;
until stop;
writeln;
end; {InsertWord}


{---------------------------------------------------------------}
{                                                               }
{  LoadDict - Load dictionary file from disk.                   }
{                                                               }
{---------------------------------------------------------------}

procedure LoadDict;

label 99;

var
   ch: char;                            {character from the file}
   errNum: integer;                     {error number to report to user}
   errString: packed array[1..5]   of char; {error number as a hex string}
   f: text;                            {file variable}
   i: integer;                         {loop/index variable}
   pathname: string[255];              {name of the file}
   word1, word2: pString32;            {dictionary entry}

begin {LoadDict}
{Get pathname of dictionary to open.}
write('Enter pathname of dictionary to open:  ');
readln(pathname);
if length(pathname) = 0 then
   goto 99;

{Open the file for reading.}
reset(f, pathname);
errNum := toolError;
if errNum <> 0 then begin              {report any error returned}
   Int2Hex(errNum, pointer(@errString[2]), 4);
   writeln('Unable to open file:  Error = ', errString);
   goto 99;
   end; {if}

{Build the dictionary from the file.}
DictInit(1);                           {clear current dict from memory}
while not (eof(f)) do begin            {Loop:}
   if eoln(f) then
      readln(f)
   else begin
      read(f, ch);                     {read English word}
      word1[0] := ch;
      for i := 1 to ord(ch) do
         read(f, word1[i]);
      read(f, ch);                     {read phonetic word}
      word2[0] := ch;
      for i := 1 to ord(ch) do
         read(f, word2[i]);
      DictInsert(word1, word2);        {insert entry into dict}
      end; {else}
   end; {while}
close(f);
DictInit(0);                           {reset dict to top}

99:
end; {LoadDict}


{-------------------------------------------------------------}
{                                                             }
{  SetSpeechGlobals - Set global speech parameters.           }
{                                                             }
{-------------------------------------------------------------}

procedure SetSpeechGlobals;


   function GetValue (min,max: integer): integer;

   { Get a value, making sure it is in the given range          }
   {                                                            }
   { Parameters:                                                }
   {    min - lowest allowed value                              }
   {    max - highest allowed value                             }
   {                                                            }
   { Returns: Value read                                        }

   var
      value: integer;                  {value read}

   begin {GetValue}
   repeat
      readln(value);
      if (value < min) or (value > max) then begin
         writeln('Please enter a value from ', min:1, ' to ', max:1, '.');
         writeln;
         write('  Value: ');
         end; {if}
   until (value >= min) and (value <= max);
   GetValue := value;
   end; {GetValue}


begin {SetSpeechGlobals}
write('Voice = ');                      {Read new global voice setting}
if voice = male then
   writeln('male ')
else
   writeln('female ');
writeln('Enter 0 to change voice to male, 1 to change voice to female.');
if GetValue(0,1) = 0 then
   voice := male
else
   voice := female;
writeln;                                {Read new global tone setting}
write('Tone = ');
if basePitch = bass then
   writeln('bass ')
else
   writeln('treble ');
writeln('Enter 0 to change tone to bass, 1 to change tone to treble.');
if GetValue(0,1) = 0 then
   basePitch := bass
else
   basePitch := treble;
writeln;                                {Read new global volume setting}
write('Volume = ', volume:1, '  ');
volume := GetValue(0,9);
writeln;                                {Read new global speed setting}
write('Speed = ', speed:1, '  ');
speed := GetValue(0,9);
writeln;                                {Read new global pitch setting}
write('Pitch = ', pitch:1, '  ');
pitch := GetValue(0,9);
                                        {set the globals}
SetSayGlobals(voice, basePitch, pitch, speed, volume);
writeln;
end; {SetSpeechGlobals}


{----------------------------------------------------------------}
{                                                                }
{  ShutDown - Shut down the tools we started; do any necessary   }
{       clean-up before exiting.                                 }
{                                                                }
{----------------------------------------------------------------}

procedure ShutDown;

begin {ShutDown}
FemaleShutDown;                         {shut down speech tools}
MaleShutDown;
ParseShutDown;
SpeechShutDown;
end; {ShutDown}


{----------------------------------------------------------------}
{                                                                }
{  SpeakPhonetics - Speak as many non-empty lines of phonetic    }
{       text as the user wants.                                  }
{                                                                }
{----------------------------------------------------------------}

procedure SpeakPhonetics;
```

```
var
   phString: packed array[0..255] of char; {phonetic string}
   stop: boolean;                       {true if user wants to exit}

begin {SpeakPhonetics}
stop := false;
repeat
   writeln;
   writeln('Enter phonetic string to speak.  Press RETURN to exit.');
   readln(phString);
   if length(phString) = 0 then
      stop := true
   else begin
      if voice = male then
         MaleSpeak(volume, speed, pitch, phString)
      else
         FemaleSpeak(volume, speed, pitch, phString);
      end; {else}
until stop;
writeln;
end; {SpeakPhonetics}


{-------------------------------------------------------------}
{                                                             }
{  SpeakText - Speak as many non-empty lines of English text as }
{       the user wants.                                       }
{                                                             }
{-------------------------------------------------------------}

procedure SpeakText;

var
   sayString: packed array[0..255] of char; {English text to speak or parse}
   stop: boolean;                       {true if user wants to exit}

begin {SpeakText}
stop := false;
repeat
   writeln;
   writeln('Enter string to speak.  Press RETURN to exit.');
   readln(sayString);
   if length(sayString) = 0 then
      stop := true
   else
      Say(sayString);
until stop;
writeln;
end; {SpeakText}

{-------------------------------------------------------------}
{                                                             }
{  WriteDict - Write dictionary to disk file.                 }
{                                                             }
{-------------------------------------------------------------}

procedure WriteDict;

label 99;

var
   errNum: integer;                     {error number to report to user}
   errString: packed array[1..5]   of char; {error number as a hex string}
   f: text;                             {file variable}
   i: integer;                          {loop/index variable}
   pathname: string[255];               {name of the file}
   stop: boolean;                       {true if user wants to exit}
   tmp: pString32Ptr;                   {pointer returned by DictDump}
   word1, word2: pString32;             {dictionary entry}
```

```
begin
{Get pathname for dictionary file.}
write('Enter pathname for dictionary file:  ');
readln(pathname);
if length(pathname) = 0 then
   goto 99;

{Open the file for writing.}
rewrite(f, pathname);
errNum := toolError;
if errNum <> 0 then begin               {report any error returned}
   Int2Hex(errNum, pointer(@errString[2]), 4);
   writeln('Unable to open file:  Error = ', errString);
   goto 99;
   end; {if}

{Write the dictionary to the file.}
DictInit(0);                            {set dictionary to top}
stop := false;
repeat                                  {Loop:}
   tmp := DictDump(word1, word2);       {get next dict entry}
   if length(word1) = 0 then
      stop := true
   else begin
      for i := 0 to length(word1) do    {write English word}
         write(f, word1[i]);
      for i := 0 to length(word2) do    {write phonetic word}
         write(f, word2[i]);
      end; {else}
until stop;
close(f);
DictInit(0);                            {reset dict to top}

99:
end; {WriteDict}

{----------------------------------------------------------------}
{                                                                }
{  Main program - Display "main menu" and call appropriate       }
{       function until user selects Quit.                        }
{                                                                }
{----------------------------------------------------------------}

begin
{Splash screen.}
writeln;
writeln('Speak - A demonstration of the Talking Tools');
writeln;
writeln('Please wait while we load the tools.');
writeln;

{Initialize the program.}
Init;

{Main loop: bring up menu; get user's selection; handle selection.}
while not done do begin
   writeln('Enter desired function:  S to speak English string');
   writeln('                         P to speak phonetic string');
   writeln('                         C to convert to phonetics');
   writeln('                         G to set global speech parameters');
   writeln('                         A to activate dictionary');
   writeln('                         T to deactivate dictionary');
   writeln('                         D to display dictionary');
   writeln('                         I to insert word into dictionary');
   writeln('                         R to remove word from dictionary');
   writeln('                         L to load dictionary from disk file');
   writeln('                         W to write dictionary to disk file');
   writeln('                         Q to quit program');
   write('                         ');
   readln(answer);
   case answer[1] of
      'S', 's': SpeakText;
      'P', 'p': SpeakPhonetics;
      'C', 'c': ConvertToPhonetics;
      'G', 'g': SetSpeechGlobals;
      'A', 'a': DictActivate(1);
      'T', 't': DictActivate(0);
      'D', 'd': DisplayDict;
      'I', 'i': InsertWord;
      'R', 'r': DeleteWord;
      'L', 'l': LoadDict;
      'W', 'w': WriteDict;
      'Q', 'q': done := true;
      otherwise: begin
                 writeln('Please enter one of S, P, C, G, A, T, D, I, R, L, W, or Q...');
                 writeln;
                 end;
   end; {case}
end; {while}

{Shut down the program.}
ShutDown;
end.
```

Speak In C

```
/*****************************************************************
*
*  Speak - A "plain vanilla" program that demonstrates all of the
*       calls available in the Speech Tool kit.
*
*  by Barbara Allred
*
*  Copyright 1991 by Byte Works, Inc.
*  Copyright 1987 - 1991 by First Byte, Inc.
*
*****************************************************************/

#pragma keep "speak"

#include <types.h>
#include <stdio.h>
#include <string.h>
#include <locator.h>
#include <errno.h>
#include <orca.h>
#include <speech.h>

#pragma lint -1

int done;                               /* true if user wants to exit pgm */

int speed;                              /* current global speed setting */
int pitch;                              /* current global pitch setting */
int volume;                             /* current global volume setting */
Gender voice;                           /* current global voice setting */
Tone basePitch;                         /* current global tone setting */

ToolTable toolRec = {                   /* table of tools we need to start*/
   4,                                   /*   # tools to load */
   {                                    /*   toolset #, min. version req. */
      {maleToolNum,0},
      {femaleToolNum,0},
      {parserToolNum,0},
      {speechToolNum,1}
   }
};
```

```
/****************************************************************
*
*  ConvertToPhonetics - Convert English text to phonetic representation.
*
****************************************************************/

void ConvertToPhonetics (void)

{
int start;                              /* position in English string to */
                                        /* begin conversion              */
int stop;                               /* true if user wants to exit */
int size;                               /* length of string */
pString255Ptr strPtr;                   /* pointer to Pascal-style string */
pString255 sayString;                   /* English text to speak or parse */
pString255 phString;                    /* phonetic string */

stop  = false;

/* Outer loop lets user enter next string to convert. */
/* Entering null string signals it's time to exit. */
do {
   printf("\nEnter string to translate to phonetics.  Press RETURN to exit.\n");
   fgets(sayString, 256, stdin);
   size = strlen(sayString);
   sayString[size-1] = '\0';           /* replace '\n' with '\0' */
   strPtr = c2pstr(sayString);
   start  = 0;

   /* Inner loop is necessary in the event that the */
   /* complete English string wasn't converted. */
   if (strlen(sayString) != 0) {
      do {
         ++start;
         start = Parse(strPtr, phString, start);
         printf("%p\n", phString);
         }
      while (start < size-1);
      }
   else
      stop = true;

   }
while (!stop);
} /* ConvertToPhonetics */

/****************************************************************
*
*  DeleteWord - Removes entries from the current exceptions dictionary.
*
****************************************************************/

void DeleteWord (void)

{
int size;                               /* length of string */
int stop;                               /* true if user wants to exit */
pString32 word;                         /* dictionary entry */


stop = false;
printf("\n");
do {
   printf("Press RETURN for the dictionary entry to exit function.\n");
   printf("Word to delete from dictionary?  ");
   fgets(word, 33, stdin);
   size = strlen(word);
   word[size-1] = '\0';                /* replace '\n' with '\0' */
   if (strlen(word) == 0)
      stop = true;
   else
```

```
      DictDelete(c2pstr(word));
   printf("\n");
   }
while (!stop);
printf("\n");
} /* DeleteWord */

/*****************************************************************
*
*  DisplayDict - Displays current exceptions dictionary, one
*       entry at a time.
*
*****************************************************************/

void DisplayDict (void)

{
int flag;                               /* dict. initialization flag */
int anErr;                              /* true if error has occurred */
int stop;                               /* true if user wants to exit */
char answer;                            /* user's response to queries */
pString32Ptr wordPtr;                   /* pointer to phonetic word */
pString32 word1, word2;                 /* dictionary entry */


/* Before displaying the dictionary, let user initialize it. */
anErr = false;
do {
   printf("\nBefore displaying the dictionary, lets initialize it.\n");
   printf("Enter 0 to reset dictionary to beginning.\n");
   printf("Enter 1 to delete current dictionary.\n");
   printf("Enter 2 to NOT initialize dictionary.\n\n");
   scanf(" %d%*[^\n]%*c", &flag);
   if ((flag < 0) || (flag > 2)) {
      anErr = true;
      printf("\n");
      printf("Please enter either 0, 1, or 2.\n\n");
      }
   }
while (anErr);
if (flag != 2)
   DictInit(flag);

/* While there are still entries in the dictionary, */
/* get and then display the next entry. */
stop = false;
do {
   wordPtr = DictDump(word1, word2);
   if (strlen(word1) == 0)
      stop = true;
   else {
      printf("Next entry:  %p    %p   Continue? (Y or N)\n", word1, word2);
      scanf(" %c%*[^\n]%*c", &answer);
      if ((answer == 'N') || (answer == 'n'))
         stop = true;
      }
   }
while (!stop);
printf("\n");
} /* DisplayDict */

/*****************************************************************
*
*  Init - Load the tools we need and initialize our data structures.
*
*****************************************************************/

void Init (void)

{
int errNum;                             /* error number to report to user */
```

```
LoadTools(&toolRec);                    /* load the tools */
errNum = toolerror();
if (errNum) {                           /* report any error returned */
   printf("Unable to load tools:  Error = %X\n", errNum);
   done = true;
   }
else {                                  /* start the speech tools */
   ParseStartUp(userid());
   MaleStartUp();
   FemaleStartUp();
   SpeechStartUp();

   done = false;                        /* initialize globals */
   voice = Male;                        /* these are the default settings for */
   basePitch = Bass;                    /*   the global speech parameters */
   speed = 5;
   volume = 5;
   pitch = 5;
   }
} /* Init */

/****************************************************************
*
*  InsertWord - Insert new entries into exceptions dictionary.
*
****************************************************************/

void InsertWord (void)

{
pString32Ptr wPtr;                      /* temp; for conversions */
int i;                                  /* index/loop variable */
int size;                               /* length of string */
int stop;                               /* true if user wants to exit */
pString32 word1, word2;                 /* dictionary entry */

stop = false;
do {
   printf("\nPress RETURN for the dictionary entries to exit function.\n");

   printf("Enter English word to add to dictionary:  ");
   fgets(word1, 33, stdin);
   size = strlen(word1);
   word1[size-1] = '\0';                /* replace '\n' with '\0' */
   wPtr = c2pstr(word1);
   for (i = 0; i <= size; i++)
      word1[i] = wPtr[i];

   printf("\nEnter phonetic representation of word to add to dictionary:  ");
   fgets(word2, 33, stdin);
   size = strlen(word2);
   word2[size-1] = '\0';
   wPtr = c2pstr(word2);
   for (i = 0; i <= size; i++)
      word2[i] = wPtr[i];

   if ((strlen(word1) == 0) || (strlen(word2) == 0))
      stop = true;
   else
      DictInsert(word1, word2);
   printf("\n");
   }
while (!stop);
printf("\n");
} /* InsertWord */

/****************************************************************
*
*  LoadDict - Load dictionary file from disk.
*
****************************************************************/
```

```
void LoadDict (void)

{
FILE *f;                                /* file variable */
char pathname[255];                     /* file name */
int i;                                  /* loop/index variable */
int size;                               /* length of string */
char ch;                                /* char read from file */
pString32 word1, word2;                 /* dictionary entry */

/* Get pathname of dictionary to open. */
printf("Enter pathname of dictionary to open:  ");
gets(pathname);

/* Open the file for reading. */
errno = 0;
f = fopen(pathname, "r");
if (errno) {                            /* report any error returned */
   printf("%s: %i\n", strerror(errno), errno);
   return;
   }

/* Build the dictionary from the file. */
DictInit(1);                            /* clear current dict from memory */
fscanf(f, "%c", &size);                 /* initial read before loop */
while (! (feof(f))) {                   /* Loop: */
   word1[0] = size;                     /*   read English word from file */
   for (i = 1; i <= size; i++) {
      fscanf(f, "%c", &ch);
      word1[i] = ch;
      }
   fscanf(f, "%c", &size);              /*   read phonetic word from file */
   word2[0] = size;
   for (i = 1; i <= size; i++) {
      fscanf(f, "%c", &ch);
      word2[i] = ch;
      }
   DictInsert(word1, word2);            /*   insert entry into dict */
   fscanf(f, "%c", &size);
   }
fclose(f);
DictInit(0);                            /* reset dict to top */
} /* LoadDict */

/****************************************************************
*
*  GetValue - Get a value, making sure it is in the given range
*
*  Inputs:
*       min - lowest allowed value
*       max - highest allowed value
*
*  Returns: Value read
*
****************************************************************/

int GetValue (int min, int max)

{
int value;                              /* value read */

do {
   scanf(" %d%*[^\n]%*c", &value);
   if ((value < min) || (value > max))
      printf("Please enter a value from %d to %d.\n\n  Value: ", min, max);
   }
while ((value < min) || (value > max));
return value;
} /* GetValue */
```

```
/****************************************************************
*
*  SetSpeechGlobals - Set global speech parameters.
*
****************************************************************/

void SetSpeechGlobals (void)

{
printf("Voice = ");                     /* Read new global voice setting */
if (voice == Male)
   printf("male\n");
else
   printf("female\n");
printf("Enter 0 to change voice to male, 1 to change voice to female.\n");
if (GetValue(0,1) == 0)
   voice = Male;
else
   voice = Female;
printf("\nTone = ");                    /* Read new global tone setting */
if (basePitch == Bass)
   printf("bass\n");
else
   printf("treble\n");
printf("Enter 0 to change tone to bass, 1 to change tone to treble.");
if (GetValue(0,1) == 0)
   basePitch = Bass;
else
   basePitch = Treble;
printf("\nVolume = %d  ", volume);      /* Read new global volume setting */
volume = GetValue(0,9);
printf("\nSpeed = %d  ", speed);        /* Read new global speed setting */
speed = GetValue(0,9);
printf("\nPitch = %d  ", pitch);        /* Read new global pitch setting */
pitch = GetValue(0,9);
                                        /* set the globals */
SetSayGlobals(voice, basePitch, pitch, speed, volume);
putchar('\n');
} /* SetSpeechGlobals */

/****************************************************************
*
*  ShutDown - Shut down the tools we started; do any necessary
*       clean-up before exiting.
*
****************************************************************/

void ShutDown (void)

{
MaleShutDown();
FemaleShutDown();
ParseShutDown();
SpeechShutDown();
} /* ShutDown */

/****************************************************************
*
*  SpeakPhonetics - Speak as many non-empty lines of phonetic
*       text as the user wants.
*
****************************************************************/

void SpeakPhonetics (void)

{
int size;                               /* length of string */
int stop;                               /* true if user wants to exit */
pString255 phString;                    /* phonetic string */

stop = false;
```

```
do {
   printf("\nEnter phonetic string to speak.  Press RETURN to exit.\n");
   fgets(phString, 256, stdin);
   size = strlen(phString);
   phString[size-1] = '\0';
   if (strlen(phString) == 0)
      stop = true;
   else {
      if (voice == Male)
         MaleSpeak(volume, speed, pitch, c2pstr(phString));
      else
         FemaleSpeak(volume, speed, pitch, c2pstr(phString));
      }
   }
while (!stop);
printf("\n");
} /* SpeakPhonetics */

/****************************************************************
*
*  SpeakText - Speak as many non-empty lines of English text as
*       the user wants.
*
****************************************************************/

void SpeakText (void)

{
int size;                               /* length of string */
int stop;                               /* true if user wants to exit */
pString255 sayString;                   /* English text to speak or parse */

stop = false;
do {
   printf("\nEnter string to speak.  Press RETURN to exit.\n");
   fgets(sayString, 256, stdin);
   size = strlen(sayString);
   sayString[size-1] = '\0';
   if (sayString[0] == '\0')
      stop = true;
   else
      Say(c2pstr(sayString));
   }
while (!stop);
printf("\n");
} /* SpeakText */

/****************************************************************
*
*  WriteDict - Write dictionary to disk file.
*
****************************************************************/

void WriteDict (void)

{
FILE *f;                                /* file variable */
char pathname[255];                     /* file name */
int i;                                  /* loop/index variable */
int stop;                               /* true if user wants to exit */
pString32Ptr tmp;                       /* returned by DictDump */
pString32 word1, word2;                 /* dictionary entry */

/* Get pathname for dictionary file. */
printf("Enter pathname for dictionary file:  ");
gets(pathname);

/* Open the file for writing. */
f = fopen(pathname, "w");
if (errno) {                            /* report any error returned */
   printf("%s %i\n", strerror(errno), errno);
   return;
   }

/* Write the dictionary to the file. */
DictInit(0);                            /* set dictionary to top */
stop = false;
do {                                    /* Loop: */
   tmp = DictDump(word1, word2);        /*   get next dict entry */
   if (tmp == NULL)
      stop = true;
   else {
      for (i = 0; i < strlen(word1); i++)
         fprintf(f, "%c", word1[i]);
      for (i = 0; i < strlen(word2); i++)
         fprintf(f, "%c", word2[i]);
      }
   }
while (!stop);
fclose(f);
DictInit(0);                            /* reset dict to top */
} /* WriteDict */

/****************************************************************
*
*  Main program - Display "main menu" and call appropriate
*       function until user selects Quit.
*
****************************************************************/

void main (void)

{
char answer;                            /* user's response to queries */

printf("\nSpeak - A demonstration of the Speech Toolkit.\n\n");
printf("Please wait while we load the tools.\n");

Init();
while (!done) {
   printf("Enter desired function:  S to speak English string\n");
   printf("                         P to speak phonetic string\n");
   printf("                         C to convert to phonetics\n");
   printf("                         G to set global speech parameters\n");
   printf("                         A to activate dictionary\n");
   printf("                         T to deactivate dictionary\n");
   printf("                         D to display dictionary\n");
   printf("                         I to insert word into dictionary\n");
   printf("                         R to remove word from dictionary\n");
   printf("                         L to load dictionary from disk\n");
   printf("                         W to write dictionary to disk\n");
   printf("                         Q to quit program\n\n");
   scanf(" %c%*[^\n]%*c", &answer);

   switch (answer) {
      case 'S': case 's':   SpeakText();
                            break;
      case 'P': case 'p':   SpeakPhonetics();
                            break;
      case 'C': case 'c':   ConvertToPhonetics();
                            break;
      case 'G': case 'g':   SetSpeechGlobals();
                            break;
      case 'A': case 'a':   DictActivate(1);
                            break;
      case 'T': case 't':   DictActivate(0);
                            break;
      case 'D': case 'd':   DisplayDict();
                            break;
      case 'I': case 'i':   InsertWord();
                            break;
      case 'R': case 'r':   DeleteWord();
                            break;
      case 'L': case 'l':   LoadDict();
                            break;
      case 'W': case 'w':   WriteDict();
                            break;
      case 'Q': case 'q':   done = true;
                            break;
      default:  printf("Please enter one of S, P, C, G, A, T, D, I, R, L, W, or Q...\n\n");
      }
   } /* while */
ShutDown();
}
```

Speak In Assembly

```
         keep  speak
         mcopy speak.macros
****************************************************************
*
*  Speak - A "plain vanilla" demonstration the Talking Tools
*        Speech Tool kit.
*
*  by Barbara Allred
*
*  Copyright 1991 by Byte Works, Inc.
*  Copyright 1987 - 1991 by First Byte, Inc.
*
****************************************************************
*
Speak    start
         using Globals

         phk                           ensure code, data in same bank
         plb
         sta   myID                    get userID passed by Loader

         jsr   Init                    initialize program
         bcs   Rtl
         jsr   Main                    if no error, execute the program
Rtl      jsr   ShutDown
         lda   #0
         rtl
         end

************************************************
*
*  Globals - Speak's global data area.
*
************************************************
*
Globals  data
;
; Constants
;
maleVoice      gequ    0               speech parameters that are enumerations
femaleVoice    gequ    1
bassTone       gequ    0
trebleTone     gequ    1

fileNotFoundErr gequ   $0046           GS/OS file-not-found error code
handle         gequ    0               "generic" handle
ptr            gequ    4               "generic" pointer
;
; Global variables
;
; ORCA/M strings:  2 length bytes before chars.
;
tmp            ds  4                   4-byte temporary
```

```
count          ds  2                    2-byte temporary

               dc  i1'255'              will be using ORCA/M's getstring macro
sayString      dc  i1'255'              English text to speak or parse
               ds  255
               dc  i1'255'              will be using ORCA/M's getstring macro
phString       dc  i1'255'              phonetic string
               ds  255
               dc  i1'1'                will be using ORCA/M's gets function
answer         dc  i1'1'                user's response to queries
               ds  1
               dc  i1'32'
word1          dc  i1'32'               strings to use with dictionary
               ds  32
               dc  i1'32'
word2          dc  i1'32'
               ds  32
;
; Other general variables
;
myID           ds  2                    program's user ID
errNum         ds  2                    error number to report to user
done           dc  i2'0'                true if user wants to exit program
;
; Speech Tool kit variables
;
voice          dc  i2'maleVoice'        default voice = male
basePitch      dc  i2'bassTone'         default tone = bass
speed          dc  i2'5'                default speed = 5
pitch          dc  i2'5'                default pitch = 5
volume         dc  i2'5'                default volume = 5
;
;  Start/stop tools data structures
;
toolTable      anop                     tool table to pass to LoadTools function
ttNumTools     dc  i2'4'                # tools to be loaded
tsArray        anop                     toolset #, min. version required
               dc  i2'50,0'               male voice
               dc  i2'51,0'               female voice
               dc  i2'52,0'               parser
               dc  i2'53,0'               GS/OS interface
;
; Error Messages
;
msg0           dc  i1'13',c'Error = $'
hexValue       ds  4
toolErr        dw  'Unable to load Speech Tools             '
openErr        dw  'Unable to open file                     '
memErr         dw  'Unable to allocate memory for buffer    '
readErr        dw  'Unable to read file                     '
infoErr        dw  'Unable to obtain information about file '
createErr      dw  'Unable to create file                   '
zeroErr        dw  'Unable to set file size to zero         '
writeErr       dw  'Unable to write file                    '
;
;  Data structures for GS/OS file handling calls.
;
pathname       dc  i1'255'              ORCA string to read dict filename
               dc  i1'255'
               dc  255c' '

options        dc  i2'6'                6 bytes total in options area
               ds  4
;                                       GS/OS open record
openRec        dc  i2'15'                 pcount
openRef        ds  2                      refNum
openPath       dc  a4'pathname'           pathname
               dc  i2'$0003'              request read/write access
               dc  i2'0'                  resource number:  open data fork
openAccess     ds  2                      access
openFileType   ds  2                      filetype
openAuxType    ds  4                      auxType
```

```
               ds  2                      storage type
               ds  8                      create date/time
               ds  8                      mod date/time
               dc  i4'options'            pointer to GS/OS result buffer
openSize       ds  4                      eof:  # bytes that can be read
               ds  4                      blocks used
               ds  4                      resource eof
               ds  4                      resource blocks
;
;                                       GS/OS read record
readRec        dc  i2'5'                  pcount
readRef        ds  2                      reference #
readBuffer     ds  4                      pointer to buffer to read into
readRequest    ds  4                      # bytes to read
transferCount  ds  4                      # bytes actually read
cache          dc  i2'0'                  don't cache
;
;                                       GS/OS close record
closeRec       dc  i2'1'                  pCount
closeRef       ds  2                      reference #
;
;                                       GS/OS write record
writeRec       dc  i2'5'                  pCount
writeRef       ds  2                      file reference #
writeData      ds  4                      pointer to data to write
writeRequest   ds  4                      # bytes to write
writeTransfer  ds  4                      # bytes actually written
               dc  i2'0'                  don't cache file
;
;                                       GS/OS getFileInfo record
GFIRec         dc  i2'12'                 pCount
GFIPath        dc  a4'pathname'           pointer to GS/OS input string
GFIInfo        ds  26                     not relevant for our purposes
               dc  a4'options'            pointer to GS/OS output buffer
               ds  24                     not interested in this stuff
;
;                                       GS/OS create record
createRec      dc  i2'7'                  pCount
createPath     dc  a4'pathname'           pointer to GS/OS input string
               dc  i2'$00C3'              destroy, rename, write, read access
createTyp      dc  i2'$00F2'              filetype = dictionary
               dc  i4'0'                  auxtype
               dc  i2'$0001'              standard file
               dc  i4'0'                  initial size of data fork is 0
               dc  i4'0'                  initial size of resource fork is 0
;
;                                       GS/OS EOF record
setEOFRec      dc  i2'3'                  pCount
setEOFRef      ds  2                      file reference #
               dc  i2'0'                  base of 0
               dc  i4'0'                  displacement of 0 to create empty file
               end

****************************************************************
*
*  ConvertToPhonetics - Convert English strings to phonetic
*        strings until user ready to stop.
*
****************************************************************
*
ConvertToPhonetics start
         using Globals

phStart  gequ  0                        char. in sayString to begin PARSE
size     gequ  2                        size of English string to PARSE

CTP0     putcr                          write carriage return
         puts  #'Enter string to parse.  Press RETURN to exit.',cr=t

         gets  sayString-1,cr=t         get string to PARSE
         lda   sayString                if length = 0, exit
         and   #$00FF
         sta   size
         bne   CTP1
         rts

CTP1     stz   phStart                  initialize where to begin parsing

CTP2     inc   phStart                  loop to PARSE current string
         pha                              integer result
         ph4   #sayString                 string to PARSE
         ph4   #phString                  string to receive phonetics
         lda   phStart                    where in English string to begin
         pha
         _Parse
         pl2   phStart                    returns last char. converted

         ph4   #phString                  write the phonetic string
         _WriteLine
         putcr                          write carriage return

         lda   phStart                    translated all the chars.?
         cmp   size
         bne   CTP2                       No - continue inner loop
         brl   CTP0                       Yes - continue outer loop
         end

****************************************************************
*
*  DeleteWord - Delete word from exceptions dictionary.
*
****************************************************************
*
DeleteWord start
         using Globals

         putcr
         puts  #'Word to delete from dictionary?',cr=t

         gets  word1-1,cr=t             get string to DELETE

         ph4   #word1                   DELETE the word from the dictionary
         _DictDelete
         rts
         end

****************************************************************
*
*  DisplayDict - Display exceptions dictionary.
*
****************************************************************
*
DisplayDict start
         using Globals

addr     gequ  4

DD0      putcr                          write carriage return
         puts  #'Before displaying dictionary, we can initialize it.',cr=t
         puts  #'Enter 0 to reset dictionary to beginning.',cr=t
         puts  #'Enter 1 to delete current dictionary.',cr=t
         puts  #'Enter 2 to NOT initialize dictionary.',cr=t

         gets  answer-1,cr=t            get initialization flag
         lda   answer+1
         and   #$00FF
         cmp   #'0'                     check if valid response
         blt   Err1
         cmp   #'3'
         bge   Err1
         sec                            if valid, convert to integer
         sbc   #$30

         cmp   #2                       skip initializing if user says to
         beq   DD1
         pha
         _DictInit
         bra   DD1

Err1     puts  #'Please enter either 0, 1, or 2.',cr=t
         brl   DD0
;
; Loop to display dictionary, from current word to end.
;
DD1      pha                           room for long result
         pha
         ph4   #word1                  English word
         ph4   #word2                  its phonetic translation
         _DictDump
         pl4   addr
         lda   word1                   check if at end of dictionary -
         and   #$00FF                    length (word1) = 0
         bne   DD2
         rts

DD2      putcr                         write carriage return
         puts  #'Next entry:  '
         ph4   #word1
         _WriteString
         puts  #'   '
         ph4   #word2
         _WriteString
         puts  #'      Continue? (Y or N)',cr=t

         gets  answer-1,cr=t
         lda   answer+1
         and   #$00FF
         cmp   #'N'
         beq   Rts
         cmp   #'n'
         beq   Rts
         brl   DD1
Rts      rts
         end

****************************************************************
*
*  Init - Start the tools, initialize global data.
*
*  Outputs:
*        carry flag - Set if error detected; clear otherwise.
*
****************************************************************
*
Init     start
         using Globals
;
; Display welcoming message.
;
         puts  #'Speak - A demonstration of the Talking Tools.',cr=t
         putcr
         puts  #'Please wait while we load the tools...',cr=t
         putcr
         putcr                         write carriage return
;
; Start the speech tools.
;
         ph4   #toolTable              load the RAM-based tools we need
         _LoadTools
         bcc   I1                      check for error from LoadTools call
         sta   errNum
         ph4   #toolErr
         jsr   ReportErr
         sec                           set error flag
         rts
```

```
I1       lda   myID
         pha
         _ParseStartUp
         _MaleStartUp
         _FemaleStartUp
         _SpeechStartUp

Rts      clc                           return OK flag
         rts
         end

****************************************************************
*
*  InsertWord - Add a new word to the exceptions dictionary.
*
****************************************************************
*
InsertWord start
         using Globals

IW0      putcr                         write carriage return
         puts  #'Press RETURN for the dictionary entries to exit.',cr=t

         puts  #'Enter English word to add to dictionary:  '
         gets  word1-1,cr=t            get English word
         lda   word1                   if length = 0, exit
         and   #$00FF
         bne   IW1
Rts      rts

IW1      puts  #'Enter phonetic representation of word to add:  '
         gets  word2-1,cr=t            get phonetic word
         lda   word2                   if length = 0, exit
         and   #$00FF
         beq   Rts

         ph4   #word1
         ph4   #word2
         _DictInsert
         brl   IW0
         end

****************************************************************
*
* LoadDict - Load dictionary from disk file.
*
****************************************************************
*
LoadDict start
         using Globals
;
; Get name of file to open.
;
         short M                       init. ORCA string to receive pathname
         lda   #255
         sta   pathname
         sta   pathname+1
         long  M

         puts  #'Enter pathname of dictionary to open:  '
         gets  pathname,cr=t           get name of file to open
         lda   pathname+1              check empty string
         and   #$00FF
         bne   LD1
         rts

LD1      sta   pathname                convert ORCA string to GS/OS input string
;
; Open the file, allocate a buffer into which it will be read, then close file.
;
         _OpenGS  openRec              open the file
         bcc   LD2                     handle error
         sta   errNum
         ph4   #openErr
         jsr   ReportErr
         rts

LD2      lda   openRef                  get ready to read and then close the file
         sta   readRef
         sta   closeRef

         ph2   #1                       clear current dict from memory
         _DictInit
;
; If the file is empty, close the file and skip loading it.
;
         lda   openSize
         ora   openSize+2
         bne   LD3
         _CloseGS closeRec
         rts
;
; If the file is not empty, allocate a read buffer, read and then close the
; file.
;
LD3      ph4   #0                       allocate memory block to read files
         ph4   openSize                 read in the whole thing
         ph2   myID
         ph2   #$C010                   locked, can't move, don't purge, don't
!                                         cross bank bound., don't page align
         ph4   #0                       no absolute address specified
         _NewHandle
         bcc   LD3A                     handle error:
         plx                              throw away zero handle
         plx
         sta   errNum
         ph4   #memErr
         jsr   ReportErr                  report error
         _CloseGS  closeRec               close file
         rts                              return

LD3A     pl4   handle                   get ready to dereference the handle

         lda   [handle]                 dereference memory handle
         sta   readBuffer
         sta   ptr
         ldy   #2
         lda   [handle],Y
         sta   readBuffer+2
         sta   ptr+2

         lda   openSize                 get # bytes to read
         sta   readRequest
         lda   openSize+2
         sta   readRequest+2

         _ReadGS  readRec               make the Read call
         php                            save error flag from read
         pha                            save error #
         _CloseGS closeRec
         pla
         plp
         bcc   LD4                      handle error
         sta   errNum
         ph4   #readErr
         jsr   ReportErr
         bra   LD5
;
; Create new dictionary from entries in buffer.
;
LD4      clc                            tmp := address of last byte in buffer
         lda   ptr
         adc   transferCount
         sta   tmp
         lda   ptr+2
         adc   transferCount+2
         sta   tmp+2

LD4A     lda   ptr+2                    while ptr < tmp do begin
         cmp   tmp+2
         blt   LD4C
         beq   LD4B
         bra   LD5
LD4B     lda   ptr
         cmp   tmp
         blt   LD4C
         bra   LD5

LD4C     lda   ptr+2                      push pointer to English word
         pha
         lda   ptr
         pha
         lda   [ptr]                      calc. pointer to phonetic word:
         and   #$00FF                     length (English word) + 1 + textPtr
         inc   A
         clc
         adc   ptr
         bcc   LD4D
         inc   ptr+2
LD4D     sta   ptr
         ldx   ptr+2
         phx                              push pointer to phonetic word
         pha
         _DictInsert                      insert new entry into dictionary

         lda   [ptr]                      calc. pointer to next entry
         and   #$00FF
         inc   A
         clc
         adc   ptr
         bcc   LD4E
         inc   ptr+2
LD4E     sta   ptr
         bra   LD4A                     end {while}
;
; Clean up and return.
;
LD5      lda   handle+2
         pha
         lda   handle
         pha
         _DisposeHandle
         ph2   #0                       reset dict to top
         _DictInit
         rts
         end

****************************************************************
*
*  Main - Speak's main function.  Presents main menu and acts
*        on user's choice.
*
****************************************************************
*
Main     start
         using Globals
;
; Display main menu.
;
Top      puts  #'Enter desired function:  S to speak English string ',cr=t
         puts  #'                         P to speak phonetic string',cr=t
         puts  #'                         C to convert to phonetics ',cr=t
         puts  #'                         G to set global speech parameters',cr=t
         puts  #'                         A to activate dictionary  ',cr=t
         puts  #'                         T to deactivate dictionary',cr=t
         puts  #'                         D to display dictionary   ',cr=t
         puts  #'                         I to insert word into dictionary ',cr=t
         puts  #'                         R to remove word from dictionary ',cr=t
         puts  #'                         L to load dictionary from disk',cr=t
         puts  #'                         W to write dictionary to disk',cr=t
         puts  #'                         Q to quit program         ',cr=t
         putcr

         gets  answer-1,cr=t            get user's choice
         lda   answer+1                 dispatch appropriate routine:
         and   #$00FF                   case (answer):

         cmp   #'S'                       'S','s':  SpeakText;
         bne   M2
M1       jsr   SpeakText
         brl   Top
M2       cmp   #'s'
         beq   M1

         cmp   'P'                        'P','p': SpeakPhonetics
         bne   M4
M3       jsr   SpeakPhonetics
         brl   Top
M4       cmp   #'p'
         beq   M3

         cmp   #'C'                       'C','c': ConvertToPhonetics
         bne   M6
M5       jsr   ConvertToPhonetics
         brl   Top
M6       cmp   #'c'
         beq   M5

         cmp   #'G'                       'G','g': SetSpeechGlobals
         bne   M8
M7       jsr   SetSpeechGlobals
         brl   Top
M8       cmp   #'g'
         beq   M7

         cmp   #'A'                       'A','a': Activate dict
         bne   M10
M9       ph2   #1
         _DictActivate
         brl   Top
M10      cmp   #'a'
         beq   M9

         cmp   #'T'                       'T','t': Deactivate dict
         bne   M12
M11      ph2   #0
         _DictActivate
         brl   Top
M12      cmp   #'t'
         beq   M11

         cmp   #'D'                       'D','d': DisplayDict
         bne   M14
M13      jsr   DisplayDict
         brl   Top
M14      cmp   #'d'
         beq   M13

         cmp   #'I'                       'I','i': InsertWord
         bne   M16
M15      jsr   InsertWord
         brl   Top
M16      cmp   #'i'
         beq   M15

         cmp   #'R'                       'R','r': DeleteWord
         bne   M18
M17      jsr   DeleteWord
```

```
         brl   Top
M18      cmp   #'r'
         beq   M17

         cmp   #'L'                       'L','l': LoadDict
         bne   M20
M19      jsr   LoadDict
         brl   Top
M20      cmp   #'l'
         beq   M19

         cmp   #'W'                       'W','w': WriteDict
         bne   M22
M21      jsr   WriteDict
         brl   Top
M22      cmp   #'w'
         beq   M21

         cmp   #'Q'                       'Q','q': exit Main
         beq   Rts
         cmp   #'q'
         bne   M23
Rts      rts

M23      putcr                          write carriage return
         puts  #'Please enter one of S, P, C, G, A, T, D, I, R, or Q',cr=t
         putcr                          write carriage return
         brl   Top
         end

****************************************************************
*
*  ReportErr - Report tool errors detected by program.
*
****************************************************************
*
ReportErr start
         using Globals

rtsAddr  gequ  8

         pl2   rtsAddr                  save return address

         _WriteLine
         pha                            room for long result
         pha
         ph2   errNum
         _HexIt
         pl4   hexValue
         ph4   #msg0
         _WriteLine

         ph2   rtsAddr                  restore return address
         rts
         end

****************************************************************
*
*  SetSpeechGlobals - Set global speech parameters.
*
****************************************************************
*
SetSpeechGlobals start
         using Globals

         putcr                          write carriage return
         puts  #'Follow the prompts to change the speech parameters.',cr=t
;
; Let user change current voice setting.
;
SSG1     puts  #'Voice = '              display current voice setting
         lda   voice
         bne   SSG2
         puts  #'male  '
         bra   SSG3
SSG2     puts  #'female  '
SSG3     puts  #'Enter 0 for male voice, 1 for female voice.',cr=t

         gets  answer-1,cr=t            get user's response
         lda   answer+1
         and   #$00FF
         cmp   #'0'                     check if it's valid
         blt   DoErr1
         cmp   #'2'
         bge   DoErr1
         sec                            if valid, convert to integer
         sbc   #$30
         sta   voice
         bra   SSG4
DoErr1   jsr   Err1
         brl   SSG1

Err1     putcr                          write carriage return
         puts  #'Value must be either 0 or 1.',cr=t
         rts
;
; Let user change current tone setting.
;
SSG4     puts  #'Tone = '               display current tone setting
         lda   basePitch
         bne   SSG5
         puts  #'bass    '
         bra   SSG6
SSG5     puts  #'treble  '
SSG6     puts  #'Enter 0 for bass tone, 1 for treble tone.',cr=t

         gets  answer-1,cr=t            get user's response
         lda   answer+1
         and   #$00FF
         cmp   #'0'                     check if it's valid
         blt   DoErr2
         cmp   #'2'
         bge   DoErr2
         sec                            if valid, convert to integer
         sbc   #$30
         sta   basePitch
         bra   SSG7

DoErr2   jsr   Err1
         brl   SSG4
;
; Let user change current speed setting.
;
SSG7     puts  #'Speed = '              display current speed setting
         put2  speed,cr=t

         gets  answer-1,cr=t            get user's response
         lda   answer+1
         and   #$00FF
         cmp   #'0'                     check if it's valid
         blt   DoErr3
         cmp   #':'
         bge   DoErr3
         sec                            if valid, convert to integer
         sbc   #$30
         sta   speed
         bra   SSG8

DoErr3   jsr   Err2
         bra   SSG7

Err2     putcr                          write carriage return
         puts  #'Value must be between 0 and 9.',cr=t
         rts
```

```
;
; Let user change current volume setting.
;
SSG8     puts  #'Volume = '            display current volume setting
         put2  volume,cr=t

         gets  answer-1,cr=t           get user's response
         lda   answer+1
         and   #$00FF
         cmp   #'0'                    check if it's valid
         blt   DoErr4
         cmp   #':'
         bge   DoErr4
         sec                           if valid, convert to integer
         sbc   #$30
         sta   volume
         bra   SSG9

DoErr4   jsr   Err2
         bra   SSG8
;
; Let user change current pitch setting.
;
SSG9     puts  #'Pitch = '             display current pitch setting
         put2  pitch,cr=t

         gets  answer-1,cr=t           get user's response
         lda   answer+1
         and   #$00FF
         cmp   #'0'                    check if it's valid
         blt   DoErr5
         cmp   #':'
         bge   DoErr5
         sec                           if valid, convert to integer
         sbc   #$30
         sta   pitch

         lda   voice
         pha
         lda   basePitch
         pha
         lda   pitch
         pha
         lda   speed
         pha
         lda   volume
         pha
         _SetSayGlobals
         rts

DoErr5   jsr   Err2
         bra   SSG9
         end

****************************************************************
*
*  ShutDown - Shut down tools, do any necessary clean-up before
*        exiting program.
*
****************************************************************
*
ShutDown start
         using Globals

         _MaleShutDown
         _FemaleShutDown
         _ParseShutDown
         _SpeechShutDown
         rts
         end
```

```
****************************************************************
*
*  SpeakPhonetics - Speak phonetic strings until user is ready
*        to stop.
*
****************************************************************
*
SpeakPhonetics start
         using Globals

SP0      putcr                          write carriage return
         puts  #'Enter phonetic string to speak.  RETURN to exit.',cr=t

         gets  phString-1,cr=t          get string to SPEAK
         lda   phString                 if length = 0, exit
         and   #$00FF
         bne   SP1
         rts

SP1      lda   volume                   else SPEAK the string just entered
         pha
         lda   speed
         pha
         lda   pitch
         pha
         ph4   #phString
         lda   voice                    call MALE_SPEAK or FEMALE_SPEAK,
         bne   SP2                        whichever is appropriate
         _MaleSpeak
         brl   SP0
SP2      _FemaleSpeak
         brl   SP0
         end

****************************************************************
*
*  SpeakText - Speak English strings until user is ready to stop.
*
****************************************************************
*
SpeakText start
         using Globals

ST0      putcr                          write carriage return
         puts  #'Enter string to speak.  Press RETURN to exit.',cr=t

         gets  sayString-1,cr=t         get string to SAY
         lda   sayString                if length = 0, exit
         and   #$00FF
         bne   ST1
         rts

ST1      ph4   #sayString               else SAY the string just entered
         _Say
         bra   ST0
         end

****************************************************************
*
*  WriteDict - Write dictionary to a disk file.
*
****************************************************************
*
WriteDict start
         using Globals
;
; Get name of file to open.
;
         short M                        init. ORCA string to receive pathname
         lda   #255
         sta   pathname
         sta   pathname+1
         long  M

         puts  #'Enter pathname of dictionary to write:  '
         gets  pathname,cr=t            get name of file to open
         lda   pathname+1               check empty string
         and   #$00FF
         bne   WD1
         rts

WD1      sta   pathname                 convert ORCA string to GS/OS input string
;
; Check if the file exists.  If not, create the file.
;
         _GetFileInfoGS GFIRec          make GetFileInfo call to see if exists
         bcc   WD3
         cmp   #fileNotFoundErr         if error not file-not-found
         beq   WD2
         sta   errNum                     report error
         ph4   #infoErr
         jsr   ReportErr
         rts                              and exit

WD2      _CreateGS createRec            else create the file
         bcc   WD3
         sta   errNum
         ph4   #createErr
         jsr   ReportErr
         rts
;
; Open the file, set its size to zero, allocate a write buffer.
;
WD3      _OpenGS  openRec               open the file
         bcc   WD3A                     handle error
         sta   errNum
         ph4   #openErr
         jsr   ReportErr
         rts

WD3A     lda   openRef                  init. GS/OS file reference #s
         sta   setEOFRef
         sta   writeRef
         sta   closeRef

         _SetEOFGS setEOFRec            set file's size to 0
         bcc   WD3B                     handle error
         sta   errNum
         ph4   #zeroErr
         jsr   ReportErr
         rts
;
; Allocate the write buffer.
;
WD3B     ph4   #0                       allocate memory block to read files
         ph4   #1024                    1K buffer
         ph2   myID
         ph2   #$C010                   locked, can't move, don't purge, don't
;                                         cross bank bound., don't page align
         ph4   #0                       no absolute address specified
         _NewHandle
         bcc   WD3C                     handle error:
         plx                              throw away zero handle
         plx
         sta   errNum
         ph4   #memErr
         jsr   ReportErr                  report error
         _CloseGS  closeRec               close file
         rts                              return

WD3C     pl4   handle                   get ready to dereference the handle

         lda   [handle]                 dereference memory handle
         sta   writeData
         ldy   #2
         lda   [handle],Y
         sta   writeData+2

         ph2   #0                       reset dictionary to top
         _DictInit
;
; Main loop:  while not at end of dictionary, fill the buffer with dictionary
;             entries, then write the buffer to disk.
;
WD4      pha                            room for long result
         pha
         ph4   #word1
         ph4   #word2
         _DictDump                      get 1st dictionary entry
         pla
         pla

WD4A     stz   writeRequest             init. FillBuffer variables
         stz   writeRequest+2
         lda   [handle]
         sta   ptr
         ldy   #2
         lda   [handle],Y
         sta   ptr+2
         ldy   #0

         jsr   FillBuffer
         php                            save flag returned from FillBuffer rtn

         _WriteGS  writeRec             write buffer to disk
         bcc   WD4B
         sta   errNum
         ph4   #writeErr
         jsr   ReportErr
         plp
         bra   WD5

WD4B     plp                            retrieve flag returned from FillBuffer
         bcc   WD4A
;
; Final clean-up:  Close the file, deallocate buffer.
;
WD5      _CloseGS  closeRec             close the file
         lda   handle+2
         pha
         lda   handle
         pha
         _DisposeHandle
         ph2   #0                       reset dict. to top
         _DictInit
         rts
;
; FillBuffer:  fill the write buffer with dictionary entries.
;
FillBuffer anop
         lda   word1                    check if we've gotten all entries
         and   #$00FF
         bne   FB1
         sec                            yes - set flag that we're done
         rts

FB1      lda   writeRequest             no - check if new entries will fit in
         sta   tmp                        buffer
         lda   writeRequest+2
         sta   tmp+2
         lda   word1                    add size of English word
         and   #$00FF
         clc
         adc   tmp
         bcc   FB1A
         inc   tmp+2
```

```
FB1A     sta   tmp
         lda   word2                    add size of phonetic word
         and   #$00FF
         clc
         adc   tmp
         bcc   FB1B
         inc   tmp+2
FB1B     sta   tmp
         lda   #2                       add length bytes to total
         clc
         adc   tmp
         bcc   FB1C
         inc   tmp+2
FB1C     sta   tmp
         cmp   #1024                    ensure that new total size is
         blt   FB2                        less than or equal to buffer size
         clc                            set flag that we're not done, then exit
         rts
;
; We're not at the end of the dictionary, and the current entry will fit in
; our buffer, so we move the entries to the buffer, then get the next entry.
;
FB2      lda   tmp                      update current amt. chars. in buffer
         sta   writeRequest
         lda   tmp+2
         sta   writeRequest+2

         lda   word1                    move English word to buffer
         and   #$00FF
         sta   count
         ldx   #0
         short M
FB2A     lda   word1,X
         sta   [ptr],Y
         inx
         iny
         dec   count
         bpl   FB2A
         long  M

         lda   word2                    move phonetic word to buffer
         and   #$00FF
         sta   count
         ldx   #0
         short M
FB2B     lda   word2,X
         sta   [ptr],Y
         inx
         iny
         dec   count
         bpl   FB2B
         long  M

         phy                            save index into buffer
         pha                            room for long result
         pha
         ph4   #word1
         ph4   #word2
         _DictDump                      get next dictionary entry
         pla
         pla
         ply
         brl   FillBuffer
         end
```

Appendix B - Licensing the Speech Tools

The speech tools used in this product are copyrighted by First Byte, Inc., and have been licensed to the Byte Works for use in our Talking Tools package. While you can create as many programs as you like that make use of the speech tools, and distribute your programs in any way you choose, you cannot distribute the speech tool files themselves without obtaining a separate license from First Byte, Inc. Specifically, the files that you cannot distribute without a license are Tool050, Tool051, and Tool052. To obtain current information about licensing the speech tools, contact First Byte, Inc. at this address:

First Byte, Inc.
3333 E. Spring Street #302
Long Beach, CA 90806
(213) 565-7006

The tool Tool053 was developed by the Byte Works, Inc. to correct minor incompatibilities between the speech tools developed by First Byte, Inc., and the latest version of the Apple IIGS operating system. You may distribute this file with your programs, so long as the following copyright message appears somewhere in your documentation or in the About box of the program itself:

Tool053 Copyright 1991, Byte Works Inc. Used with permission.

Finally, you should feel free to make use of the ideas represented in any of the sample programs in this book or on the accompanying disks in your own programs. If you use any of the source verbatim in your program, however, you must include the following statement in the source code and somewhere in the documentation or in the About box:

Some subroutines Copyright 1991, Byte Works, Inc. Used with permission.