C H A P T E R 1

Speech Recognition Manager

Contents

About the Speech Recognition Manager 1-5
Speech Objects 1-8
Object References 1-9
Object Properties 1-10
Language Models 1-11
Feedback Services 1-13
Using the Speech Recognition Manager 1-17
Checking for Speech Recognition Capabilities 1-17
Opening Recognition Systems and Recognizers 1-18
Building a Language Model 1-21
Setting the Language Model 1-25
Setting the Rejection Word 1-25
Starting and Stopping Speech Recognition 1-27
Handling Recognition Notifications 1-27
Using Apple Events 1-28
Using Callback Routines 1-30
Interpreting Recognition Results 1-33
Saving and Loading Language Objects 1-37
Speech Recognition Manager Reference 1-38
Constants 1-39
Gestalt Selectors and Response Values 1-39
Recognition System IDs 1-40
Recognition System Properties 1-40
Apple Event Selectors 1-42
Recognizer Properties 1-43
Search Status Flags 1-47
Notification Flags 1-48

Listen Key Modes 1-49
Recognition Result Properties 1-49
Language Object Properties 1-51
Language Object Types 1-53
Data Structures 1-53
Speech Recognition CallBack Structure 1-53
Callback Routine Parameter Structure 1-55
Speech Recognition Manager Routines 1-55
Opening and Closing Recognition Systems 1-56
Creating and Manipulating Recognizers 1-57
Managing Speech Objects 1-62
Creating Language Objects 1-66
Manipulating Language Objects 1-70
Traversing Speech Objects 1-75
Reading and Writing Language Objects 1-78
Using the System Feedback Window 1-82
Application-Defined Routines 1-89
Speech Recognition Callback Routines 1-89
Summary of the Speech Recognition Manager 1-91
C Summary 1-91
Constants 1-91
Data Types 1-95
Speech Recognition Manager Routines 1-96
Application-Defined Routines 1-100
Pascal Summary 1-100
Constants 1-100
Data Types 1-103
Speech Recognition Manager Routines 1-104
Application-Defined Routines 1-107
Result Codes 1-108

Apple Computer, Inc.
© 1996 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc. Printed in the United States of America.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple Macintosh computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for printing or clerical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, and Macintosh, are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Adobe Illustrator and PostScript are trademarks of Adobe Systems Incorporated, which may be registered in certain jurisdictions.

FrameMaker is a registered trademark of Frame Technology Corporation.

Helvetica and Palatino are registered trademarks of Linotype Company.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

This chapter describes the Speech Recognition Manager, the part of the Macintosh system software that your application can use to respond to speech. You use the Speech Recognition Manager to define the words and phrases you want to listen for and to control other aspects of recognizing speech and reacting to successful recognitions.

To use this chapter, you should be familiar with the Apple Event Manager, as described in *Inside Macintosh: Interapplication Communication*. You need to know how to receive and process Apple events if you rely on the default behavior for being notified about the words and phrases recognized by the Speech Recognition Manager.

This chapter begins by describing the basic capabilities of the Speech Recognition Manager. Then it shows how to use some of those capabilities to recognize speech and to react to that speech. The section "Speech Recognition Manager Reference," beginning on page 1-38, provides a complete reference to the constants, data structures, and functions provided by the Speech Recognition Manager.

Note
This chapter describes the application programming interfaces supported by version 1.5 and later of the Speech Recognition Manager. ◆

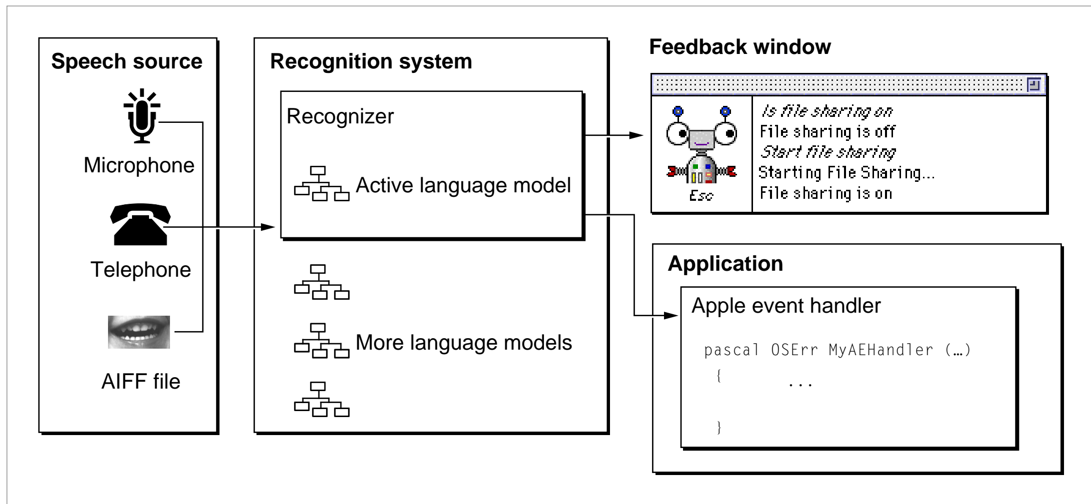
About the Speech Recognition Manager

The **Speech Recognition Manager** provides speech recognition services for Macintosh computers. **Speech recognition** is the process of listening to and recognizing spoken utterances. The main functional parts of the Speech Recognition Manager and its connection to your application are illustrated in Figure 1-1 on page 1-6.

The Speech Recognition Manager is part of the PlainTalk software package. **PlainTalk** is a collection of operating system software that enables Macintosh computers to speak written text and to respond to spoken commands. The PlainTalk package includes the Speech Recognition Manager and the Speech Synthesis Manager.

Note
The Speech Synthesis Manager was formerly called the Speech Manager. Its name has been changed to distinguish it from the Speech Recognition Manager and to describe its operation more clearly. ◆

Figure 1-1 The parts of the Speech Recognition Manager



To use the Speech Recognition Manager, you must first open a **recognition system** that defines certain global characteristics of the speech recognition process. Typically, you create a recognition system when your application starts up and later close it when your application exits. A recognition system determines, for example, whether the feedback window is displayed. (See “Feedback Services” on page 1-13 for information about the feedback window.)

You define the words and phrases you want to listen for by creating a **language model.** For example, if you wanted to allow the user to select an animal from among ten whose pictures are displayed in a window, you could build a language model containing ten phrases, the names of those ten animals. Your application can create several language models and set one of them as the **active language model.** This is useful if the words or phrases for which you are

listening can change according to context (for instance, according to what's displayed in a window). Similarly, your application can create language models that contain other language models, and you can change parts of language models dynamically according to context.

Your language model is associated with a **recognizer,** which performs the work of recognizing utterances and reporting its results to your application. A recognizer communicates with your application by sending it a **recognition notification,** which specifies the event that prompted the notification and other information about that event. For example, when a recognizer recognizes an utterance described in your active language model, the recognizer informs your application that it has recognized something by sending it a recognition notification that contains a **recognition result** describing the recognized utterance.

You can receive recognition notifications in one of two ways. By default, the Speech Recognition Manager sends recognition notifications to your application's Apple event handler. If your application already handles Apple events, you simply need to add code to handle the events in the Apple event speech class. If you're not writing an application and your code cannot receive Apple events, you can instruct the Speech Recognition Manager to send recognition notifications to an application-defined **speech recognition callback routine**.

Your application can inspect a recognition result to determine what was said and how to react to that utterance. In the example described earlier, where the user can utter an animal's name, you might respond by playing a recorded sound from that animal or by displaying its name. A recognition result contains the text of the recognized utterance together with other information that makes it easy for your application to interpret the result. (See "Recognition Result Properties" on page 1-49 for a complete description of the information contained in a recognition result.)

You can use the Speech Recognition Manager to recognize continuous speech from any human speaker, subject to these limitations:

- The Speech Recognition Manager is supported only on PowerPC-based Macintosh computers.
- The Speech Recognition Manager is designed to recognize speech from adult speakers of North American English. It doesn't work well for children, and it isn't yet localized for other regions.

- The Speech Recognition Manager works best with language models that are relatively small. You can obtain maximum recognition accuracy by limiting the active language model to a few distinct words or phrases at any one time. The current system can recognize a few dozen phrases fairly well.

Speech Objects

The Speech Recognition Manager is *object oriented* in the sense that many of its capabilities are accessed by creating and manipulating speech objects. A **speech object** is an instance of a **speech class,** which defines a set of properties for objects in the class. The behavior of a speech object is determined by the set of properties associated with the object's class. Here are the basic type definitions for speech objects:

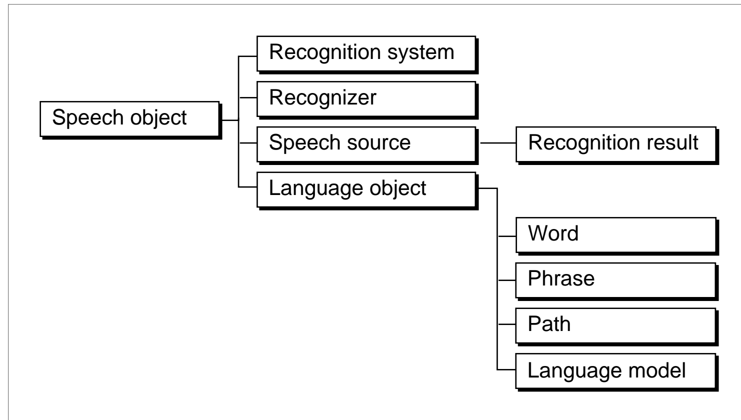
```
typedef struct OpaqueSRSpeechObject     *SRSpeechObject;

typedef SRSpeechObject                  SRRecognitionSystem;
typedef SRSpeechObject                  SRRecognizer;
typedef SRSpeechObject                  SRSpeechSource;
typedef SRSpeechObject                  SRLanguageObject;

typedef SRSpeechSource                  SRRecognitionResult;
```

As these definitions make clear, all speech classes are arranged in the **speech class hierarchy,** a hierarchical structure that provides for inheritance and overriding of class data and methods. Figure 1-2 shows the speech class hierarchy.

Figure 1-2 The speech class hierarchy



As you can see, words, phrases, paths, and language models are all subclasses of the language object class. In addition, language objects, recognition systems, recognizers, and recognition results are all subclasses of the speech object class.

This hierarchy is useful for determining which Speech Recognition Manager routines operate on which types of objects. A function that takes a parameter of type `SRSpeechObject` can operate on an object located anywhere in this hierarchy, because all speech classes are derived from the `SRSpeechObject` class. (So, for example, you can use the `SRGetProperty` function to get a property of a recognizer, a recognition system, a word, a phrase, and so on.) A function that takes a parameter of type `SRLanguageObject` can operate on any language object.

Object References

You access a speech object by using an **object reference** (or, more briefly, a **reference**). You obtain a reference whenever you create an object (for instance, by calling the `SRNewWord` function) or whenever you retrieve an object (for instance, by calling the `SRGetIndexedItem` function). A reference is essentially a tag for some private information maintained internally by the Speech Recognition Manager. The Speech Recognition Manager allocates space (usually in the system heap) for that information when you first create an object and returns a reference for that object to your application. You can dispose of that memory only indirectly, by calling `SRReleaseObject` to release an object.

It's possible to have more than one reference to a speech object. (For example, if you call the `SRGetIndexedItem` routine twice on the same speech object with the same index, you'll be given two different references to the same speech object.) For each speech object, the Speech Recognition Manager maintains a **reference count** (that is, the number of references that exist to that object).

Certain operations on a speech object increase its reference count, and other operations decrease it. When you first create a speech object (by calling a function beginning `SRNew`), its reference count is set to 1. When you retrieve an object (by calling a function beginning `SRGet`), the object's reference count is incremented by 1. When you release an object, the object's reference count is decremented by 1. If the reference count of an object becomes 0, the memory occupied by that object is disposed of.

IMPORTANT

You must balance every function call that creates an object reference with a call to the `SRReleaseObject` function when you are finished using that reference. Failure to do so will result in a memory leak. Every call to a function of the form `SRNew`*Object* or `SRGet`*Object* that successfully returns an object reference must be balanced by a call to `SRReleaseObject`. ▲

Object Properties

Each type of speech object has one or more **properties** associated with it that control some of the object's behavior. For instance, a word has a spelling property that indicates how the word is spelled. A property is just an item of data associated with an object; this item of data has both a **property type** and a **property value.**

The property types associated with an object depend on the location of that object in the speech class hierarchy. Every language object (that is, every instance of any subclass of the `SRLanguageObject` class) shares a number of properties, including properties that determine the spelling of the object, an application-defined reference constant for the object, and whether the object can meaningfully be repeated by the user. By contrast, a recognition system is not a language object and hence has no spelling property. Instead, recognition systems have other properties (for instance, the feedback and listening modes property mentioned later in the next section).

You can get or set a specific property of an object by calling the `SRGetProperty` and `SRSetProperty` functions. You indicate the object for which you want to get or set a property by passing an object reference. You indicate the property of that object you want to get or set by passing a **property selector.** For instance, the following lines of code set a word's reference constant property to a specified value.

```
/* suppose that the word myWord has already been created */
unsigned long       myVal;
OSErr               myErr;

myVal = 3;
myErr = SRSetProperty(myWord, kSRRefCon, &myVal, sizeof(myVal));
```

The constant `kSRRefCon` is a property selector for the reference constant property of a language object. As shown later (in "Building a Language Model" on page 1-21 and "Interpreting Recognition Results" on page 1-33), you'll probably use a language object's reference constant property to help you interpret the data passed to you in a recognition result. In general, all properties of an object have reasonable default values.

See "Recognition System Properties" on page 1-40, "Recognizer Properties" on page 1-43, "Recognition Result Properties" on page 1-49, and "Language Object Properties" on page 1-51 for a description of the properties defined by the Speech Recognition Manager.

Language Models

You specify the words and phrases for which you want the Speech Recognition Manager to listen by defining a language model, which is a list of zero or more words, phrases, or paths. For example, suppose that you want the user to be able to utter commands like "call Arlo" and "schedule a lunch with Brent next Tuesday" (perhaps with other names and days as well). A standard method of specifying language models like this is to display the model in **Backus-Naur Form (BNF).** Listing 1-1 is a BNF description of a relatively simple language model.

Listing 1-1 A BNF description of the language model <TopLM>

<*TopLM*>= <*call*> <*person*> | schedule meeting with <*person*> |view today's schedule;
<*call*>= call | phone| dial;
<*person*>= Arlo | Brent | Matt | my wife;

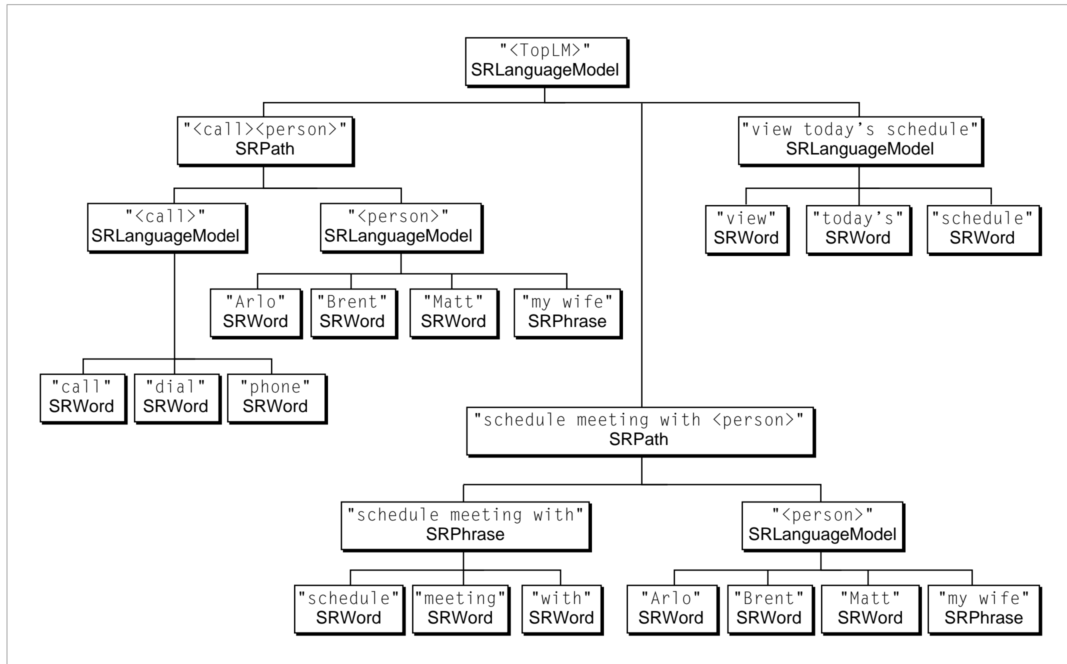
A language model is built using four kinds of objects, collectively called **language objects:** words, phrases, paths, and language models. A **word** represents a single speakable word (for example, "dial" in the BNF description). A **phrase** is a sequence of zero or more words (for example, "view today's schedule" in the BNF description). A **path** is a sequence of zero or more words, phrases, or language models (for example, "<*call*> <*person*>" in the BNF description). Finally, a **language model** is a list of zero or more words, phrases, or paths (for example, "call | phone | dial" in the BNF description).

By convention, the name of a language model is enclosed in the characters "<" and ">". As you can see, the terms on the left sides of the "=" characters in the BNF description are names of language models. Some of those language models are used in paths that define other language models. The right side of each line in a BNF description is a set of one or more paths. When more than one path occurs, the character "|" (which you can read "or") separates the paths.

Figure 1-3 provides another way, using a hierarchical decomposition, of illustrating the structure of the language model <*TopLM*> described in Listing 1-1. As you can see, the language model <*TopLM*> can be decomposed into three component parts: two paths (of type `SRPath`) and a phrase (of type `SRPhrase`). The path "<*call*> <*person*>" is decomposed into two language models (each of type `SRLanguageModel`), and the language model <*call*> is decomposed into three words (of type `SRWord`).

Note
In Figure 1-3, some language objects are depicted more than once in the decomposition of the language model <*TopLM*>. For example, the language model <*person*> appears twice in the hierarchy. When stored in memory, a language object appears only once, and multiple occurrences of the language object are handled using an object reference, See "Object References" on page 1-9 for complete details. ◆

Figure 1-3 Structure of the language model <TopLM>



A language model that does not occur in the definition of any other language model is a **top-level language model.** A language model that does occur in the definition of some other language model is an **embedded language model.** In the BNF description given earlier, the language model <*TopLM*> is a top-level language model and the language model <*person*> is an embedded language model.

Note
See "Building a Language Model," beginning on page 1-21 for sample code that builds a language model. ◆

Feedback Services

The Speech Recognition Manager includes a set of **feedback services** that provide important cues and responses to the user. As you will see, it is very

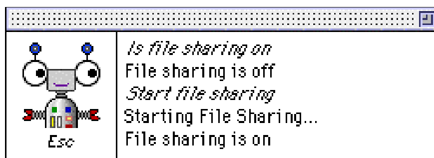
easy to obtain the default feedback behavior by setting the `kSRFeedbackAndListeningModes` property of your recognition system (see Listing 1-3 on page 1-19). The services consist of a floating feedback window that is normally visible when a recognizer is active and a set of functions that you can use to modify the operation of the feedback window. Because users typically have difficulty using speech recognition systems without some kind of audio or visual feedback, you should use these feedback services unless you have reasons to provide your own method of eliciting utterances from the user and providing responses to the user.

IMPORTANT

In general, you should use the Speech Recognition Manger's feedback services unless your application cannot. For example, under System 7.5, if your application doesn't call `WaitNextEvent` or is not high-level event aware, then it should not use the feedback services. By using these services, you are guaranteed to conform to the standard speech recognition interface. In addition, you will automatically benefit from future enhancements to the feedback services. ▲

A **feedback window** (shown in Figure 1-4) appears on the screen whenever a recognizer is active (unless the recognizer is configured to display no feedback window). The feedback window consists of two panes: a status pane on the left and a transcript pane on the right.

Figure 1-4 A feedback window



Note

The appearance of the feedback window may change in future versions of the Speech Recognition Manager. ◆

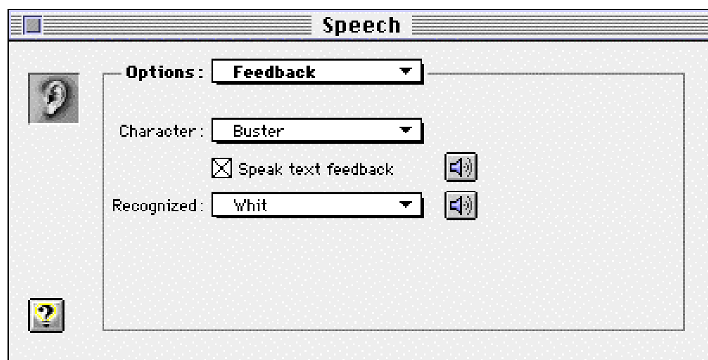
The **status pane** contains a **feedback character** (usually a head) whose expressions help indicate which state the recognizer is in. For instance, if the

recognizer is awaiting an utterance, the eyes of the feedback character shown in Figure 1-4 will look straight at the user. The status pane also contains a **listening mode indicator,** a word or phrase that indicates which keyword must be uttered or which key must be held down in order for a recognizer to start listening. In Figure 1-4, the listening mode indicator is the word “Esc”, which indicates that the user must hold down the Escape key while uttering a command. (This is the **push-to-talk** listening mode.)

The **transcript pane** contains a readable transcript of the few most recent recognized utterances and feedback. There are two kinds of text in the transcript pane: recognized text and output text. **Recognized text** is text drawn into the transcript pane that represents a recognized utterance from the user. **Output text** is text drawn into the transcript pane that represents a response to a recognized utterance. For example, in Figure 1-4, the sentence “Is file sharing on” is recognized text, and the response “File sharing is off” is output text. Any application can request that a string be spoken or displayed in the feedback window.

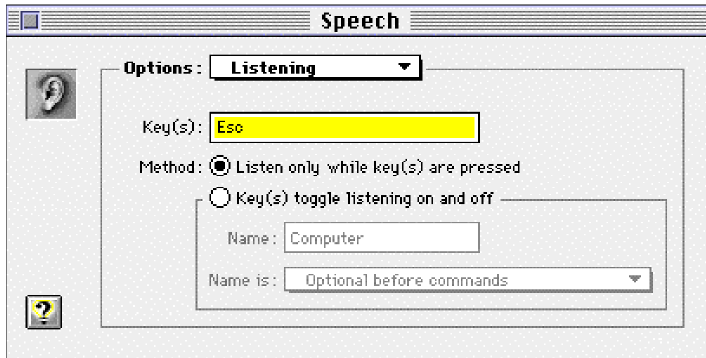
The user can control several aspects of the feedback window using the Feedback option in the Speech control panel (shown in Figure 1-5). As you can see, a user can select the feedback character and the sound that is played whenever an utterance is recognized. The user can also determine whether any output text is to be read aloud to the user.

Figure 1-5 The Feedback option of the Speech control panel



The user can control the listening mode of a recognizer using the Listening option in the Speech control panel (shown in Figure 1-6).

Figure 1-6 The Listening option of the Speech control panel



Whether the feedback window appears when a recognizer is active and whether the recognizer uses the listening modes selected by the user in the Speech control panel depend on the **feedback and listening modes** of the recognition system associated with that recognizer. When you create a recognition system, you should explicitly set the desired feedback and listening modes. In general, you should set the feedback and listening modes to `kSRHasFeedbackHasListenModes`, so that the user has the same speech input and feedback experience provided by other applications using speech recognition. See "Recognition System Properties" on page 1-40 for a complete description of the available feedback and listening modes.

Note
To use the feedback services, your application must able to handle Apple events (as indicated by its `'SIZE'` resource). Other code using the feedback services must be running in the layer of an application that is able to handle Apple events. ◆

You can use Speech Recognition Manager functions to control certain aspects of the feedback window. For instance, you can draw output text in the feedback window, and you can have the feedback character speak text (complete with lip

synchronization). See "Using the System Feedback Window" on page 1-82 for a description of these functions.

Using the Speech Recognition Manager

This section illustrates basic ways of using the Speech Recognition Manager. In particular, it provides source code examples that show how you can

- determine whether the Speech Recognition Manager is available
- open a recognition system and a recognizer
- build a language model to define the words and phrases your application wants to listen for
- set that model as the active language model
- start and stop listening
- handle recognition results using an Apple event handler or a speech recognition callback routine
- interpret the data passed to your application in a recognition result
- save a language model (or other language object) into a resource or data file, and read that object from that file

Note
The code examples shown in this section provide only very rudimentary error handling. ◆

Checking for Speech Recognition Capabilities

Before calling any speech recognition routines, you need to verify that the Speech Recognition Manager is available in the current operating environment and that it has the capabilities you need. You can verify that the Speech Recognition Manager is available by calling the `Gestalt` function with the `gestaltSpeechRecognitionVersion` selector. `Gestalt` returns a long word whose value indicates the version of the Speech Recognition Manager.

IMPORTANT

You should ensure that the value returned in the `response` parameter is greater than or equal to 0x0150 before using the programming interfaces described in this chapter. ▲

Listing 1-2 illustrates how to determine whether the Speech Recognition Manager is available.

Listing 1-2 Checking for the availability of the Speech Recognition Manager

```
Boolean MyHasSpeechRecognitionMgr (void)
{
    OSErr           myErr;
    long            mySRVersion;
    Boolean         myHasSRMgr = FALSE;

    myErr = Gestalt (gestaltSpeechRecognitionVersion, &mySRVersion);
    if (!myErr)
        if (mySRVersion >= 0x0150)
            myHasSRMgr = TRUE;

    return myHasSRMgr;
}
```

Note

For more information on the `Gestalt` function, see *Inside Macintosh: Operating System Utilities*. ◆

The Speech Recognition Manager also defines a speech attributes selector for `Gestalt`. You can use this selector to get information about the available speech sources. See “Gestalt Selectors and Response Values” on page 1-39 for complete details.

Opening Recognition Systems and Recognizers

Before calling any other Speech Recognition Manager routines, you need to call the `SROpenRecognitionSystem` function to open a recognition system. Listing 1-3 shows how to initialize speech recognition for your application.

Listing 1-3 Initializing speech recognition

```
SRRecognitionSystem            gRecSystem = NULL;      /* our recognition system */
SRRecognizer                   gRecognizer = NULL;     /* our recognizer */

SRRecognitionSystem MyInitSpeechRecognition (void)
{
    OSErr                      myErr;
    SRRecognitionSystem        mySystem = NULL;

    /* Ensure that the Speech Recognition Manager is available. */
    if (MyHasSpeechRecognitionMgr()) {
        /* Open the default recognition system. */
        myErr = SROpenRecognitionSystem(&mySystem, kSRDefaultRecognitionSystemID);

        if (!myErr) {
            /* Use standard feedback window and listening modes. */
            short myModes = kSRHasFeedbackHasListenModes;
            myErr = SRSetProperty(mySystem, kSRFeedbackAndListeningModes,
                                                        &myModes, sizeof(myModes));
        }

        /* Set reference constant of rejected word. */
        if (!myErr)
            myErr = MySetRejectedWordRefCon(mySystem);
    }

    return (mySystem);
}
```

Note that the `MyInitSpeechRecognition` function defined in Listing 1-3 explicitly sets the feedback and listening modes property of the recognition system to the value `kSRHasFeedbackHasListenModes` immediately after it calls the `SROpenRecognitionSystem` function. This instructs the recognition system to use the standard feedback and listening behavior for any recognizers associated with it, which helps ensure that the user has a consistent experience when using any applications that use the Speech Recognition Manager. In general, you should use other feedback and listening modes values only if your application provides its own feedback mechanism.

`MyInitSpeechRecognition` also calls `MySetRejectedWordRefCon` (shown in Listing 1-8 on page 1-26) to simplify the processing of recognition results.

Once you've opened a recognition system, you need to call `SRNewRecognizer` to create a recognizer associated with that recognition system, as follows:

```
gRecSystem = MyInitSpeechRecognition();
if (gRecSystem != NULL)
    myErr = SRNewRecognizer(gRecSystem, &gRecognizer, kSRDefaultSpeechSource);
```

`SRNewRecognizer` takes a reference to an existing recognition system and a **speech source ID,** which specifies a speech source. `SRNewRecognizer` returns a reference to the new recognizer through its second parameter (here, `gRecognizer`).

You can terminate your connection to the Speech Recognition Manager by calling `SRReleaseObject` to dispose of your recognizer and then `SRCloseRecognitionSystem` to close your recognition system, as illustrated in Listing 1-4.

Listing 1-4 Ending speech recognition

```
OSErr MyTerminateSpeechRecognition (void)
{
    OSErr                           myErr;

    /* Stop processing any incoming sound. */
    myErr = SRStopListening(gRecognizer);

    /* Balance call to SRNewRecognizer in MyInitSpeechRecognition. */
    myErr = SRReleaseObject(gRecognizer);

    /* Balance call to SROpenRecognitionSystem in MyInitSpeechRecognition. */
    myErr = SRCloseRecognitionSystem(gRecSystem);

    return(myErr);
}
```

Building a Language Model

You specify the words and phrases you want recognized by assigning a language model to a recognizer. Listing 1-5 illustrates one way to construct a language model. It calls the `SRNewLanguageModel` function to create a new empty language model and then calls other routines to add words and phrases to that model. The function `MyBuildLanguageModel` defined in Listing 1-5 constructs a language model for the BNF diagram shown on page 1-12 (and listed in the comments at the beginning of the listing).

IMPORTANT

There are many other ways to create language models. For example, you might read the basic words and phrases from resources of type `'STR#'`, or you might read language objects from a resource or data file using the functions `SRNewLanguageObjectFromHandle` or `SRNewLanguageObjectFromDataFile` (as shown later, in "Saving and Loading Language Objects" on page 1-37). The techniques used in Listing 1-5 are intended only to illustrate one way of building embedded language models and a top-level language model. ▲

The function `MyBuildLanguageModel` takes an existing recognition system as a parameter. See "Opening Recognition Systems and Recognizers" on page 1-18 for information on opening recognition systems.

Listing 1-5 Creating a language model

```
SRLanguageModel MyBuildLanguageModel (SRRecognitionSystem mySystem);

/* creates language model for the following BNF:
<TopLM>     = <call> <person> | schedule meeting with <person> | view today's schedule;
<call>      = call | phone | dial;
<person>    = Arlo | Matt | Brent | my wife;
*/

const long      kTopLMRefCon = 'top ';
const long      kCallPersonRefCon = 'call';

const char      kCallLMName[] = "<call>";
```

```
const char *    kCallSynonyms[] = {"call", "phone", "dial", NULL};

const char      kPersonLMName[] = "<person>";
const char *    kPersonNames[] = {"Arlo", "Matt", "Brent", "my wife", NULL};

const char      kTopLMName[] = "<TopLM>";
const char      kScheduleMeetingWith[] = "schedule meeting with";
const char      kViewTodaysSchedule[] = "view today's schedule";

SRLanguageModel MyBuildLanguageModel (SRRecognitionSystem mySystem)
{
    OSErr           myErr = noErr;
    SRLanguageModel myCallLM = 0, myPersonLM = 0, myTopLM = 0;
    SRPath          myPath;
    char **         myStringArray;
    char *          myCurrString;

    // create an embedded language model named "<call>"
    myErr = SRNewLanguageModel(mySystem, &myCallLM, kCallLMName, strlen(kCallLMName));
    if (!myErr) {
        SRPhrase    myPhrase;

        myStringArray = (char **) kCallSynonyms;
        while ((myCurrString = *myStringArray) != NULL) {
            /* Note that we call SRNewPhrase instead of SRNewWord */
            /* so that we don't have to know if any of the synonyms */
            /* is more than one word. */
            myErr = SRNewPhrase(mySystem, &myPhrase, myCurrString,
                                                          strlen(myCurrString));
            if (!myErr) {
                myErr = SRAddLanguageObject(myCallLM, myPhrase);
                SRReleaseObject(myPhrase);             /* balances SRNewPhrase */
            }
            myStringArray++;
        }
    }

    /* Create an embedded language model named "<person>". */
    /* Note that this code uses SRAddText, a useful shortcut */
    /* which calls SRNewPhrase internally. */
    if (!myErr)
```

```
    myErr = SRNewLanguageModel(mySystem, &myPersonLM, kPersonLMName,
                                                        strlen(kPersonLMName));
if (!myErr) {
    long        myRefCon = 0;

    myStringArray = (char **) kPersonNames;
    while ((myCurrString = *myStringArray) != NULL) {
        myErr = SRAddText(myPersonLM, myCurrString, strlen(myCurrString),
                                                                myRefCon++);
        /* Note that we set the refcon in the SRAddText call, so */
        /* we can use it later when processing the search result. */
        myStringArray++;
    }
}

/* Create a top-level language model named "<TopLM>". */
if (!myErr)
    myErr = SRNewLanguageModel(mySystem, &myTopLM, kTopLMName, strlen(kTopLMName));

/* We set the refcon of the top-level language model, */
/* so that we can identify it in the search result. */
if (!myErr)
    myErr = SRSetProperty(myTopLM, kSRRefCon, &kTopLMRefCon, sizeof(kTopLMRefCon));

/* Create a path for "<call> <person>" and add to "<TopLM>". */
if (!myErr) myErr = SRNewPath(mySystem, &myPath);
if (!myErr) {
    if (!myErr) myErr = SRAddLanguageObject(myPath, myCallLM);
    if (!myErr) myErr = SRAddLanguageObject(myPath, myPersonLM);
    if (!myErr) myErr = SRAddLanguageObject(myTopLM, myPath);

    /* We set the refcon of the path, so that we can identify */
    /* it in the search result. */
    if (!myErr)
        myErr = SRSetProperty(myPath, kSRRefCon, &kCallPersonRefCon,
                                                sizeof(kCallPersonRefCon));
    SRReleaseObject(myPath);                    /* balances SRNewPath */
}

/* Create a path for "schedule meeting with <person>" and add to "<TopLM>". */
if (!myErr) myErr = SRNewPath(mySystem, &myPath);
```

```
    if (!myErr) {
        if (!myErr) myErr = SRAddText(myPath, kScheduleMeetingWith,
                                                strlen(kScheduleMeetingWith), 0);
        if (!myErr) myErr = SRAddLanguageObject(myPath, myPersonLM);
        if (!myErr) myErr = SRAddLanguageObject(myTopLM, myPath);
        SRReleaseObject(myPath);                    /* balances SRNewPath */
    }

    /* Add "view today's schedule" to "<TopLM>". */
    if (!myErr)
        myErr = SRAddText(myTopLM, kViewTodaysSchedule,
                                            strlen(kViewTodaysSchedule), 0);

    if (myCallLM)
        SRReleaseObject(myCallLM);              /* balances SRNewLanguageModel */
    if (myPersonLM)
        SRReleaseObject(myPersonLM);            /* balances SRNewLanguageModel */

    return myTopLM;
}
```

The `MyBuildLanguageModel` function creates the embedded language models used in the example language model and then adds them to the top-level language model `myTopLM`. As you can see, you can add a word or a phrase to a language model in several ways. You can use the sequence of functions `SRNewPhrase`, `SRAddLanguageObject`, and `SRReleaseObject`. Alternatively, you can use the function `SRAddText`, which adds some text directly to a language model without having you explicitly create objects for that text.

Also, notice that calls to `SRNew`*Object* are balanced with calls to `SRReleaseObject` for any object reference that won't be needed outside the routine (namely, all objects except `myTopLM`, which is returned at the end of the routine).

Note particularly that the `MyBuildLanguageModel` function defined in Listing 1-5 sets the `kSRRefCon` property of the various language objects it creates to known values. For example, `MyBuildLanguageModel` sets the `kSRRefCon` property of a name to an index into the array of names, like this:

```
myErr = SRAddText(myPersonLM, myCurrString,
                                    strlen(myCurrString), myRefCon++);
```

Later, when interpreting recognition results, you can read a language object's `kSRRefCon` property to determine what the user said. See "Interpreting Recognition Results," beginning on page 1-33 for details.

Often, you'll want to designate certain words or phrases as optional. For example, you might wish to let the user say "meeting with Arlo" in addition to "schedule meeting with Arlo" (that is, making the word "schedule" optional). You can designate any language object as optional by setting its optional property, as illustrated in Listing 1-6.

Listing 1-6 Making a word optional

```
/* suppose that the word myWord has already been created */
Boolean     myVal = TRUE;

myErr = SRSetProperty(myWord, kSROptional, &myVal, sizeof(myVal));
```

Setting the Language Model

To use a language model for recognition, you must attach it to a recognizer. You can attach a language model to a recognizer by calling the `SRSetLanguageModel` function, as shown in Listing 1-7.

Listing 1-7 Setting the active language model

```
myErr = SRSetLanguageModel(gRecognizer, myTopLM);
```

The recognizer maintains its own reference to the active language model, so you can (if you wish) release your reference to the active language model immediately after the `SRSetLanguageModel` call. If you later need a reference to the active language model, you can call the `SRGetLanguageModel` function.

Setting the Rejection Word

Sometimes the recognizer can determine that the user has spoken but cannot identify the spoken utterance as matching any phrase in the active language model. In this case, the recognizer may indicate it has rejected the utterance by

returning a special language model object in the recognition result's `kSRLanguageModelFormat` property. (For complete details, see "Recognition Result Properties" on page 1-49.)

By default, the returned object is a word (of type `SRWord`) whose spelling is "???". Listing 1-8 shows how your application can set the rejection word's `kSRRefCon` value to a unique value. Your application can use this value when processing recognition results (as in Listing 1-17 on page 1-35) to determine that an utterance has been rejected.

Listing 1-8 Setting the rejection word's reference constant value

```
const long kRejectedWordRefCon = 'rejc';

OSErr MySetRejectedWordRefCon (SRRecognitionSystem mySystem)
{
    OSErr           myErr = noErr;
    SRWord          myRejectedWord = 0;
    Size            myLen = sizeof(myRejectedWord);

    /* Get a reference to the rejected word. */
    myErr = SRGetProperty(mySystem, kSRRejectedWord, &myRejectedWord, &myLen);

    /* Set the refcon of the rejected word, so we can */
    /* use it later when processing the search result. */
    if (!myErr)
        myErr = SRSetProperty(myRejectedWord, kSRRefCon,
                            &kRejectedWordRefCon, sizeof(kRejectedWordRefCon));

    if (myRejectedWord)
        SRReleaseObject(myRejectedWord);            /* balance SRGetProperty */

    return myErr;
}
```

Starting and Stopping Speech Recognition

Once you've assigned a language model to a recognizer, you can have the recognizer start processing sound and recognizing utterances by calling the `SRStartListening` function:

```
myErr = SRStartListening(gRecognizer);
```

After you've called `SRStartListening`, the recognizer listens for utterances and reports its results to your application by sending it recognition notifications. See the next section for information on processing these notifications.

You can stop a recognizer from listening and reporting results to your application by calling the `SRStopListening` function:

```
myErr = SRStopListening(gRecognizer);
```

You can resume listening by calling the `SRStartListening` function.

Handling Recognition Notifications

By default, a recognizer sends recognition notifications to your application's Apple event handler. If you wish, you can instruct a recognizer to send notifications to a speech recognition callback routine. See "Using Callback Routines" on page 1-30 for information on using callback routines.

IMPORTANT

You should use an Apple event handler to receive and process recognition notifications, unless your software is not an application (for example, a control panel or other software that cannot easily accept Apple events). You cannot process recognition results from within a callback handler, but you can do so from within an Apple event handler. ▲

You need to specify to the recognizer what events you want to be notified about. By default, your application is notified when the recognizer has finished recognizing an utterance. You can also be notified when the recognizer begins the process of recognizing an utterance (that is, when the user begins speaking). You do this by setting the notification property of a recognizer, as shown in Listing 1-9.

Listing 1-9 Requesting notification of a recognition beginning

```
unsigned long myFlags;

myFlags = kSRNotifyRecognitionBeginning | kSRNotifyRecognitionDone;
myErr = SRSetProperty(gRecognizer, kSRNotificationParam,
                                              &myFlags, sizeof(myFlags));
```

IMPORTANT
If you enable recognizer notification for the beginning of a recognition, then each time your application receives a recognition notification, you *must* call either `SRContinueRecognition` or `SRCancelRecognition` before speech recognition can continue. Otherwise, the recognizer will suspend its operations while waiting for you to call one of these functions. ▲

In general, you need to receive `kSRNotifyRecognitionBeginning` notifications only if you want to update or modify the active language model according to context at the start of each utterance.

Using Apple Events

By default, the Speech Recognition Manager uses Apple events to inform your application of recognizer events. To receive notifications through Apple events, you need to install Apple event handlers for the events in the speech class during application startup, as illustrated in Listing 1-10.

Listing 1-10 Installing an Apple event handler for speech events

```
myErr = AEInstallEventHandler(kAESpeechSuite, kAESpeechDetected,
                NewAEEventHandlerProc(MyHandleSpeechDetected), 0, FALSE);
if (!myErr)
    myErr = AEInstallEventHandler(kAESpeechSuite, kAESpeechDone,
                NewAEEventHandlerProc(MyHandleSpeechDone), 0, FALSE);
```

Listing 1-11 illustrates how to respond to a recognizer notification of type `kSRNotifyRecognitionDone` sent to an Apple event handler.

Listing 1-11 Handling recognition done notifications with an Apple event handler

```
pascal OSErr MyHandleSpeechDone (AppleEvent *theAEevt, AppleEvent *reply, long refcon)
{
    long                        actualSize;
    DescType                    actualType;
    OSErr                       recStatus = noErr, myErr = noErr
    SRRecognitionResult         recResult;
    SRRecognizer                myRec;

    /* Get recognition result status and recognizer. */
    myErr = AEGetParamPtr(theAEevt, keySRSpeechStatus, typeShortInteger,
                &actualType, (Ptr)&recStatus, sizeof(recStatus), &actualSize);
    if (!myErr)
        myErr = recStatus;

    if (!myErr) {
        myErr = AEGetParamPtr(theAEevt, keySRRecognizer, typeSRRecognizer,
                &actualType, (Ptr)&myRec, sizeof(myRec), &actualSize);

        if (!myErr) {
            myErr = AEGetParamPtr(theAEevt, keySRSpeechResult, typeSRSpeechResult,
                    &actualType, (Ptr)&recResult, sizeof(recResult), &actualSize);
            if (!myErr) {
                /* Process the recognition result here.*/
                MyProcessRecognitionResult(recResult);

                /* Release the processed result. */
                SRReleaseObject(recResult);
            }
        }
    }

    return(myErr);
}
```

(The function `MyProcessRecognitionResult` is shown on page 1-35.) Listing 1-12 illustrates how to respond to a recognition notification of type `kSRNotifyRecognitionBeginning` sent to an Apple event handler.

Listing 1-12 Handling recognition beginning notifications with an Apple event handler

```
pascal OSErr MyHandleSpeechDetected (AppleEvent *theAEevt,
                                        AppleEvent *reply, long refcon)
{
    OSErr           myErr = noErr, recStatus = 0;
    DescType        actualType;
    long            actualSize;
    SRRecognizer    myRec;

    /* Get status and recognizer. */
    myErr = AEGetParamPtr(theAEevt, keySRSpeechStatus, typeShortInteger,
                    &actualType, (Ptr)&recStatus, sizeof(recStatus), &actualSize);
    if (!myErr)
        myErr = recStatus;

    if (!myErr) {
        myErr = AEGetParamPtr(theAEevt, keySRRecognizer, typeSRRecognizer,
                    &actualType, (Ptr)&myRec, sizeof(myRec), &actualSize);
        if (!myErr) {
            /* The user has started speaking. We can adjust the language model */
            /* to reflect the current context. Then we must call either */
            /* SRContinueRecognition or SRCancelRecognition. */

            myErr = SRContinueRecognition(myRec);
        }
    }

    return(myErr);
}
```

Using Callback Routines

To instruct a recognizer to send notifications using a speech recognition callback routine instead of using Apple events, you set the `kSRCallBackParam` property of the recognizer to the address of a **callback routine parameter structure,** which specifies the address of your callback routine, as shown in Listing 1-13.

IMPORTANT
You should use an Apple event handler to receive and process recognition notifications, unless your software cannot easily accept Apple events. ▲

Listing 1-13 Installing a speech recognition callback routine

```
pascal OSErr MyInstallNotificationCallBack (SRRecognizer recognizer)
{
    SRCallBackParam         myCallBackPB;

    myCallBackPB.callBack = NewSRCallBackProc(MyNotificationCallBack);

    return SRSetProperty(recognizer, kSRCallBackParam,
                            &myCallBackPB, sizeof (myCallBackPB));
}
```

Note
See page 1-55 for details on the callback routine parameter structure (of type `SRCallBackParam`). ◆

You can remove a speech recognition callback routine, as shown in Listing 1-14.

Listing 1-14 Removing a speech recognition callback routine

```
pascal void MyRemoveNotificationCallBack (void)
{
    SRCallBackParam             myCallBackPB;
    SRCallBackUPP               mySavedCallBack;
    Size                        myLen;
    OSErr                       myErr = noErr;

    myLen = sizeof(myCallBackPB);
    myErr = SRGetProperty(gRecognizer, kSRCallBackParam, &myCallBackPB, &myLen);
    if (myErr == noErr) {
        if (myCallBackPB.callBack != nil) {
            mySavedCallBack = myCallBackPB.callBack;
            myCallBackPB.callBack = nil;
            myErr = SRSetProperty(gRecognizer, kSRCallBackParam, &myCallBackPB,
```

```
                                                            sizeof(myCallBackPB));
            DisposeRoutineDescriptor(mySavedCallBack);
        }
    }
}
```

IMPORTANT

You should not call any Speech Recognition Manager routines other than `SRContinueRecognition` or `SRCancelRecognition` in your speech recognition callback routine. Usually, your callback routine should simply queue the notification it receives for later processing by your software (for instance, when you receive background processing time). ▲

You can queue the notification by setting a global flag that indicates the recognition result to process, as shown in Listing 1-15.

Listing 1-15 Handling notifications with a callback routine

```
/* last recognition result received */
SRRecognitionResult                             gLastRecResult;

pascal void MyNotificationCallBack (SRCallBackStruct * param)
{
    OSErr   myErr = param->status;

    if (!myErr) {
        /* Handle recognition beginning event. */
        /* Here we just continue speech recognition. */
        if ((param->what) & kSRNotifyRecognitionBeginning) {
            SRRecognizer    myRec = (SRRecognizer) (param->instance);
            myErr = SRContinueRecognition(myRec);
        }

        /* Handle recognition done event. */
        /* Here we save the rec result in gLastRecResult. */
        /* At idle time in our event loop, if gLastRecResult != NULL, */
        /* we call MyProcessRecognitionResult(gLastRecResult) */
        else if (param->what & kSRNotifyRecognitionDone) {
```

```
            SRRecognitionResult myRecResult =
                                (SRRecognitionResult) (param->message);
            if (myRecResult)
                gLastRecResult = myRecResult;
                /* We might get more than one result before we get to */
                /* our idle check, so we should really be putting */
                /* this in a queue. */
        }
    }
}
```

The `MyNotificationCallBack` function simply determines what event prompted the recognition notification and sets a global flag to signal the application to do the correct thing (for example, process the recognition result). Your application needs to examine that flag periodically to determine whether to handle the result. Listing 1-16 shows an example of a routine that does this.

Listing 1-16 Checking whether a recognition result needs processing

```
pascal void MyIdleCheckForSpeechResult (void)
{
    if (gLastRecResult != NULL)
        MyProcessRecognitionResult(gLastRecResult);

    gLastRecResult = NULL;
}
```

If a recognition result is pending, the application calls its routine to handle recognition results, `MyProcessRecognitionResult` (defined in the next section).

IMPORTANT

See “Speech Recognition Callback Routines” on page 1-89 for a complete description of the limitations of using callback routines. ▲

Interpreting Recognition Results

A recognition result contains information about a recognized utterance. The standard way to determine what the user said is to read the language model

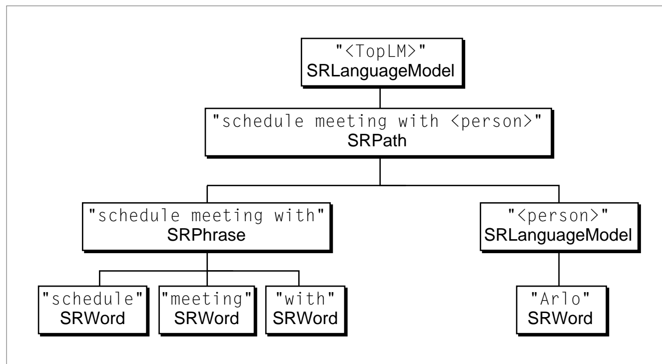
format property of the recognition result. The value of this property is a language model that contains a copy of each word, phrase, path, and language model used in the recognized utterance. You can inspect that language model to determine precisely what the user said and then respond accordingly. To retrieve this language model, call `SRGetProperty`, as follows:

```
SRLanguageModel         myResultLM;
Size                    myLen;

myLen = sizeof(myResultLM);
myErr = SRGetProperty(recResult, kSRLanguageModelFormat,
                                              &myResultLM, &myLen);
```

The language model returned by `SRGetProperty` is a subset of the complete active language model. For instance, the language model returned for the utterance "schedule meeting with Arlo" has the structure shown in Figure 1-7. This is a subset of the language model shown in Figure 1-3 on page 1-13.

Figure 1-7 A language model for a recognized utterance



You can traverse this language model starting from its root by calling `SRCountItems` and `SRGetIndexedItem`. Then, for any item returned, you can inspect its type and reference constant to identify it. The `MyProcessRecognitionResult` function defined in Listing 1-17 shows how to process recognition results for the sample language model built in Listing 1-5 on page 1-21.

Listing 1-17 Handling a recognition result

```
void MyProcessRecognitionResult (SRRecognitionResult recResult)
{
    OSErr               myErr = noErr;

    if (recResult) {
        SRLanguageModel         myResultLM;
        Size                    myLen;

        myLen = sizeof(myResultLM);
        myErr = SRGetProperty(recResult, kSRLanguageModelFormat, &myResultLM, &myLen);

        if (!myErr) {
            long                myRefCon;
            SRLanguageObject    mySubElement;

            myLen = sizeof(myRefCon);
            myErr = SRGetProperty(myResultLM, kSRRefCon, &myRefCon, &myLen);

            /* if it's a valid result from our top-level LM, */
            /* then parse and process its elements */
            if (!myErr) {
                switch (myRefCon) {
                    case kRejectedWordRefCon:
                        /* do nothing, if utterance was rejected */
                        break;
                    case kTopLMRefCon:
                        /* parse and process element of top-level LM */
                        /* get the first sub-element (here there is only one) */
                        myErr = SRGetIndexedItem(myResultLM, &mySubElement, 0);
                        if (!myErr) {
                            myLen = sizeof(myRefCon);
                            myErr = SRGetProperty(mySubElement, kSRRefCon, &myRefCon,
                                                                        &myLen);
                            /* Call a subroutine to process the subelement */
                            if (!myErr) switch (myRefCon) {
                                case kCallPersonRefCon:
                                    myErr = MyCallPersonInPath((SRPath) mySubElement);
                                    break;
                                /* ...process "schedule meeting with <person>" */
```

```
                        /* ...process "view today's schedule" */

                        default:
                            break;
                    }
                }
                /* release subelement when done with it */
                myErr = SRReleaseObject(mySubElement);
                break;
        }
    }

        /* Release myResultLM fetched above when done with it. */
        myErr = SRReleaseObject(myResultLM);
    }

    /* Release SRRecognitionResult because we are done with it. */
    myErr = SRReleaseObject(recResult);
  }
}
```

When `MyProcessRecognitionResult` determines that the first item in the language model is an object whose reference constant is `kCallPersonRefCon`, it dispatches to the application-defined function `MyCallPersonInPath`, defined in Listing 1-18, which continues parsing the language model to determine whom to call.

Listing 1-18 Calling a name in a path

```
const char *    kPhoneNumbers[] =
                {"555-4567", "555-4568", "555-4569", "(123) 456-7890", NULL};

OSErr MyCallPersonInPath (SRPath recognizedPath)
{
    OSErr                   myErr = noErr;
    SRLanguageObject        myPersonLM;

    /* recognizedPath has two sub-elements: */
    /* "<call>" (SRLanguageModel) and "<person>" (SRLanguageModel) */
```

```
    /* Here we get the second sub-element. */
    myErr = SRGetIndexedItem(recognizedPath, &myPersonLM, 1);
    if (!myErr) {
        SRLanguageObject    myPhraseSpoken;

        /* myPersonLM has one sub-element, the name (a phrase) */
        myErr = SRGetIndexedItem(myPersonLM, &myPhraseSpoken, 0);
        if (!myErr) {
            long            myRefCon;
            Size            myLen = sizeof(myRefCon);

            /* When we built the language model, we set the refcon to the index */
            /* of names in a list. The phone numbers in kPhoneNumbers correspond */
            /* to those names */
            myErr = SRGetProperty(myPhraseSpoken, kSRRefCon, &myRefCon, &myLen);
            if (!myErr) {
                short       myArrayIndex = myRefCon;

                printf("Now calling %s. Dialing %s.\n",
                        kPersonNames[myArrayIndex], kPhoneNumbers[myArrayIndex]);
            }

            /* release myPhraseSpoken when done with it */
            myErr = SRReleaseObject(myPhraseSpoken);
        }

        /* release myPersonLM when done with it */
        myErr = SRReleaseObject(myPersonLM);
    }

    return myErr;
}
```

Note that both `MyProcessRecognitionResult` and `MyCallPersonInPath` are careful to release any language object references when they are no longer needed.

Saving and Loading Language Objects

The Speech Recognition Manager provides several functions that you can use to put a language object into the data or resource fork of a file and to load a

saved language object from a file. For example, you can save a language model in a resource by calling the `SRPutLanguageObjectIntoHandle` function. This function creates a description of the language model and puts it into a block of memory you've already allocated, resizing the handle as necessary and overwriting any existing contents. You call `SRPutLanguageObjectIntoHandle` like this:

```
myErr = SRPutLanguageObjectIntoHandle(myLModel, myHandle);
```

If `SRPutLanguageObjectIntoHandle` returns successfully, you can then use standard Resource Manager routines (for example, `AddResource`) to add the data in the handle to a file's resource fork. The resource type and ID of the new resource are specified by your application.

You can read a saved language object from a resource by loading the resource using Resource Manager functions (for example, `GetResource`) and then converting the resource data into a language object by calling the `SRNewLanguageObjectFromHandle` function, like this:

```
myErr = SRNewLanguageObjectFromHandle(mySys, myLModel, myHandle);
```

The format of the language object data stored in a handle (or data file) is private. You should always use the supplied Speech Recognition Manager functions to read and write language object data.

Speech Recognition Manager Reference

This section describes the constants, data structures, and routines provided by the Speech Recognition Manager.

Note
This chapter describes the application programming interfaces supported by version 1.5 and later of the Speech Recognition Manager. ◆

Constants

This section describes the constants provided by the Speech Recognition Manager.

Gestalt Selectors and Response Values

You can pass the `gestaltSpeechRecognitionVersion` selector to the `Gestalt` function to determine the version of the Speech Recognition Manager installed on a computer.

```
enum {
    gestaltSpeechRecognitionVersion                = 'srtb',
    gestaltSpeechRecognitionAttr                   = 'srta'
};
```

`Gestalt` returns a long word in the `response` parameter that is the current version number of the Speech Recognition Manager. If the value of the `response` parameter is 0x00000000 (or if `Gestalt` returns an error), the Speech Recognition Manager is not available on the target computer.

You can pass the `gestaltSpeechRecognitionAttr` selector to the `Gestalt` function to get the attributes of the Speech Recognition Manager. `Gestalt` returns information to you by returning a long word in the `response` parameter. The returned values are defined by constants:

```
enum {
    gestaltDesktopSpeechRecognition                = 1L<<0,
    gestaltTelephoneSpeechRecognition              = 1L<<1
};
```

Constant descriptions

`gestaltDesktopSpeechRecognition`
: If this bit is set, the Speech Recognition Manager supports the desktop microphone.

`gestaltTelephoneSpeechRecognition`
: If this bit is set, the Speech Recognition Manager supports telephone input. In versions 1.5 and earlier, this bit is always 0.

Recognition System IDs

When you call `SROpenRecognitionSystem` to open a recognition system, you indicate the system to open by passing a **recognition system ID.** The Speech Recognition Manager defines this constant for recognition system IDs:

```
enum {
    kSRDefaultRecognitionSystemID                    = 0
};
```

Constant descriptions

`kSRDefaultRecognitionSystemID`
: The default speech recognition system.

Recognition System Properties

A recognition system (that is, an instance of the `SRRecognitionSystem` class) has a set of properties that you can inspect and change by calling the `SRGetProperty` and `SRSetProperty` routines. You specify a property by passing a property selector to those functions. The Speech Recognition Manager defines these property selectors for recognition systems:

```
enum {
    kSRFeedbackAndListeningModes                     = 'fbwn',
    kSRRejectedWord                                  = 'rejq',
    kSRCleanupOnClientExit                           = 'clup'
};
```

Constant descriptions

`kSRFeedbackAndListeningModes`
: The feedback and listening modes of the recognition system. The value of this property is an integer that determines some of the features of a recognizer subsequently created by your application. The available values are described below. The default value for version 1.5 is `kSRNoFeedbackNoListenModes`, but most applications should set this to `kSRHasFeedbackHasListenModes`.

```
enum {
    kSRNoFeedbackNoListenModes            = 0,
    kSRHasFeedbackHasListenModes          = 1,
    kSRNoFeedbackHasListenModes           = 2
};
```

If the feedback and listening modes value of a recognition system is set to `kSRNoFeedbackNoListenModes`, the next created recognizer has no feedback window and doesn't use the listening modes selected by the user in the Speech control panel. (For example, push-to-talk is a listening mode.) If the feedback and listening modes value of a recognition system is set to `kSRHasFeedbackHasListenModes`, the next created recognizer opens a feedback window that uses the listening modes selected by the user in the Speech control panel. If the feedback and listening modes value of a recognition system is set to `kSRNoFeedbackHasListenModes`, the next created recognizer has no feedback window but does use the listening modes selected by the user in the Speech control panel.

`kSRRejectedWord` The rejected word of the recognition system. The value of this property is a value of type `SRWord` that will be returned in a recognition result object when a recognizer encounters an unrecognizable utterance. For example, if an utterance is rejected, the `kSRLanguageModelFormat` property of the recognition result is the rejected word. By default, a recognition system's rejected word is spelled "???" and has a reference constant of 0. (For complete details, see "Recognition Result Properties" on page 1-49.)

`kSRCleanupOnClientExit`
The cleanup mode of the recognition system. Applications should never change the value of this property from its default value `TRUE`. If, however, you write some code other than an application that doesn't have a process ID (as issued by the Process Manager), you should set this property to `FALSE` so that speech objects you allocate will not be associated with any other process. By default, the value of a recognition system's cleanup mode is `TRUE`.

Apple Event Selectors

The Speech Recognition Manager defines a number of selectors that you can use to handle recognition notifications in your Apple events handler.

```
enum {
    kAESpeechSuite                                  = 'sprc'
};
```

Constant descriptions

`kAESpeechSuite` The Apple event class for speech recognition events.

```
enum {
    kAESpeechDetected                               = 'srbd',
    kAESpeechDone                                   = 'srsd'
};
```

Constant descriptions

`kAESpeechDetected` The event ID for a speech-detected event.

`kAESpeechDone` The event ID for a speech-done event.

```
enum {
    keySRRecognizer                                 = 'krec',
    keySRSpeechResult                               = 'kspr',
    keySRSpeechStatus                               = 'ksst'
};
```

Constant descriptions

`keySRRecognizer` The keyword for the recognizer parameter.

`keySRSpeechResult` The keyword for the recognition result parameter.

`keySRSpeechStatus` The keyword for the speech status parameter, which is of type `typeShortInteger`.

```
enum {
    typeSRRecognizer                                = 'trec',
    typeSRSpeechResult                              = 'tspr'
};
```

Constant descriptions

`typeSRRecognizer` The type for the recognizer parameter.

`typeSRSpeechResult` The type for the recognition result parameter.

Recognizer Properties

Every recognizer has a set of properties that you can inspect and change by calling the `SRGetProperty` and `SRSetProperty` routines. You specify a property by passing a property selector to those functions. The Speech Recognition Manager defines these property selectors for recognizers:

```
enum {
    kSRNotificationParam                    = 'noti',
    kSRCallBackParam                        = 'call',
    kSRSearchStatusParam                    = 'stat',
    kSRForegroundOnly                       = 'fgon',
    kSRBlockBackground                      = 'blbg',
    kSRBlockModally                         = 'blmd',
    kSRWantsResultTextDrawn                 = 'txfb',
    kSRWantsAutoFBGestures                  = 'dfbr',
    kSRSoundInVolume                        = 'volu',
    kSRReadAudioFSSpec                      = 'aurd',
    kSRCancelOnSoundOut                     = 'caso',
    kSRListenKeyMode                        = 'lkmd',
    kSRListenKeyCombo                       = 'lkey',
    kSRListenKeyName                        = 'lnam',
    kSRKeyWord                              = 'kwrd',
    kSRKeyExpected                          = 'kexp'
};
```

Note
The listen key properties (that is, `kSRListenKeyMode` through `kSRKeyExpected`) are provided for use by applications that want to provide their own visual feedback. If your application uses the default feedback mechanisms, you do not need to access those properties. ◆

Constant descriptions

`kSRNotificationParam`
: The notification property. The value of this property is a 4-byte unsigned integer whose bits encode the kinds of events of which the recognizer will notify your

application. See the section “Notification Flags” on page 1-48 for the bit masks that are defined for this property. By default, the value of a recognizer’s notification property is `kSRNotifyRecognitionDone`.

`kSRCallBackParam` The callback property. The value of this property is of type `SRCallBackParam` that determines whether recognition notifications are sent to your application via Apple events or via an application-defined callback routine. To specify a callback routine, set the value of this property to the address of a callback routine parameter structure. By default, the value of a recognizer’s callback property is `NULL`, indicating that Apple events are to be used to report recognizer events.

`kSRSearchStatusParam`
The search status. The value of this property is a 4-byte unsigned integer whose bits indicate the current state of the recognizer. See the section “Search Status Flags” on page 1-47 for the bit masks that are defined for this property. This property is read-only; you cannot set it.

`kSRForegroundOnly` The foreground-only flag. The value of this property is a Boolean value that indicates whether the recognizer is enabled only when your application is the foreground application (`TRUE`) or not (`FALSE`). By default, the value of a recognizer’s foreground-only flag is `TRUE`.

`kSRBlockBackground`
The background-blocking flag. The value of this property is a Boolean value that indicates whether all recognizers owned by other applications are automatically disabled whenever your application is the foreground application (`TRUE`) or are not automatically disabled (`FALSE`). By default, the value of a recognizer’s background-blocking flag is `FALSE`.

`kSRBlockModally` The modal-blocking flag. The value of this property is a Boolean value that indicates whether the language model associated with this recognizer is the only active language model (`TRUE`) or not (`FALSE`). When this flag is `TRUE`, your application’s recognizer blocks those of other applications even when it isn’t the foreground application; in addition, the feedback window is hidden if you’re not using it. Setting this property to `TRUE` prevents speech recognition

from working for other applications, so you want to use this property only if your application is taking over the computer (like some games) or briefly attempting to constrain the language model severely. By default, the value of a recognizer's modal-blocking flag is `FALSE`.

`kSRWantsResultTextDrawn`
The text feedback flag. The value of this property is a Boolean value that indicates whether the results of a search are to be automatically displayed as text in the feedback window (`TRUE`) or not (`FALSE`). If you set the value of this property to `FALSE`, you should call `SRDrawRecognizedText` with a string representing what the user said. By default, the value of a recognizer's text feedback flag is `TRUE`.

`kSRWantsAutoFBGestures`
The automatic feedback gestures flag. The value of this property is a Boolean value that determines whether the feedback gestures are automatically drawn (`TRUE`) or not (`FALSE`). If you want more control over feedback behavior, you should set this property to `FALSE`; then call `SRProcessBegin` when you want to begin responding to a spoken request and `SRProcessEnd` when you are finished. During that time, the feedback character displays appropriate animated gestures to indicate that it's busy performing the task. By default, the value of a recognizer's automatic feedback gestures flag is `TRUE`.

`kSRSoundInVolume`
The sound input volume. The value of this property is a 2-byte unsigned integer between 0 and 100, inclusive, that indicates the current sound input volume. This property is read-only; you cannot set it.

`kSRReadAudioFSSpec`
The audio file property. You can use this property to perform speech recognition from an audio file. The value of this property is a pointer to a file system specification (a structure of type `FSSpec`). The file system specification indicates an AIFF file that contains raw audio data (16-bit audio data sampled at 22.050 kHz). After you create a new recognizer using the speech source ID `kSRCanned22kHzSpeechSource`, you must set this recognizer property to perform recognition from an audio file. Setting the audio source to a file also allows the Speech

Recognition Manager to process sound data at system background time rather than at interrupt time or deferred task time.

`kSRCancelOnSoundOut`
The cancel during sound output flag. The value of this property is a Boolean value that indicates whether speech recognition is canceled whenever any sound is output by the computer during an utterance (`TRUE`) or whether speech recognition continues (`FALSE`). By default, the value of a recognizer's cancel during sound output flag is `TRUE`.

`kSRListenKeyMode` The **listen key mode.** The value of this property is a 2-byte unsigned integer that indicates whether the listen key operates in push-to-talk or toggle-listening mode. See page 1-49 for a description of the available listen key modes. The value of a recognizer's listen key mode is whatever the user has selected in the Speech control panel. This property is read-only; you cannot set it.

`kSRListenKeyCombo` The listen key combination property. The value of this property is a 2-byte unsigned integer that specifies the key combination the user must press for the listen key. The high-order byte of this value has the same format as the high-order byte of the `modifiers` field of an event record. The low-order byte of this value has the same format as the key code contained in the `message` field of an event record. (See *Inside Macintosh: Macintosh Toolbox Essentials* for complete information about event records.) The value of a recognizer's listen key combination property is whatever the user has selected in the Speech control panel. This property is read-only; you cannot set it.

`kSRListenKeyName` The listen key name property. The value of this property is a string (of type `Str63`) that represents the listen key combination specified by the `kSRListenKeyCombo` property. The value of a recognizer's listen key name property is whatever the user has selected in the Speech control panel. This property is read-only; you cannot set it.

`kSRKeyWord` The key word property. The value of this property is a string (of type `Str255`) that represents the key word that must precede utterances when the recognizer is in toggle-listen mode. The value of a recognizer's key word

property is whatever the user has selected in the Speech control panel. This property is read-only; you cannot set it.

`kSRKeyExpected` The key expected flag. The value of this property is a Boolean value that indicates whether the recognizer expects the user to hold down a key or to utter the key word in order to have the recognizer begin listening (`TRUE`) or not (`FALSE`). The value of a recognizer's key expected flag is a function of the user's Speech control panel selections. This property is `TRUE` whenever text is visible below the feedback character in the lower-left corner of the feedback window. This property is read-only; you cannot set it.

Search Status Flags

You can determine the current status of a recognizer search by getting the recognizer's search status, which is a property of type `kSRSearchStatusParam`. That property's value is a 4-byte unsigned integer. The Speech Recognition Manager defines the following masks for bits in that value:

```
enum {
    kSRIdleRecognizer                          = 1L<<0,
    kSRSearchInProgress                        = 1L<<1,
    kSRSearchWaitForAllClients                 = 1L<<2,
    kSRMustCancelSearch                        = 1L<<3,
    kSRPendingSearch                           = 1L<<4
};
```

Constant descriptions

`kSRIdleRecognizer` If this bit is set, the search engine is not active and the user is able to make a new utterance.

`kSRSearchInProgress`
If this bit is set, a search is currently in progress.

`kSRSearchWaitForAllClients`
If this bit is set, a search is not currently in progress, but will begin as soon as every recognizer using the speech source used by this recognizer has called `SRContinueRecognition` to indicate that the search should begin.

`kSRMustCancelSearch`
: If this bit is set, a search is about to be canceled (for example, because the recognizer determined a sound to be non-speech).

`kSRPendingSearch`
: If this bit is set, a search is about to begin.

Notification Flags

You can indicate which recognizer events you want your application to be notified of by setting the recognizer's notification property, which is a property of type `kSRNotificationParam`. That property's value is a 4-byte unsigned integer. The Speech Recognition Manager defines the following masks for bits in that value:

```
enum {
    kSRNotifyRecognitionBeginning                    = 1L<<0,
    kSRNotifyRecognitionDone                         = 1L<<1
};
```

Constant descriptions

`kSRNotifyRecognitionBeginning`
: If this bit is set, your application will be notified when the user starts speaking and recognition is ready to begin. When your application gets this notification, it must call either `SRContinueRecognition` or `SRCancelRecognition` in order for recognition either to continue or be canceled. If you do not call one of these functions, the recognizer will simply wait until you do (and hence appear to have quit working). Note that calling `SRCancelRecognition` cancels a recognition only for the application that requested it, not for all applications.

`kSRNotifyRecognitionDone`
: If this bit is set, your application will be notified when recognition is finished and the result (if any) of that recognition is available. See the example on page 1-29.

Listen Key Modes

You can get (but not set) a recognizer's listen key mode by accessing its property of type `kSRListenKeyMode`. That property's value is a 2-byte unsigned integer that determines whether the listen key operates in push-to-talk or toggle-listening mode. The Speech Recognition Manager defines these constants for listen key modes:

```
enum {
    kSRUseToggleListen                          = 0,
    kSRUsePushToTalk                            = 1
};
```

Constant descriptions

`kSRUseToggleListen` The recognizer interprets presses on the listen key as a toggle to turn listening on or off.

`kSRUsePushToTalk` The recognizer listens only when the listen key is held down.

Recognition Result Properties

Every recognition result object has a set of properties that you can inspect by calling the `SRGetProperty` routine. You specify a property by passing a property selector to those functions. The Speech Recognition Manager defines these property selectors for recognition results:

IMPORTANT

`SRGetProperty` returns an object reference as the value of a recognition result's `kSRPhraseFormat`, `kSRPathFormat`, or `kSRLanguageModelFormat` property. You must make sure to release that object reference (by calling `SRReleaseObject`) when you are finished using it. ▲

```
enum {
    kSRLanguageModelFormat                      = 'lmfm',
    kSRPathFormat                               = 'lmpt',
    kSRPhraseFormat                             = 'lmph',
    kSRTEXTFormat                               = 'TEXT'
};
```

Constant descriptions

`kSRLanguageModelFormat`
: The language model format. The value of this property is a language model that contains a copy of each word, phrase, path, and language model used in the recognized utterance. If the utterance was rejected, the value of this property is the rejected word (that is, the `kSRRejectedWord` property of the recognition system). The name and reference constant of this language model are the same as the name and reference constant of the active language model, and each subitem in the language model retains its own reference constant property value. See "Interpreting Recognition Results," beginning on page 1-33 for information on how to use this language model to interpret results quickly.

`kSRPathFormat`
: The path format. The value of this property is a path that contains a sequence of words (of type `SRWord`) and phrases (of type `SRPhrase`) representing the text of the recognized utterance. If the utterance was rejected, this path or phrase contains one object, the rejected word. The reference constant value of the path is always 0, but each word or phrase in the path retains its own reference constant property value from the original active language model.

`kSRPhraseFormat`
: The phrase format. The value of this property is a phrase that contains one word (of type `SRWord`) for each word in the recognized utterance. If the utterance was rejected, this phrase contains one object, the rejected word. The reference constant value of the phrase is always 0, but each word in the phrase retains its own reference constant property value.

`kSRTEXTFormat`
: The text format. The value of this property is a variable-length string of characters that is the text of the recognized utterance. If the utterance was rejected, this text is the spelling of the rejected word. The string value does not include either a length byte (as in Pascal strings) or a null terminating character (as in C strings).

Language Object Properties

Every language object (that is, any instance of a subclass of the `SRLanguageObject` class) has a set of properties that you can inspect and change by calling the `SRGetProperty` and `SRSetProperty` routines. You specify a property by passing a property selector to those functions. The Speech Recognition Manager defines these property selectors for language objects:

```
enum {
    kSRSpelling                             = 'spel',
    kSRLMObjType                            = 'lmtp',
    kSRRefCon                               = 'refc',
    kSROptional                             = 'optl',
    kSREnabled                              = 'enbl',
    kSRRepeatable                           = 'rptb',
    kSRRejectable                           = 'rjbl',
    kSRRejectionLevel                       = 'rjct'
};
```

Constant descriptions

`kSRSpelling` The spelling of a language object. The value of this property is a variable-length string of characters. For an object of type `SRWord`, the value is the spelled word. For an object of type `SRPhrase`, the value is the concatenation of the spellings of each word in the phrase, separated by a language-dependent separation character (for example, by a space character). For an object of type `SRPath`, the value is the concatenation of the spellings of each word and language model name in the path. For an object of type `SRLanguageModel`, the value is the name of the language model. For any object, the string value does not include either a length byte (as in Pascal strings) or a null terminating character (as in C strings).

`kSRLMObjType` The type of a language object. The value of this property is a four-character constant of type `OSType`; see the section “Language Object Types” on page 1-53 for the values that are defined for this property. You cannot set it.

`kSRRefCon` The reference constant. The value of this property is a 4-byte value specified by your application. By default, the value of a reference constant property is zero (0).

`kSROptional` The optional flag. The value of this property is a Boolean value that indicates whether speaking the words, phrases, paths, and language models represented by the object is optional (`TRUE`) or required (`FALSE`). A user is not required to utter optional words, phrases, or language models. By default, the value of an object's optional flag is `FALSE`.

`kSREnabled` The enabled flag. The value of this property is a Boolean value that indicates whether the object is enabled (`TRUE`) or disabled (`FALSE`). Disabled objects are ignored during speech recognition. By default, the value of an object's enabled flag is `TRUE`.

`kSRRepeatable` The repeatable flag. The value of this property is a Boolean value that indicates whether the object is repeatable (`TRUE`) or not (`FALSE`). A user can utter a repeatable object more than once. By default, the value of an object's repeatable flag is `FALSE`.

`kSRRejectable` The rejectable flag. The value of this property is a Boolean value that indicates whether the object is rejectable (`TRUE`) or not (`FALSE`). An object is **rejectable** if a recognition system can return the rejected word instead of that object. (The rejected word is the value of the `kSRRejectedWord` property of the recognition system.) By default, the value of an object's rejectable flag is `FALSE`. However, if an entire utterance is rejected, you can still get the rejected word. See "Recognition Result Properties" on page 1-49.

`kSRRejectionLevel` The rejection level. The value of this property is a 2-byte unsigned integer of type `SRRejectionLevel` between 0 and 100, inclusive, that determines how likely a recognizer is to reject a language object whose `kSRRejectable` property is `TRUE`. If an object's rejection level is close to 0, the recognizer is less likely to reject utterances (and hence more likely to return a result with phrases from the current language model, whether or not the user actually said something in that language model); if an object's rejection level is close to 100, the recognizer is more likely to reject utterances. You can set an object's rejection flag to `TRUE` and its rejection level to some appropriate value to reduce the likelihood that a recognizer will mistakenly recognize a random user utterance as part of the active language

model. By default, the value of an object's rejection level is 50.

Language Object Types

The Speech Recognition Manager defines constants for the four subclasses of the SRLanguageObject class. You can use these constants, for instance, to help interpret the value of a language object's kSRLMObjType property.

```
enum {
    kSRLanguageModelType                        = 'lmob',
    kSRPathType                                 = 'path',
    kSRPhraseType                               = 'phra',
    kSRWordType                                 = 'word'
};
```

Constant descriptions

kSRLanguageModelType
: A language model (that is, an object of type SRLanguageModel).

kSRPathType
: A path (that is, an object of type SRPath).

kSRPhraseType
: A phrase (that is, an object of type SRPhrase).

kSRWordType
: A word (that is, an object of type SRWord).

Data Structures

This section describes the data structures provided by the Speech Recognition Manager.

Speech Recognition CallBack Structure

When you receive a notification of recognition results through an application-defined callback routine (instead of using an Apple event handler), the Speech Recognition Manager sends your callback routine a pointer to a **speech recognition callback structure,** defined by the SRCallBackStruct data type.

Note
For information on writing a speech recognition callback routine, see "Speech Recognition Callback Routines" on page 1-89. ◆

```
struct SRCallBackStruct {
    long                                    what;
    long                                    message;
    SRRecognizer                            instance;
    OSErr                                   status;
    short                                   flags;
    long                                    refcon;
};
typedef struct SRCallBackStruct SRCallBackStruct;
```

Field descriptions

`what` A notification flag that indicates the kind of event that caused this notification to be issued. This field contains either `kSRNotifyRecognitionBeginning` or `kSRNotifyRecognitionDone`. See "Notification Flags" on page 1-48 for complete details on the available notification flags.

`message` If the value of the `status` field is `noErr` and the value of the `what` field is `kSRNotifyRecognitionDone`, this field contains a reference to a recognition result. Your callback routine can inspect the properties of this recognition result to determine what the user said.

IMPORTANT
Note that your callback routine must release this reference (by calling `SRReleaseObject`) when it is finished using it. If the value of the `status` field is not `noErr`, the value of this field is undefined. ▲

`instance` A reference to the recognizer that issued this notification. You should not call `SRReleaseObject` on this recognizer reference in response to a recognition notification.

`status` An error code indicating the status of the recognition. If the value of this field is `noErr`, the `message` field contains a reference to a recognition result. If the value of this field is `kSRRecognitionDone` and the value of the `what` field is

`kSRNotifyRecognitionDone`, the recognizer finished without error but nothing belonging to that recognizer was recognized; in this case, the `message` field does not contain a reference to a recognition result. If the value of this field is any other value, some other error occurred.

`flags` Reserved.

`refcon` An application-defined reference constant. The value in this field is the value you passed in the `refcon` field of a callback routine parameter structure (of type `SRCallBackParam`).

Callback Routine Parameter Structure

If you want to receive recognition notifications using a speech recognition callback routine instead of an Apple event handler, you must change the value of the `kSRCallBackParam` property of the current recognizer. The value of the `kSRCallBackParam` property is the address of a callback routine parameter structure, defined by the `SRCallBackParam` data type.

```
struct SRCallBackParam {
    SRCallBackUPP                        callBack;
    long                                 refcon;
};
typedef struct SRCallBackParam SRCallBackParam;
```

Field descriptions

`callBack` A routine descriptor for a speech recognition callback routine. You can use the macro `NewSRCallBackProc` (defined in `SpeechRecognition.h`) to create this routine descriptor.

`refcon` An application-defined reference constant. This value is passed to your callback routine in the `refcon` field of a speech recognition callback structure. You can pass any 4-byte value you wish.

Speech Recognition Manager Routines

This section describes the routines provided by the Speech Recognition Manager.

Opening and Closing Recognition Systems

The Speech Recognition Manager provides routines for opening and closing recognition systems. You open a single recognition system when your application starts up and close it before your application exits.

Note
See "Recognition System Properties" on page 1-40 for a description of the available recognition system properties. ◆

SROpenRecognitionSystem

You can use the `SROpenRecognitionSystem` function to open a recognition system.

```
pascal OSErr SROpenRecognitionSystem (
                    SRRecognitionSystem *system,
                    OSType systemID);
```

`system` On exit, a reference to the recognition system having the specified system ID.

`systemID` A recognition system ID. See "Recognition System IDs" on page 1-40 for the available system IDs.

DESCRIPTION

The `SROpenRecognitionSystem` function returns, in the `system` parameter, a reference to the recognition system specified by the `systemID` parameter.

SPECIAL CONSIDERATIONS

You should open a single recognition system when your application starts up and close it (by calling the function `SRCloseRecognitionSystem`) before your application exits.

SRCloseRecognitionSystem

You should use the `SRCloseRecognitionSystem` function to close a recognition system when your application is finished using it (for example, just before your application quits).

```
pascal OSErr SRCloseRecognitionSystem (SRRecognitionSystem system);
```

`system` A recognition system.

DESCRIPTION

The `SRCloseRecognitionSystem` function closes the recognition system specified by the `system` parameter. If any speech objects are still attached to that recognition system, they are disposed of and any references you have to those objects are thereby rendered invalid.

Creating and Manipulating Recognizers

The Speech Recognition Manager provides routines that you can use to create and manage recognizers.

Note
See "Recognizer Properties" on page 1-43 for a description of the available recognizer properties. ◆

SRNewRecognizer

You can use the `SRNewRecognizer` function to create a new recognizer.

```
pascal OSErr SRNewRecognizer (
                    SRRecognitionSystem system,
                    SRRecognizer *recognizer,
                    OSType sourceID);
```

`system` A recognition system.

`recognizer` On exit, a reference to a new recognizer associated with the specified recognition system and using the specified speech source.

`sourceID` A speech source ID.

DESCRIPTION

The `SRNewRecognizer` function returns, in the `recognizer` parameter, a reference to a new recognizer that is associated with the recognition system specified by the `system` parameter and that uses the speech source specified by the `sourceID` parameter. The Speech Recognition Manager supports several speech sources, which you can specify using these constants:

```
enum {
    kSRDefaultSpeechSource                    = 0,
    kSRLiveDesktopSpeechSource                = 'dklv',
    kSRCanned22kHzSpeechSource                = 'ca22'
};
```

In version 1.5, the default speech source is `kSRLiveDesktopSpeechSource`. `SRNewRecognizer` may need to load substantial amounts of data from disk into memory. As a result, you might want to change the cursor to the watch cursor before you call `SRNewRecognizer`.

SPECIAL CONSIDERATIONS

You should call the `SRReleaseObject` function to release the object reference returned by `SRNewRecognizer` when you're done using it.

SRStartListening

You can use the `SRStartListening` function to start a recognizer listening and reporting results to your application.

```
pascal OSErr SRStartListening (SRRecognizer recognizer);
```

`recognizer` A recognizer.

DESCRIPTION

The `SRStartListening` function instructs the recognizer specified by the `recognizer` parameter to begin processing sound from its speech source and reporting its results to your application (either using Apple events or through a speech recognition callback routine).

SPECIAL CONSIDERATIONS

You must already have built a language model and attached it to the recognizer (by calling the `SRSetLanguageModel` function) before you call `SRStartListening`.

SRStopListening

You can use the `SRStopListening` function to stop a recognizer listening and reporting results to your application.

```
pascal OSErr SRStopListening (SRRecognizer recognizer);
```

`recognizer` A recognizer.

DESCRIPTION

The `SRStopListening` function instructs the recognizer specified by the `recognizer` parameter to stop processing sound from its speech source and reporting its results to your application.

SRGetLanguageModel

You can use the `SRGetLanguageModel` function to get a recognizer's active language model.

```
pascal OSErr SRGetLanguageModel (
                    SRRecognizer recognizer,
                    SRLanguageModel *languageModel);
```

`recognizer` A recognizer.

`languageModel` On exit, a reference to the language model currently active for the specified recognizer.

DESCRIPTION

The `SRGetLanguageModel` function returns, in the `languageModel` parameter, a reference to the language model that is currently associated with the recognizer specified by the `recognizer` parameter.

SPECIAL CONSIDERATIONS

`SRGetLanguageModel` increases the reference count of the specified language model. You should call the `SRReleaseObject` function to release the language model reference returned by `SRGetLanguageModel` when you're done using it.

SRSetLanguageModel

You can use the `SRSetLanguageModel` function to set a recognizer's active language model.

```
pascal OSErr SRSetLanguageModel (
                    SRRecognizer recognizer,
                    SRLanguageModel languageModel);
```

`recognizer` A recognizer.

`languageModel` A language model.

DESCRIPTION

The `SRSetLanguageModel` function sets the active language model for the recognizer specified by the `recognizer` parameter to the model specified by the `languageModel` parameter. See "Creating Language Objects," beginning on page 1-66 for routines you can use to build a language model.

If no other references exist to the language model currently in use by the specified recognizer, calling `SRSetLanguageModel` with a different language model causes the current one to be released.

SRContinueRecognition

You can use the `SRContinueRecognition` function to have a recognizer continue recognizing speech.

```
pascal OSErr SRContinueRecognition (SRRecognizer recognizer);
```

`recognizer` A recognizer.

DESCRIPTION

The `SRContinueRecognition` function instructs the recognizer specified by the `recognizer` parameter to continue recognizing speech. You need to call either `SRContinueRecognition` or `SRCancelRecognition` each time your application is notified that the user has started speaking (using Apple events or through an application-defined callback routine). See Listing 1-9 on page 1-28 for information on how to request such a notification.

SRCancelRecognition

You can use the `SRCancelRecognition` function to have a recognizer cancel the attempt to recognize the current utterance.

```
pascal OSErr SRCancelRecognition (SRRecognizer recognizer);
```

`recognizer` A recognizer.

DESCRIPTION

The `SRCancelRecognition` function instructs the recognizer specified by the `recognizer` parameter to stop recognizing speech. You need to call either `SRContinueRecognition` or `SRCancelRecognition` each time your application is notified that the user has started speaking (using Apple events or through an application-defined callback routine). See Listing 1-9 on page 1-28 for information on how to request such a notification.

SRIdle

You can use the `SRIdle` function to grant processing time to the Speech Recognition Manager if your application does not call `WaitNextEvent` frequently.

```
pascal OSErr SRIdle (void);
```

DESCRIPTION

The `SRIdle` function grants processing time to the Speech Recognition Manager, thereby allowing it to process incoming sound and send recognition results.

SPECIAL CONSIDERATIONS

Most applications do not need to call the `SRIdle` function. You need to call it only if your application does a significant amount of processing without periodically calling `WaitNextEvent`. If you do use the `SRIdle` function, you should call it often enough that the Speech Recognition Manager can perform its work.

Note, however, that if you call `SRIdle` and not `WaitNextEvent`, you give time to the recognizer but not to the feedback window. You must call `WaitNextEvent` periodically to have the feedback animations work correctly if your recognizer is using the standard feedback window.

Managing Speech Objects

The Speech Recognition Manager provides routines that operate on any speech object. You can use these routines to get and set the properties of speech objects, to get a reference to a speech object, and to release speech objects.

SRGetProperty

You can use the `SRGetProperty` function to get the current value of a property of a speech object.

```
pascal OSErr SRGetProperty (
                    SRSpeechObject srObject,
                    OSType selector,
                    void *property,
                    Size *propertyLen);
```

`srObject` A speech object.

`selector` A property selector. See “Recognition System Properties” on page 1-40, “Recognizer Properties” on page 1-43, and “Language Object Properties,” beginning on page 1-51 for lists of the available property selectors.

`property` The address of a buffer into which the value of the specified property is to be copied.

`propertyLen` On entry, the address of a variable of type `Size` that contains the length, in bytes, of the specified buffer. On exit, if the buffer is large enough to hold the returned property value and no error occurs, `SRGetProperty` sets `propertyLen` to the total number of bytes in the value of the specified property.

DESCRIPTION

The `SRGetProperty` function returns, through the `property` parameter, the value of the property of the speech object specified by the `srObject` parameter whose type is specified by the `selector` parameter.

The `propertyLen` parameter is a pointer to the length of the buffer into which the property value is to be copied. If the value is of a fixed size, then `propertyLen` should point to a variable of type `Size` that specifies that size. If the size of the value can vary (for example, if the value is a string), then `propertyLen` should point to a variable of type `Size` that specifies the number of bytes in the buffer pointed to by the `property` parameter. If that buffer is not large enough to hold the returned property value, `SRGetProperty` returns the result code `kSRBufferTooSmall` as its function result.

Not all selectors are valid for all types of speech objects. If the selector you specify does not specify a property of the specified speech object, `SRGetProperty` returns the result code `kSRCantGetProperty` or `kSRBadSelector`.

SPECIAL CONSIDERATIONS

If `SRGetProperty` returns an object reference, you must make sure to release that object reference (by calling `SRReleaseObject`) when you are finished using it. Most selectors do not cause `SRGetProperty` to return object references. For example, passing the selector `kSRSpelling` causes `SRGetProperty` to return a buffer of text, not an object reference.

SRSetProperty

You can use the `SRSetProperty` function to set the value of a property of a speech object.

```
pascal OSErr SRSetProperty (
                     SRSpeechObject srObject,
                     OSType selector,
                     const void *property,
                     Size propertyLen);
```

`srObject` A speech object.

`selector` A property selector. See "Recognition System Properties" on page 1-40, "Recognition Result Properties" on page 1-49, "Recognizer Properties" on page 1-43, and "Language Object Properties," beginning on page 1-51 for lists of the available property selectors.

`property` The address of a buffer containing the value to which the specified property is to be set.

`propertyLen` The length, in bytes, of the specified buffer.

DESCRIPTION

The `SRSetProperty` function sets the value of the property of the speech object specified by the `srObject` parameter to the value specified through the `property`

parameter. The `selector` parameter specifies which property is to be set and the `propertyLen` parameter specifies it size, in bytes.

Not all properties can be set. If you attempt to set a property that cannot be set, `SRSetProperty` returns the result code `kSRCantSetProperty` or `kSRBadSelector` as its function result.

SRGetReference

You can use the `SRGetReference` function to obtain an extra reference to a speech object.

```
pascal OSErr SRGetReference (
                        SRSpeechObject srObject,
                        SRSpeechObject *newObjectRef);
```

`srObject` A speech object.

`newObjectRef` On exit, a new reference to the specified speech object.

DESCRIPTION

The `SRGetReference` function returns, in the `newObjectRef` parameter, a new reference to the speech object specified by the `srObject` parameter.

SPECIAL CONSIDERATIONS

`SRGetReference` increases the reference count of the specified speech object. You should call the `SRReleaseObject` function to release the object reference returned by `SRGetReference` when you're done using it.

SRReleaseObject

You can use the `SRReleaseObject` function to release a speech object.

```
pascal OSErr SRReleaseObject (SRSpeechObject srObject);
```

`srObject` A speech object.

DESCRIPTION

The `SRReleaseObject` function releases the object reference specified by the `srObject` parameter. If there are no other remaining references to that object, `SRReleaseObject` disposes of the memory occupied by the object.

SPECIAL CONSIDERATIONS

Your application should balance every function call that returns an object reference with a call to `SRReleaseObject`. This means that every call to a function whose name begins with `SRNew` or `SRGet` that successfully returns an object reference must be balanced with a call to `SRReleaseObject`.

In addition, you should call `SRReleaseObject` to release references to `SRSearchResult` objects that are passed to your application (via an Apple event handler or a callback routine).

SEE ALSO

For more information on creating and releasing object references, see "Object References" on page 1-9.

Creating Language Objects

The Speech Recognition Manager provides routines that you can use to create language objects, which you use to define the recognizable utterances for a recognizer. You can create words, phrases, paths, and language models. See "Manipulating Language Objects" on page 1-70 for routines you can use to configure a language model. Once you've created and configured a language model, you can use the `SRSetLanguageModel` function (described on page 1-60) to set a recognizer's active language model.

Note
See "Language Object Properties" on page 1-51 for a description of the available language model properties. ◆

SRNewWord

You can use the `SRNewWord` function to create a new word.

```
pascal OSErr SRNewWord (
                    SRRecognitionSystem system,
                    SRWord *word,
                    const void *text,
                    Size textLength);
```

`system` A recognition system.

`word` On exit, a reference to a new word associated with the specified recognition system.

`text` The address of a buffer that contains the characters that comprise the word.

`textLength` The size, in bytes, of the specified text.

DESCRIPTION

The `SRNewWord` function returns, in the `word` parameter, a reference to a new word associated with the recognition system specified by the `system` parameter. The word's spelling is specified by the `text` and `textLength` parameters.

SPECIAL CONSIDERATIONS

You should call the `SRReleaseObject` function to release the word reference returned by `SRNewWord` when you're done using it.

SRNewPhrase

You can use the `SRNewPhrase` function to create a new phrase.

```
pascal OSErr SRNewPhrase (
                    SRRecognitionSystem system,
                    SRPhrase *phrase,
                    const void *text,
                    Size textLength);
```

`system` A recognition system.

`phrase` On exit, a reference to a new phrase associated with the specified recognition system.

`text` The address of a buffer that contains the words that comprise the phrase.

`textLength` The size, in bytes, of the specified text.

DESCRIPTION

The `SRNewPhrase` function returns, in the `phrase` parameter, a reference to a new phrase associated with the recognition system specified by the `system` parameter. The phrase's contents (that is, the words that comprise the phrase) is specified by the `text` and `textLength` parameters. You can, if you wish, create a new empty phrase and then add words to it by calling the `SRAddText` or `SRAddLanguageObject` functions.

SPECIAL CONSIDERATIONS

You should call the `SRReleaseObject` function to release the phrase reference returned by `SRNewPhrase` when you're done using it.

SRNewPath

You can use the `SRNewPath` function to create a new path.

```
pascal OSErr SRNewPath (SRRecognitionSystem system, SRPath *path);
```

`system` A recognition system.

`path` On exit, a reference to a new empty path associated with the specified recognition system.

DESCRIPTION

The `SRNewPath` function returns, in the `path` parameter, a reference to a new empty path associated with the recognition system specified by the `system` parameter. You can then add objects to a path by calling the `SRAddText` or `SRAddLanguageObject` functions.

SPECIAL CONSIDERATIONS

You should call the `SRReleaseObject` function to release the path reference returned by `SRNewPath` when you're done using it.

SRNewLanguageModel

You can use the `SRNewLanguageModel` function to create a new language model.

```
pascal OSErr SRNewLanguageModel (
                    SRRecognitionSystem system,
                    SRLanguageModel *model,
                    const void *name,
                    Size nameLength);
```

`system` A recognition system.

`model` On exit, a reference to a new empty language model associated with the specified recognition system.

`name` The address of a buffer that contains the name of the language model. By convention, this name should begin with the character "<" and end with the character ">".

`nameLength` The size, in bytes, of the specified name.

DESCRIPTION

The `SRNewLanguageModel` function returns, in the `model` parameter, a reference to a new language model associated with the recognition system specified by the `system` parameter. The new language model is initially empty and has the name specified by the `name` and `nameLength` parameters. The name of the language model should be unique among all the language models your application creates, and it should be comprehensible to users. (For example, a language model that defined a list of names might be called “<Names>”).

Note

The convention that language model names begin with the character “<” and end with the character “>” is adopted to support future utilities that display the names of language models to the user (perhaps as part of showing the user what he or she can say). ◆

You can add language objects (that is, words, phrases, paths, and other language models) to a language model by calling the `SRAddText` and `SRAddLanguageObject` functions.

SPECIAL CONSIDERATIONS

You should call the `SRReleaseObject` function to release the language model reference returned by `SRNewLanguageModel` when you’re done using it.

SEE ALSO

You can get or set the name of an existing language model by calling the `SRGetProperty` or `SRSetProperty` functions with the `kSRSpelling` property selector.

Manipulating Language Objects

The Speech Recognition Manager provides routines that you can use to manipulate language objects. You can use these routines to alter the contents of a language object. See “Creating Language Objects” on page 1-66 for the routines you can use to create language objects.

SRAddText

You can use the SRAddText function to add text to the contents of a language object.

```
pascal OSErr SRAddText (
                    SRLanguageObject base,
                    const void *text,
                    Size textLength,
                    long refCon);
```

base — A language object.

text — The address of a buffer that contains the words or phrase to add to the contents of the specified language object.

textLength — The size, in bytes, of the specified text.

refCon — An application-defined reference constant. The value of the reference constant property of the new word or phrase representing the specified text is set to this value.

DESCRIPTION

The SRAddText function adds objects representing the text specified by the text and textLength parameters to the contents of the language object specified by the base parameter. In addition, the value of the reference constant property of the added objects is set to the value specified by the refCon parameter.

The SRAddText function is useful for phrases, paths, and language models. If the base parameter specifies a path or language model, SRAddText is equivalent to calling SRNewPhrase, SRAddLanguageObject, and SRReleaseObject for the phrase specified by the text parameter and calling SRSetProperty to set the value of the reference constant property of the new phrase.

If the base parameter specifies a phrase, SRAddText is equivalent to calling SRNewWord, SRAddLanguageObject, and SRReleaseObject for each distinguishable word in the text parameter and calling SRSetProperty to set the value of the reference constant property of the new words.

SPECIAL CONSIDERATIONS

`SRAddLanguageObject` does not alter the value of the reference constant property of the language object specified by the `base` parameter.

SRAddLanguageObject

You can use the `SRAddLanguageObject` function to add a language object to some other language object.

```
pascal OSErr SRAddLanguageObject (
                        SRLanguageObject base,
                        SRLanguageObject addon);
```

`base` A language object.

`addon` A language object.

DESCRIPTION

The `SRAddLanguageObject` function adds the language object specified by the `addon` parameter to the language object specified by the `base` parameter. For example, if `addon` specifies a word and `base` specifies a phrase, then `SRAddLanguageObject` appends that word to the end of that phrase.

The `SRAddLanguageObject` function is useful for adding language objects to phrases, paths, and language models. For a phrase or a path, `SRAddLanguageObject` appends the specified object to the end of the phrase or path. For a language model, `SRAddLanguageObject` adds the specified object to the list of alternative recognizable utterances.

The language object to which you add an object acquires a new reference to the added object. Accordingly, any changes you subsequently make to the added object are reflected in any object to which you added it. The base object releases its reference to the added object when the base object is disposed of.

SEE ALSO

See `SRAddText` (page 1-71) for a useful shortcut function.

SREmptyLanguageObject

You can use the `SREmptyLanguageObject` function to empty the contents of a language object.

```
pascal OSErr SREmptyLanguageObject (SRLanguageObject languageObject);
```

`languageObject`
A language object.

DESCRIPTION

The `SREmptyLanguageObject` function empties the contents of the language object specified by the `languageObject` parameter. (For example, if `languageObject` specifies a phrase containing two words, calling `SREmptyLanguageObject` would result in a phrase that contains no words.) Any properties of that object that are not related to its contents are unchanged. In particular, `SREmptyLanguageObject` does not alter the value of the reference constant property of that language object.

If there are no other references to the words, phrases, paths, and embedded language objects that were contained in the language object, calling `SREmptyLanguageObject` causes them to be disposed of.

SRChangeLanguageObject

You can use the `SRChangeLanguageObject` function to change the contents of a language object.

```
pascal OSErr SRChangeLanguageObject (
                    SRLanguageObject languageObject,
                    const void *text,
                    Size textLength);
```

`languageObject`
A language object.

`text` The address of a buffer that contains the words or phrase to which the contents of the specified language object are to be changed.

`textLength` The size, in bytes, of the specified text.

DESCRIPTION

The `SRChangeLanguageObject` function changes the contents of the language object specified by the `languageObject` parameter to the data specified by the `text` and `textLength` parameters. `SRChangeLanguageObject` is a convenient shortcut for calling `SREmptyLanguageObject` and then `SRAddText`.

`SRChangeLanguageObject` does not alter the value of the reference constant property of the language object specified by the `languageObject` parameter.

If there are no other references to the words, phrases, paths, and embedded language objects that were contained in the language object, calling `SRChangeLanguageObject` causes them to be disposed of.

SPECIAL CONSIDERATIONS

If you want to swap rapidly among several language models, you could use the `SRSetLanguageObject` function instead of `SRChangeLanguageObject`. Or, you could use the `kSREnabled` property to rapidly enable and disable parts of the current language model to reflect the current context.

SRRemoveLanguageObject

You can use the `SRRemoveLanguageObject` function to remove a language object from another language object that contains it.

```
pascal OSErr SRRemoveLanguageObject (
                    SRLanguageObject base,
                    SRLanguageObject toRemove);
```

`base` A language object.

`toRemove` A language object.

DESCRIPTION

The SRRemoveLanguageObject function removes the language object specified by the toRemove parameter from the language object specified by the base parameter. The object specified by the base parameter should be a container one of whose subitems is the object specified by the toRemove parameter.

Calling SRRemoveLanguageObject never disposes of the memory associated with the removed item, because the reference that your application passes in the toRemove parameter continues to be valid after the call (until your application calls SRReleaseObject to release that reference). Note, however, that the base parameter no longer has its own additional reference to the removed object after SRRemoveLanguageObject completes successfully.

Traversing Speech Objects

The Speech Recognition Manager provides routines that you can use to traverse and manipulate speech objects that contain other objects. For instance, you can use these routines to determine how many words are contained in a phrase.

Note
In Speech Recognition Manager version 1.5, these routines are useful only for operating on language objects (of type SRLanguageObject), although they are defined for all speech objects. ◆

SRCountItems

You can use the SRCountItems function to determine the number of subitems in a container object.

```
pascal OSErr SRCountItems (SRSpeechObject container, long *count);
```

container — A speech object.

count — On exit, the number of subitems in the specified speech object.

DESCRIPTION

The `SRCountItems` function returns, in the `count` parameter, the number of subitems contained in the speech object specified by the `container` parameter. This function is useful only for speech objects that have distinguishable subitems, such as phrases (which contain words), paths (which contain words, phrases, and language models), and language models (which contain words, phrases, paths, and possibly other language models).

SRGetIndexedItem

You can use the `SRGetIndexedItem` function to get a subitem in a container object.

```
pascal OSErr SRGetIndexedItem (
                    SRSpeechObject container,
                    SRSpeechObject *item,
                    long index);
```

`container` A speech object.

`item` On exit, a reference to the subitem in the specified speech object that has the specified index.

`index` An integer ranging from 0 to one less than the number of subitems in the specified speech object.

DESCRIPTION

The `SRGetIndexedItem` function returns, in the `item` parameter, a reference to the subitem (in the speech object specified by the `container` parameter) at the index specified by the `index` parameter. This function is useful for iterating through all subitems in a container object (such as each path in a language model or each word in a phrase).

The value passed in the `index` parameter must be greater than or equal to 0 and less than the number of subitems in the specified container object. (You can call the `SRCountItems` function to determine the number of subitems contained in a speech object.) If the index you specify is not in this range, `SRGetIndexedItem` returns the result code `kSRParamOutOfRange`.

SPECIAL CONSIDERATIONS

`SRGetIndexedItem` increases the reference count of the specified speech object. You should call the `SRReleaseObject` function to release the object reference returned by `SRGetIndexedItem` when you're done using it. For example, you can get a reference to the third word in a phrase by executing this code:

```
myErr = SRGetIndexedItem(myPhrase, &myWord, 2);
```

Then, when you are finished using the word, you should execute this code:

```
myErr = SRReleaseObject(myWord);
```

SRSetIndexedItem

You can use the `SRSetIndexedItem` function to replace a subitem in a container object with some other object.

```
pascal OSErr SRSetIndexedItem (
                    SRSpeechObject container,
                    SRSpeechObject item,
                    long index);
```

`container` A speech object.

`item` A speech object.

`index` An integer ranging from 0 to one less than the number of subitems in the specified speech object.

DESCRIPTION

The `SRSetIndexedItem` function replaces the subitem having the index specified by the `index` parameter in the container object specified by the `container` parameter with the speech object specified by the `item` parameter. A reference to the replacement item is maintained separately by the container; as a result, you can release any reference to that item if you no longer need it. The reference to the replaced item is removed from the container; if that reference was the last remaining reference to the object, the object is released.

SRRemoveIndexedItem

You can use the `SRRemoveIndexedItem` function to remove a subitem from a container object.

```
pascal OSErr SRRemoveIndexedItem (SRSpeechObject container, long index);
```

`container` A speech object.

`index` An integer ranging from 0 to one less than the number of subitems in the specified speech object.

DESCRIPTION

The `SRRemoveIndexedItem` function removes from the speech object specified by the `container` parameter the subitem located at the position specified by the `index` parameter. If `SRRemoveIndexedItem` completes successfully, the number of subitems in the container object is reduced by 1, and the index of each subitem that follows the removed item is reduced by 1.

The value passed in the `index` parameter must be greater than or equal to 0 and less than the number of subitems in the specified container object. (You can call the `SRCountItems` function to determine the number of subitems contained in a speech object.) If the index you specify is not in this range, `SRRemoveIndexedItem` returns the result code `kSRParamOutOfRange`.

If there are no other references to the removed item, the memory associated with it is disposed of.

Reading and Writing Language Objects

The Speech Recognition Manager provides routines that you can use to save and load language objects to and from files.

SRPutLanguageObjectIntoHandle

You can use the `SRPutLanguageObjectIntoHandle` function to put a language object (and any embedded languages objects it contains) into a handle.

```
pascal OSErr SRPutLanguageObjectIntoHandle (
                    SRLanguageObject languageObject,
                    Handle lobjHandle);
```

`languageObject`
A language object.

`lobjHandle` A handle to a block of memory into which the data describing the specified language object is to be put. On entry, this handle can have a length of 0.

DESCRIPTION

The `SRPutLanguageObjectIntoHandle` function puts a description of the language object specified by the `languageObject` parameter into the block of memory specified by the `lobjHandle` parameter. This replaces the data in the handle and resizes the handle if necessary.

You can use Resource Manager routines (such as `AddResource`) to store language objects into resources.

SRPutLanguageObjectIntoDataFile

You can use the `SRPutLanguageObjectIntoDataFile` function to put a language object (and any embedded languages objects it contains) into a data file.

```
pascal OSErr SRPutLanguageObjectIntoDataFile (
                    SRLanguageObject languageObject,
                    short fRefNum)
```

`languageObject`
A language object.

`fRefNum` A file reference number of an open data file into which the data describing the specified language object is to be put.

DESCRIPTION

The `SRPutLanguageObjectIntoDataFile` function puts a description of the language object specified by the `languageObject` parameter into the data file specified by the `fRefNum` parameter. Data are written starting at the current file mark, and the file mark is moved to the end of the written data.

SRNewLanguageObjectFromHandle

You can use the `SRNewLanguageObjectFromHandle` function to create a language object from the handle previously created by the `SRPutLanguageObjectIntoHandle` function.

```
pascal OSErr SRNewLanguageObjectFromHandle (
                        SRRecognitionSystem system,
                        SRLanguageObject *languageObject,
                        Handle lobjHandle);
```

`system` A recognition system.

`languageObject`
On exit, a reference to a new language object.

`lobjHandle` A handle to a language object.

DESCRIPTION

The `SRNewLanguageObjectFromHandle` function returns, in the `languageObject` parameter, a reference to a new language object created and initialized using the private data to which the `lobjHandle` parameter is a handle. The data specified by `lobjHandle` should have been created by a previous call to the `SRPutLanguageObjectIntoHandle` function; if that data is not appropriately formatted, `SRNewLanguageObjectFromHandle` returns the result code `kSRCantReadLanguageObject` as its function result.

You can use this routine to load language objects from resources (for example, by using the Resource Manager function `GetResource`).

SPECIAL CONSIDERATIONS

You should call the `SRReleaseObject` function to release the language object reference returned by `SRNewLanguageObjectFromHandle` when you're done using it.

SRNewLanguageObjectFromDataFile

You can use the `SRNewLanguageObjectFromDataFile` function to read a language object from a data file.

```
pascal OSErr SRNewLanguageObjectFromDataFile (
                     SRRecognitionSystem system,
                     SRLanguageObject *languageObject,
                     short fRefNum);
```

`system` A recognition system.

`languageObject`
On exit, a reference to a new language object.

`fRefNum` A file reference number of an open data file.

DESCRIPTION

The `SRNewLanguageObjectFromDataFile` function returns, in the `languageObject` parameter, a reference to a language object whose description is stored in the open data file that has the file reference number specified by the `fRefNum` parameter. `SRNewLanguageObjectFromDataFile` reads data beginning at the current file mark.

If the language object is successfully created and initialized, the file mark is left at the byte immediately following the language object description. Otherwise, if the language object data is not appropriately formatted, `SRNewLanguageObjectFromDataFile` returns the result code `kSRCantReadLanguageObject` as its function result and the file mark is not moved.

SPECIAL CONSIDERATIONS

You should call the `SRReleaseObject` function to release the language object reference returned by `SRNewLanguageObjectFromDataFile` when you're done using it.

Using the System Feedback Window

The Speech Recognition Manager provides routines that you can use to control some aspects of the system feedback window.

SRDrawText

You can use the `SRDrawText` function to draw output text in the feedback window.

```
pascal OSErr SRDrawText (
                    SRRecognizer recognizer,
                    const void *dispText,
                    Size dispLength);
```

`recognizer` A recognizer.

`dispText` The address of a buffer that contains the text to be drawn.

`dispLength` The size, in bytes, of the specified text.

DESCRIPTION

The `SRDrawText` function draws the text specified by the `dispText` and `dispLength` parameters in the transcript portion of the feedback window associated with the recognizer specified by the `recognizer` parameter. The text is drawn in the style characteristic of all output text.

SRDrawRecognizedText

You can use the `SRDrawRecognizedText` function to draw recognized text in the feedback window.

```
pascal OSErr SRDrawRecognizedText (
                        SRRecognizer recognizer,
                        const void *dispText,
                        Size dispLength);
```

`recognizer` A recognizer.

`dispText` The address of a buffer that contains the text to be drawn.

`dispLength` The size, in bytes, of the specified text.

DESCRIPTION

The `SRDrawRecognizedText` function draws the text specified by the `dispText` and `dispLength` parameters in the transcript portion of the feedback window associated with the recognizer specified by the `recognizer` parameter. The text is drawn in the style characteristic of all recognized text. You might want to use this function to display a recognized phrase using a different spelling than the one used in the language model.

SPECIAL CONSIDERATIONS

If the value of the `kSRWantsResultTextDrawn` property of the specified recognizer is `TRUE` (which is the default value), a transcript of the text of a recognition result is automatically sent directly to the feedback window. As a result, you should call `SRDrawRecognizedText` only when the value of the recognizer's `kSRWantsResultTextDrawn` property is `FALSE`.

SRSpeakText

You can use the `SRSpeakText` function to have the feedback character in the feedback window speak a text string.

```
pascal OSErr SRSpeakText (
                    SRRecognizer recognizer,
                    const void *speakText,
                    Size speakLength);
```

`recognizer` A recognizer.

`speakText` The address of a buffer that contains the text to be spoken.

`speakLength` The size, in bytes, of the specified text.

DESCRIPTION

The `SRSpeakText` function causes the feedback character in the feedback window associated with the recognizer specified by the `recognizer` parameter to speak the text specified by the `speakText` and `speakLength` parameters. While speaking, the feedback character lip-synchs the spoken string using the Speech Synthesis Manager's phoneme callback routines. `SRSpeakText` uses the default voice and rate selected in the Speech control panel.

The text pointed to by the `speakText` parameter can contain embedded speech commands to enhance the prosody of the spoken string. See the chapter "Speech Manager" in *Inside Macintosh: Sound* for a complete discussion of embedded speech commands.

Note
The Speech Synthesis Manager was formerly called the Speech Manager. Its name has been changed to distinguish it from the Speech Recognition Manager and to describe its operation more clearly. ◆

SPECIAL CONSIDERATIONS

You can use the `SRSpeechBusy` function to determine whether the feedback character is already speaking. If it is, you can call the `SRStopSpeech` function to stop that speaking immediately.

SEE ALSO

The `SRSpeakText` function speaks the specified text but doesn't display it. Use the `SRSpeakAndDrawText` function if you want to speak and display the text.

SRSpeakAndDrawText

You can use the `SRSpeakAndDrawText` function to draw output text in the feedback window and to have the feedback character in the feedback window speak that text.

```
pascal OSErr SRSpeakAndDrawText (
                    SRRecognizer recognizer,
                    const void *text,
                    Size textLength);
```

`recognizer` A recognizer.

`text` The address of a buffer that contains the text to be drawn and spoken.

`textLength` The size, in bytes, of the specified text.

DESCRIPTION

The `SRSpeakAndDrawText` function draws the text specified by the `text` and `textLength` parameters in the transcript portion of the feedback window associated with the recognizer specified by the `recognizer` parameter. The text is drawn in the style characteristic of all output text. `SRSpeakAndDrawText` also causes the feedback character in the feedback window to speak that text. `SRSpeakAndDrawText` is simply a convenient shortcut for `SRSpeakText` and `SRDrawText`.

SRStopSpeech

You can use the `SRStopSpeech` function to terminate speech by the feedback character in a feedback window.

```
pascal OSErr SRStopSpeech (SRRecognizer recognizer);
```

`recognizer` A recognizer.

DESCRIPTION

The `SRStopSpeech` function immediately terminates any speaking by the feedback character in the feedback window associated with the recognizer specified by the `recognizer` parameter.

SRSpeechBusy

You can use the `SRSpeechBusy` function to determine if the feedback character in a feedback window is currently speaking.

```
pascal Boolean SRSpeechBusy (SRRecognizer recognizer);
```

`recognizer` A recognizer.

DESCRIPTION

The `SRSpeechBusy` function returns, as its function result, the value `TRUE` if the feedback character in the feedback window associated with the recognizer specified by the `recognizer` parameter is currently speaking. Otherwise, `SRSpeechBusy` returns the value `FALSE`.

SRProcessBegin

You can use the `SRProcessBegin` function to indicate that a recognition result is being processed.

```
pascal OSErr SRProcessBegin (SRRecognizer recognizer, Boolean failed);
```

`recognizer` A recognizer.

`failed` A Boolean value that determines how the feedback gestures are to be altered and whether the response sound is to be played (`FALSE`) or not (`TRUE`).

DESCRIPTION

The `SRProcessBegin` function causes the Speech Recognition Manager to provide the relevant feedback (in the feedback window associated with the recognizer specified by the `recognizer` parameter) indicating that the application is in the process of responding to a spoken command. Currently, the gestures of the feedback character are changed to indicate that processing is occurring.

If you set the value of the recognizer's `kSRWantsAutoFBGestures` property to `FALSE`, you should call `SRProcessBegin` at the beginning of your response to a recognition result and `SRProcessEnd` at the end of your response. During the interval separating the two calls, the feedback character displays an appropriate set of gestures showing the user that the task is being processed. If you pass the value `TRUE` in the `failed` parameter (indicating that the recognition result cannot successfully be processed), the feedback character displays frowns, shrugs, or other appropriate gestures. In addition, when `failed` is `TRUE`, you do not need to call `SRProcessEnd` to end the processing. If you pass the value `FALSE` in the `failed` parameter but determine subsequently that the recognition result cannot successfully be processed, you should call `SRProcessEnd` with the `failed` parameter set to `TRUE`.

SPECIAL CONSIDERATIONS

If the value of the `kSRWantsAutoFBGestures` property of the specified recognizer is `TRUE`, the Speech Recognition Manager calls `SRProcessBegin` internally before notifying your application of a recognition result, and it calls `SRProcessEnd` internally after your application is notified. As a result, you should call

SRProcessBegin or SRProcessEnd only when the value of the recognizer's kSRWantsAutoFBGestures property is FALSE.

Because the default value of the kSRWantsAutoFBGestures property is TRUE, most applications don't need to call SRProcessBegin. Calling SRProcessBegin is useful, however, when you know the resulting action might take a significant amount of time.

SRProcessEnd

You can use the SRProcessEnd function to indicate that a recognition result is done being processed.

```
pascal OSErr SRProcessEnd (SRRecognizer recognizer, Boolean failed);
```

recognizer — A recognizer.

failed — A Boolean value that determines how the feedback gestures are to be altered (FALSE) or not (TRUE).

DESCRIPTION

The SRProcessEnd function causes the Speech Recognition Manager to provide the relevant feedback (in the feedback window associated with the recognizer specified by the recognizer parameter) indicating that a recognition result is done being processed. Currently, the gestures of the feedback character are changed and a response sound is played.

If you pass the value FALSE for the failed parameter, the feedback character returns to its idle state, ready for the next utterance. If you pass the value TRUE for the failed parameter, the feedback character responds appropriately (for example, by shrugging briefly before returning to its idle state).

SPECIAL CONSIDERATIONS

If the value of the kSRWantsAutoFBGestures property of the specified recognizer is TRUE (the default value), the Speech Recognition Manager calls SRProcessBegin internally before notifying your application of a recognition result, and it calls SRProcessEnd internally after your application is notified. As a result, you should call SRProcessBegin or SRProcessEnd only if you have

changed the value of the recognizer's `kSRWantsAutoFBGestures` property to `FALSE`.

Because the default value of the `kSRWantsAutoFBGestures` property is `TRUE`, most applications don't need to call `SRProcessBegin` or `SRProcessEnd`. Calling `SRProcessBegin` and `SRProcessEnd` is useful, however, when you know the resulting action might take a significant amount of time.

Application-Defined Routines

This section describes the routines your application or other software component might need to define when using the Speech Recognition Manager.

Speech Recognition Callback Routines

You can receive notification of recognizer events either by installing an Apple event handler or by installing a speech recognition callback routine. In general, you should use an Apple event handler to process recognition notifications. You should use callback routines only for executable code that cannot easily receive Apple events.

MySRCallBack

You can define a speech recognition callback routine to handle recognition notifications.

```
pascal void MySRCallBack (SRCallBackStruct *param);
```

`param` A pointer to a speech recognition callback structure. See "Speech Recognition CallBack Structure" on page 1-53 for a description of this structure.

DESCRIPTION

Your speech recognition callback routine is called whenever the recognizer encounters one of the events specified in its `kSRNotificationParam` property. You can determine what event caused your routine to be called by inspecting

the `what` field of the speech recognition callback structure specified by the `param` parameter.

Because the Speech Recognition Manager is not fully reentrant, you should not call any of its routines other than `SRContinueRecognition` or `SRCancelRecognition` from within your speech recognition callback routine. Accordingly, your callback routine should simply queue the notification for later processing by your software (for instance, when it receives background processing time).

IMPORTANT

If the event is of type `kSRNotifyRecognitionBeginning` (which occurs only if you request recognition-beginning notifications, described on page 1-48), you *must* call either `SRContinueRecognition` or `SRCancelRecognition` before speech recognition can continue. A recognizer that has issued a recognition notification suspends activity until you call one of these two functions. ▲

In general, when your speech recognition callback routine receives the `kSRNotifyRecognitionBeginning` notification, it should queue an indication for your main code both to adjust the current language model (if necessary) and to call the `SRContinueRecognition` function. When your callback routine receives the `kSRNotifyRecognitionDone` notification, it should queue an indication for your main code to handle the recognition result passed in the `message` field of the speech recognition callback structure specified by the `param` parameter. You should make sure, however, that the `message` field contains a valid reference to a recognition result by inspecting the `status` field of that structure; if `status` contains any value other than `noErr`, the contents of the `message` field are undefined.

SPECIAL CONSIDERATIONS

When your callback routine is executed, your application is not the current process. As a result, some restrictions apply; for example, the current resource chain might not be that of your application.

SEE ALSO

See “Using Callback Routines” on page 1-30 for a sample speech recognition callback routine.

Summary of the Speech Recognition Manager

C Summary

Constants

Gestalt Selectors and Response Values

```
enum {
    gestaltSpeechRecognitionVersion             = 'srtb',
    gestaltSpeechRecognitionAttr                = 'srta'
};

enum {
    gestaltDesktopSpeechRecognition             = 1L<<0,
    gestaltTelephoneSpeechRecognition           = 1L<<1
};
```

Recognition System IDs

```
enum {
    kSRDefaultRecognitionSystemID               = 0
};
```

Recognition System Properties

```
enum {
    kSRFeedbackAndListeningModes                = 'fbwn',
    kSRRejectedWord                             = 'rejq',
    kSRCleanupOnClientExit                      = 'clup'
};
```

```
enum {
    kSRNoFeedbackNoListenModes              = 0,
    kSRHasFeedbackHasListenModes            = 1,
    kSRNoFeedbackHasListenModes             = 2
};
```

Speech Source IDs

```
enum {
    kSRDefaultSpeechSource                  = 0,
    kSRLiveDesktopSpeechSource              = 'dklv',
    kSRCanned22kHzSpeechSource              = 'ca22'
};
```

Apple Event Selectors

```
/* Apple event message class */
enum {
    kAESpeechSuite                          = 'sprc'
};

/* Apple event message event IDs */
enum {
    kAESpeechDetected                       = 'srbd',
    kAESpeechDone                           = 'srsd'
};

/* Apple event parameter keywords */
enum {
    keySRRecognizer                         = 'krec',
    keySRSpeechResult                       = 'kspr',
    keySRSpeechStatus                       = 'ksst'
};

/* Apple event parameter types */
enum {
    typeSRRecognizer                        = 'trec',
    typeSRSpeechResult                      = 'tspr'
};
```

Recognizer Properties

```
enum {
    kSRNotificationParam                    = 'noti',
    kSRCallBackParam                        = 'call',
    kSRSearchStatusParam                    = 'stat',
    kSRForegroundOnly                       = 'fgon',
    kSRBlockBackground                      = 'blbg',
    kSRBlockModally                         = 'blmd',
    kSRWantsResultTextDrawn                 = 'txfb',
    kSRWantsAutoFBGestures                  = 'dfbr',
    kSRSoundInVolume                        = 'volu',
    kSRReadAudioFSSpec                      = 'aurd',
    kSRCancelOnSoundOut                     = 'caso',
    kSRListenKeyMode                        = 'lkmd',
    kSRListenKeyCombo                       = 'lkey',
    kSRListenKeyName                        = 'lnam',
    kSRKeyWord                              = 'kwrd',
    kSRKeyExpected                          = 'kexp'
};
```

Search Status Flags

```
enum {
    kSRIdleRecognizer                       = 1L<<0,
    kSRSearchInProgress                     = 1L<<1,
    kSRSearchWaitForAllClients              = 1L<<2,
    kSRMustCancelSearch                     = 1L<<3,
    kSRPendingSearch                        = 1L<<4
};
```

Notification Flags

```
enum {
    kSRNotifyRecognitionBeginning           = 1L<<0,
    kSRNotifyRecognitionDone                = 1L<<1
};
```

Listen Key Modes

```
enum {
    kSRUseToggleListen                              = 0,
    kSRUsePushToTalk                                = 1
};
```

Recognition Result Properties

```
enum {
    kSRLanguageModelFormat                          = 'lmfm',
    kSRPathFormat                                   = 'lmpt',
    kSRPhraseFormat                                 = 'lmph',
    kSRTEXTFormat                                   = 'TEXT'
};
```

Language Object Properties

```
enum {
    kSRSpelling                                     = 'spel',
    kSRLMObjType                                    = 'lmtp',
    kSRRefCon                                       = 'refc',
    kSROptional                                     = 'optl',
    kSREnabled                                      = 'enbl',
    kSRRepeatable                                   = 'rptb',
    kSRRejectable                                   = 'rjbl',
    kSRRejectionLevel                               = 'rjct'
};
```

Language Object Types

```
enum {
    kSRLanguageModelType                            = 'lmob',
    kSRPathType                                     = 'path',
    kSRPhraseType                                   = 'phra',
    kSRWordType                                     = 'word'
};
```

Procedure Information

```
enum {
    uppSRCallBackProcInfo                          = kPascalStackBased |
            STACK_ROUTINE_PARAMETER(1, SIZE_CODE(sizeof(SRCallBackStruct*)))
};
```

Default Rejection Level

```
enum {
    kSRDefaultRejectionLevel                       = 50
};
```

Data Types

Speech Objects

```
typedef struct OpaqueSRSpeechObject                *SRSpeechObject;

typedef SRSpeechObject                             SRRecognitionSystem;
typedef SRSpeechObject                             SRRecognizer;
typedef SRSpeechObject                             SRSpeechSource;
typedef SRSpeechObject                             SRLanguageObject;

typedef SRSpeechSource                             SRRecognitionResult;
```

Language Objects

```
typedef SRLanguageObject                           SRLanguageModel;
typedef SRLanguageObject                           SRPath;
typedef SRLanguageObject                           SRPhrase;
typedef SRLanguageObject                           SRWord;
```

CallBack Structure

```
struct SRCallBackStruct {
    long                                what;
    long                                message;
    SRRecognizer                        instance;
    OSErr                               status;
```

```
    short                                   flags;
    long                                    refcon;
};
typedef struct SRCallBackStruct SRCallBackStruct;
```

Callback Routine Type

```
#if GENERATINGCFM
typedef UniversalProcPtr                    SRCallBackUPP;
#else
typedef SRCallBackProcPtr                   SRCallBackUPP;
#endif
```

Callback Routine Parameter Structure

```
struct SRCallBackParam {
    SRCallBackUPP                           callBack;
    long                                    refcon;
};
typedef struct SRCallBackParam SRCallBackParam;
```

Other Types

```
typedef unsigned short                      SRRejectionLevel;
```

Speech Recognition Manager Routines

Opening and Closing Recognition Systems

```
pascal OSErr SROpenRecognitionSystem (
                                    SRRecognitionSystem *system,
                                    OSType systemID);

pascal OSErr SRCloseRecognitionSystem (
                                    SRRecognitionSystem system);
```

Creating and Manipulating Recognizers

```
pascal OSErr SRNewRecognizer      (SRRecognitionSystem system,
                                   SRRecognizer *recognizer,
                                   OSType sourceID);
pascal OSErr SRStartListening     (SRRecognizer recognizer);
pascal OSErr SRStopListening      (SRRecognizer recognizer);
pascal OSErr SRGetLanguageModel   (SRRecognizer recognizer,
                                   SRLanguageModel *languageModel);
pascal OSErr SRSetLanguageModel   (SRRecognizer recognizer,
                                   SRLanguageModel languageModel);
pascal OSErr SRContinueRecognition(SRRecognizer recognizer);
pascal OSErr SRCancelRecognition  (SRRecognizer recognizer);
pascal OSErr SRIdle               (void);
```

Managing Speech Objects

```
pascal OSErr SRGetProperty        (SRSpeechObject srObject,
                                   OSType selector,
                                   void *property,
                                   Size *propertyLen);
pascal OSErr SRSetProperty        (SRSpeechObject srObject,
                                   OSType selector,
                                   const void *property,
                                   Size propertyLen);
pascal OSErr SRGetReference       (SRSpeechObject srObject,
                                   SRSpeechObject *newObjectRef);
pascal OSErr SRReleaseObject      (SRSpeechObject srObject);
```

Creating Language Objects

```
pascal OSErr SRNewWord            (SRRecognitionSystem system,
                                   SRWord *word,
                                   const void *text,
                                   Size textLength);
```

```
pascal OSErr SRNewPhrase          (SRRecognitionSystem system,
                                   SRPhrase *phrase,
                                   const void *text,
                                   Size textLength);

pascal OSErr SRNewPath            (SRRecognitionSystem system, SRPath *path);

pascal OSErr SRNewLanguageModel   (SRRecognitionSystem system,
                                   SRLanguageModel *model,
                                   const void *name,
                                   Size nameLength);
```

Manipulating Language Objects

```
pascal OSErr SRAddText            (SRLanguageObject base,
                                   const void *text,
                                   Size textLength,
                                   long refCon);

pascal OSErr SRAddLanguageObject  (SRLanguageObject base, SRLanguageObject addon);

pascal OSErr SREmptyLanguageObject(SRLanguageObject languageObject);

pascal OSErr SRChangeLanguageObject (
                                   SRLanguageObject languageObject,
                                   const void *text,
                                   Size textLength);

pascal OSErr SRRemoveLanguageObject (
                                   SRLanguageObject base, SRLanguageObject toRemove);
```

Traversing Speech Objects

```
pascal OSErr SRCountItems         (SRSpeechObject container, long *count);

pascal OSErr SRGetIndexedItem     (SRSpeechObject container,
                                   SRSpeechObject *item,
                                   long index);

pascal OSErr SRSetIndexedItem     (SRSpeechObject container,
                                   SRSpeechObject item,
                                   long index);

pascal OSErr SRRemoveIndexedItem  (SRSpeechObject container, long index);
```

Reading and Writing Language Objects

```
pascal OSErr SRPutLanguageObjectIntoHandle (
                                  SRLanguageObject languageObject, Handle lobjHandle);

pascal OSErr SRPutLanguageObjectIntoDataFile (
                                  SRLanguageObject languageObject, short fRefNum);

pascal OSErr SRNewLanguageObjectFromHandle (
                                  SRRecognitionSystem system,
                                  SRLanguageObject *languageObject,
                                  Handle lobjHandle);

pascal OSErr SRNewLanguageObjectFromDataFile (
                                  SRRecognitionSystem system,
                                  SRLanguageObject *languageObject,
                                  short fRefNum);
```

Using the System Feedback Window

```
pascal OSErr SRDrawText           (SRRecognizer recognizer,
                                   const void *dispText,
                                   Size dispLength);

pascal OSErr SRDrawRecognizedText (SRRecognizer recognizer,
                                   const void *dispText,
                                   Size dispLength);

pascal OSErr SRSpeakText          (SRRecognizer recognizer,
                                   const void *speakText,
                                   Size speakLength);

pascal OSErr SRSpeakAndDrawText   (SRRecognizer recognizer,
                                   const void *text,
                                   Size textLength);

pascal OSErr SRStopSpeech         (SRRecognizer recognizer);

pascal Boolean SRSpeechBusy       (SRRecognizer recognizer);

pascal OSErr SRProcessBegin       (SRRecognizer recognizer, Boolean failed);

pascal OSErr SRProcessEnd         (SRRecognizer recognizer, Boolean failed);
```

Application-Defined Routines

Callback Routines

```
pascal void MySRCallBack          (SRCallBackStruct *param);
```

Pascal Summary

Constants

Gestalt Selectors and Response Values

```
CONST
    gestaltSpeechRecognitionVersion                 = 'srtb';
    gestaltSpeechRecognitionAttr                    = 'srta';

    gestaltDesktopSpeechRecognition                 = $00000001;
    gestaltTelephoneSpeechRecognition               = $00000002;
```

Recognition System IDs

```
    kSRDefaultRecognitionSystemID                   = 0;
```

Recognition System Properties

```
    kSRFeedbackAndListeningModes                    = 'fbwn';
    kSRRejectedWord                                 = 'rejq';
    kSRCleanupOnClientExit                          = 'clup';

    kSRNoFeedbackNoListenModes                      = 0;
    kSRHasFeedbackHasListenModes                    = 1;
    kSRNoFeedbackHasListenModes                     = 2;
```

Speech Source IDs

```
kSRDefaultSpeechSource              = 0;
kSRLiveDesktopSpeechSource          = 'dklv';
kSRCanned22kHzSpeechSource          = 'ca22';
```

Apple Event Selectors

```
{Apple event message class}
kAESpeechSuite                      = 'sprc';

{Apple event message event IDs}
kAESpeechDetected                   = 'srbd';
kAESpeechDone                       = 'srsd';

{Apple event parameter keywords}
keySRRecognizer                     = 'krec';
keySRSpeechResult                   = 'kspr';
keySRSpeechStatus                   = 'ksst';

{Apple event parameter types}
typeSRRecognizer                    = 'trec';
typeSRSpeechResult                  = 'tspr';
```

Recognizer Properties

```
kSRNotificationParam                = 'noti';
kSRCallBackParam                    = 'call';
kSRSearchStatusParam                = 'stat';
kSRForegroundOnly                   = 'fgon';
kSRBlockBackground                  = 'blbg';
kSRBlockModally                     = 'blmd';
kSRWantsResultTextDrawn             = 'txfb';
kSRWantsAutoFBGestures              = 'dfbr';
kSRSoundInVolume                    = 'volu';
kSRReadAudioFSSpec                  = 'aurd';
kSRCancelOnSoundOut                 = 'caso';
kSRListenKeyMode                    = 'lkmd';
kSRListenKeyCombo                   = 'lkey';
kSRListenKeyName                    = 'lnam';
kSRKeyWord                          = 'kwrd';
kSRKeyExpected                      = 'kexp';
```

Search Status Flags

```
kSRIdleRecognizer                    = $00000001;
kSRSearchInProgress                  = $00000002;
kSRSearchWaitForAllClients           = $00000004;
kSRMustCancelSearch                  = $00000008;
kSRPendingSearch                     = $00000010;
```

Notification Flags

```
kSRNotifyRecognitionBeginning        = $00000001;
kSRNotifyRecognitionDone             = $00000002;
```

Recognition Result Properties

```
kSRLanguageModelFormat               = 'lmfm';
kSRPathFormat                        = 'lmpt';
kSRPhraseFormat                      = 'lmph';
kSRTEXTFormat                        = 'TEXT';
```

Language Object Properties

```
kSRSpelling                          = 'spel';
kSRLMObjType                         = 'lmtp';
kSRRefCon                            = 'refc';
kSROptional                          = 'optl';
kSREnabled                           = 'enbl';
kSRRepeatable                        = 'rptb';
kSRRejectable                        = 'rjbl';
kSRRejectionLevel                    = 'rjct';
```

Language Object Types

```
kSRLanguageModelType                 = 'lmob';
kSRPathType                          = 'path';
kSRPhraseType                        = 'phra';
kSRWordType                          = 'word';
```

Default Rejection Level

```
kSRDefaultRejectionLevel             = 50;
```

Data Types

Speech Objects

```
TYPE
    SRSpeechObject          = ^LongInt;

    SRRecognitionSystem     = SRSpeechObject;
    SRRecognizer            = SRSpeechObject;
    SRSpeechSource          = SRSpeechObject;
    SRLanguageObject        = SRSpeechObject;

    SRRecognitionResult     = SRSpeechSource;
```

Language Objects

```
    SRLanguageModel         = SRLanguageObject;
    SRPath                  = SRLanguageObject;
    SRPhrase                = SRLanguageObject;
    SRWord                  = SRLanguageObject;
```

CallBack Structure

```
    SRCallBackStruct =
    RECORD
        what:                           LongInt;
        message:                        LongInt;
        instance:                       SRRecognizer;
        status:                         OSErr;
        flags:                          Integer;
        refcon:                         LongInt;
    END;
    SRCallBackStructPtr = ^SRCallBackStruct;
    SRCallBackStructHandle = ^SRCallBackStructPtr;
```

Callback Routine Type

```
    SRCallBackProcPtr = ProcPtr;
    SRCallBackUPP = SRCallBackProcPtr;
```

Callback Routine Parameter Structure

```
SRCallBackParam =
RECORD
    callBack:                   SRCallBackUPP;
    refcon:                     LongInt;
END;
SRCallBackParamPtr = ^SRCallBackParam;
SRCallBackParamHandle = ^SRCallBackParamPtr;
```

Other Types

```
SRRejectionLevel        = Integer;
```

Speech Recognition Manager Routines

Opening and Closing Recognition Systems

```
FUNCTION SROpenRecognitionSystem  (VAR system: SRRecognitionSystem;
                                   systemID: OSType): OSErr;

FUNCTION SRCloseRecognitionSystem (system: SRRecognitionSystem): OSErr;
```

Creating and Manipulating Recognizers

```
FUNCTION SRNewRecognizer          (system: SRRecognitionSystem;
                                   VAR recognizer: SRRecognizer;
                                   sourceID: OSType): OSErr;

FUNCTION SRStartListening         (recognizer: SRRecognizer): OSErr;

FUNCTION SRStopListening          (recognizer: SRRecognizer): OSErr;

FUNCTION SRGetLanguageModel       (recognizer: SRRecognizer;
                                   VAR languageModel: SRLanguageModel): OSErr;

FUNCTION SRSetLanguageModel       (recognizer: SRRecognizer;
                                   languageModel: SRLanguageModel): OSErr;

FUNCTION SRContinueRecognition    (recognizer: SRRecognizer): OSErr;

FUNCTION SRCancelRecognition      (recognizer: SRRecognizer): OSErr;

FUNCTION SRIdle                   : OSErr;
```

Managing Speech Objects

```
FUNCTION SRGetProperty          (srObject: SRSpeechObject;
                                 selector: OSType;
                                 property: UNIV Ptr;
                                 VAR propertyLen: Size): OSErr;

FUNCTION SRSetProperty          (srObject: SRSpeechObject;
                                 selector: OSType;
                                 property: UNIV Ptr;
                                 propertyLen: Size): OSErr;

FUNCTION SRGetReference         (srObject: SRSpeechObject;
                                 VAR newObjectRef: SRSpeechObject): OSErr;

FUNCTION SRReleaseObject        (srObject: SRSpeechObject): OSErr;
```

Creating Language Objects

```
FUNCTION SRNewWord              (system: SRRecognitionSystem;
                                 VAR word: SRWord;
                                 text: UNIV Ptr;
                                 textLength: Size): OSErr;

FUNCTION SRNewPhrase            (system: SRRecognitionSystem;
                                 VAR phrase: SRPhrase;
                                 text: UNIV Ptr;
                                 textLength: Size): OSErr;

FUNCTION SRNewPath              (system: SRRecognitionSystem; VAR path: SRPath):
                                 OSErr;

FUNCTION SRNewLanguageModel     (system: SRRecognitionSystem;
                                 VAR model: SRLanguageModel;
                                 name: UNIV Ptr;
                                 nameLength: Size): OSErr;
```

Manipulating Language Objects

```
FUNCTION SRAddText              (base: SRLanguageObject;
                                 text: UNIV Ptr;
                                 textLength: Size;
                                 refCon: LongInt): OSErr;

FUNCTION SRAddLanguageObject    (base: SRLanguageObject; addon: SRLanguageObject):
                                 OSErr;
```

```
FUNCTION SREmptyLanguageObject    (languageObject: SRLanguageObject): OSErr;

FUNCTION SRChangeLanguageObject   (languageObject: SRLanguageObject;
                                   text: UNIV Ptr;
                                   textLength: Size): OSErr;

FUNCTION SRRemoveLanguageObject   (base: SRLanguageObject;
                                   toRemove: SRLanguageObject): OSErr;
```

Traversing Speech Objects

```
FUNCTION SRCountItems             (container: SRSpeechObject; VAR count: LongInt):
                                   OSErr;

FUNCTION SRGetIndexedItem         (container: SRSpeechObject;
                                   VAR item: SRSpeechObject;
                                   index: LongInt): OSErr;

FUNCTION SRSetIndexedItem         (container: SRSpeechObject;
                                   item: SRSpeechObject;
                                   index: LongInt): OSErr;

FUNCTION SRRemoveIndexedItem      (container: SRSpeechObject; index: LongInt): OSErr;
```

Reading and Writing Language Objects

```
FUNCTION SRPutLanguageObjectIntoHandle (
                                   languageObject: SRLanguageObject;
                                   lobjHandle: Handle): OSErr;

FUNCTION SRPutLanguageObjectIntoDataFile (
                                   languageObject: SRLanguageObject;
                                   fRefNum: Integer): OSErr;

FUNCTION SRNewLanguageObjectFromHandle (
                                   system: SRRecognitionSystem;
                                   VAR languageObject: SRLanguageObject;
                                   lobjHandle: Handle): OSErr;

FUNCTION SRNewLanguageObjectFromDataFile (
                                   system: SRRecognitionSystem;
                                   VAR languageObject: SRLanguageObject;
                                   fRefNum: Integer): OSErr;
```

Using the System Feedback Window

```
FUNCTION SRDrawText               (recognizer: SRRecognizer;
                                   dispText: UNIV Ptr;
                                   dispLength; Size): OSErr;

FUNCTION SRDrawRecognizedText     (recognizer: SRRecognizer;
                                   dispText: UNIV Ptr;
                                   dispLength; Size): OSErr;

FUNCTION SRSpeakText              (recognizer: SRRecognizer;
                                   speakText: UNIV Ptr;
                                   speakLength: Size): OSErr;

FUNCTION SRSpeakAndDrawText       (recognizer: SRRecognizer;
                                   text: UNIV Ptr;
                                   textLength: Size): OSErr;

FUNCTION SRStopSpeech             (recognizer: SRRecognizer): OSErr;

FUNCTION SRSpeechBusy             (recognizer: SRRecognizer): Boolean;

FUNCTION SRProcessBegin           (recognizer: SRRecognizer; failed: Boolean): OSErr;

FUNCTION SRProcessEnd             (recognizer: SRRecognizer; failed: Boolean): OSErr;
```

Application-Defined Routines

Callback Routines

```
PROCEDURE MyCallBackProc          (param: SRCallBackStructPtr);
```

Result Codes

kSRNotAvailable	–5100	Requested service not available or applicable
kSRInternalError	–5101	Internal system or hardware error condition
kSRComponentNotFound	–5102	Required component cannot be located
kSROutOfMemory	–5103	Not enough memory available
kSRNotASpeechObject	–5104	Object is not valid
kSRBadParameter	–5105	Invalid parameter specified
kSRParamOutOfRange	–5106	Parameter is out of valid range
kSRBadSelector	–5107	Unrecognized selector specified
kSRBufferTooSmall	–5108	Buffer is too small
kSRNotARecSystem	–5109	Specified object is not a recognition system
kSRFeedbackNotAvail	–5110	No feedback window associated with recognizer
kSRCantSetProperty	–5111	Cannot set the specified property
kSRCantGetProperty	–5112	Cannot get the specified property
kSRCantSetDuringRecognition	–5113	Cannot set property during recognition
kSRAlreadyListening	–5114	System is already listening
kSRNotListeningState	–5115	System is not listening
kSRModelMismatch	–5116	No acoustical models available to match request
kSRNoClientLanguageModel	–5117	Cannot access specified language model
kSRNoPendingUtterances	–5118	No utterances to search
kSRRecognitionCanceled	–5119	Search was canceled
kSRRecognitionDone	–5120	Search has finished, but nothing was recognized
kSROtherRecAlreadyModal	–5121	Another recognizer is already operating modally
kSRHasNoSubItems	–5122	Specified object has no subitems
kSRSubItemNotFound	–5123	Specified subitem cannot be located
kSRLanguageModelTooBig	–5124	Language model too big to be built
kSRAlreadyReleased	–5125	Specified object has already been released
kSRAlreadyFinished	–5126	Specified language model has already been finished
kSRWordNotFound	–5127	Spelling could not be found
kSRNotFinishedWithRejection	–5128	Language model not finished with rejection
kSRExpansionTooDeep	–5129	Language model is left recursive or is embedded too many levels
kSRTooManyElements	–5130	Too many elements added to phrase, path, or other language object
kSRCantAdd	–5131	Can't add specified type of object to the base language object

`kSRSndInSourceDisconnected`	–5132	Sound input source is disconnected
`kSRCantReadLanguageObject`	–5133	Cannot create language object from file or pointer
`kSRNotImplementedYet`	–5199	Feature is not yet implemented

Glossary

active language model The language model that is currently associated with a recognizer. You can call the `SRSetLanguageModel` function to set the active language model.

Apple event A high-level event that adheres to the Apple Event Interprocess Messaging Protocol. An Apple event consists of attributes (including the event class and event ID, which identify the event and its task) and, usually, parameters (which contain data used by the target application for the event).

Apple event handler An application-defined function that extracts pertinent data from an Apple event, performs the action requested by the Apple event, and returns a result.

Apple Event Manager The collection of routines that allows client applications to send Apple events to server applications for the purpose of requesting services or information.

Backus-Naur Form (BNF) A standard form for representing formal grammars or other language models.

BNF See **Backus-Naur Form.**

callback routine See **speech recognition callback routine.**

callback routine parameter structure A data structure that contains information about a speech recognition callback routine. A callback routine parameter structure is defined by the `SRCallBackParam` data type.

cleanup mode A property of a recognition system that determines whether the recognition system and all other objects it has created are disposed of when your code exits.

container See **container object.**

container object Any speech object that contains distinguishable subitems. For example, a phrase may contain distinguishable words.

embedded language model A language model that occurs in the path or set of paths that define some other language model. Compare **top-level language model.**

embedded speech command In a buffer of input text, a sequence of characters enclosed by command delimiters that provides instructions to a speech synthesizer.

feedback and listening modes A property of a recognition system that determines whether a feedback window is displayed for any recognizer associated with that system and whether the recognizer uses the listening modes selected by the user in the Speech control panel.

feedback character The picture (often of a head) displayed in a feedback window.

feedback services A feature of the Speech Recognition Manager that provides a standard set of cues and responses during speech recognition and the ability for your application to alter those cues and responses.

feedback window A floating window that provides audiovisual cues and responses to a PlainTalk user. A feedback window is divided into a status pane and a transcript pane. See also **feedback character, output text, recognized text.**

key word The word that must precede utterances when the recognizer is in toggle-listen mode.

key word mode See **toggle-listen mode.**

language model A language object that represents a list of zero or more words, phrases, or paths. A language model is defined by the `SRLanguageModel` data type. See also **embedded language model, top-level language model.**

language object Any speech object that belongs to a subclass of the `SRLanguageObject` class. See also **language model, path, phrase, word.**

listening mode indicator A word or phrase that indicates which keyword must be uttered or which key must be held down in order for a recognizer to start listening. The current listening mode indicator appears at the bottom of the status pane in a feedback window.

listen key The key (or key combination) that either must be held down in order for a recognizer to start listening or that, when pressed, toggles listening on and off.

listen key mode A recognizer property that determines whether the listen key operates in push-to-talk or toggle-listening mode.

notification See **recognition notification.**

object See **speech object.**

object reference See **reference.**

output text Any text drawn into the feedback window that represents a response to a recognized utterance. Compare **recognized text.**

path A language object that represents a sequence of zero or more words, phrases, or embedded language models. A path is defined by the `SRPath` data type.

phrase A language object that represents a sequence of zero or more words. A phrase is defined by the `SRPhrase` data type.

PlainTalk A collection of operating system managers, control panels, and other software that enables Macintosh computers to speak written text and to respond to spoken commands. See also **Speech Recognition Manager, Speech Synthesis Manager.**

property An item of data associated with an object. Properties control some of the object's behavior.

property selector A 4-byte value passed to `SRGetProperty` or `SRSetProperty` that specifies which property of an object to get or set.

property type The type of a property. A property type is specified by a property selector.

property value The value of a property.

push-to-talk mode A listening mode that requires a key to be held down in order for a recognizer to start listening. Compare **toggle-listening mode.**

recognition notification A message sent to your application by a recognizer when certain events occur in the recognizer.

recognition result A speech object that describes a recognized utterance. A recognition result is defined by the `SRRecognitionResult` data type.

recognition system A speech object that defines certain global characteristics of the speech recognition process. A recognition system is defined by the `SRRecognitionSystem` data type.

recognition system ID A value passed to the `SROpenRecognitionSystem` function that specifies a recognition system.

recognized speech The product of converting speech into digitally-stored words or phonemes or into computer actions. See also **Speech Recognition Manager, synthesized speech.**

recognized text Any text drawn into the feedback window that represents a recognized utterance of the user. Compare **output text.**

recognizer A speech object that recognizes utterances and sends recognition results to your application. A recognizer is defined by the `SRRecognizer` data type.

reference A 4-byte value associated with a speech object.

reference constant A 4-byte value defined by your application that can be associated with a language object as the value of its `kSRRefCon` property.

reference count The number of references an application has to a speech object.

reject To deem an utterance unrecognizable.

rejectable Said of an object if a recognition system can return the value `kSRRejectedWord` instead of that object.

rejected word A word that is passed to your application in a recognition result whenever a recognizer hears some sound but cannot recognize it.

separation character A character used to separate the words in a phrase. By default, a recognition system's separation character is the character " ".

speech The process or product of speaking.

speech class A structure for the data that characterize speech objects, together with a set of properties for those objects. Compare **speech object.**

speech class hierarchy The hierarchical arrangement of speech object classes.

speech command See **embedded speech command.**

Speech Manager See **Speech Synthesis Manager.**

speech object Any instance of a speech class.

speech recognition The process of listening to and interpreting spoken utterances. The Speech Recognition Manager provides speech recognition services for Macintosh computers.

speech recognition callback routine An application-defined routine called by the Speech Recognition Manager when certain recognizer events occur (for instance, when the recognizer has recognized an utterance).

speech recognition callback structure A data structure that contains information about a recognition result. A speech recognition callback structure is defined by the `SRCallBackStruct` data type.

Speech Recognition Manager The part of the Macintosh system software that provides a standardized method for Macintosh applications to recognize speech. Compare **Speech Synthesis Manager.**

Speech Synthesis Manager The part of the Macintosh system software that provides a standardized method for Macintosh applications to generate synthesized speech. Previously called the Speech Manager. Compare **Speech Recognition Manager.**

status pane The portion of a feedback window that provides information about the status of a recognizer. The status pane includes the feedback character and the listening mode indicator. Compare **transcript pane.**

subitem An item in a container.

synthesized speech The product of converting nonaural tokens (such as written or digitally-stored words or phonemes) into speech. See also **recognized speech, Speech Synthesis Manager.**

text-to-speech See **synthesized speech.**

toggle-listening mode A listening mode that interprets presses on the listen key as a toggle to turn listening on or off. Compare **push-to-talk mode.**

top-level language model A language model that does not occur in the path or set of paths that define any other language model. Compare **embedded language model.**

transcript pane The portion of a feedback window that contains a readable transcript of the few most recent recognized utterances and feedback. Compare **status pane.** See also **output text, recognized text.**

value See **property value.**

voice recognition See **speech recognition.**

word A language object that represents a single spoken word. A word is defined by the `SRWord` data type. The properties of a word include its spelling, its pronunciation, and its reference constant.

Index

Symbols

< 1-12
> 1-12
| 1-12

A

active language model 1-7
Apple event handlers
  introduced 1-7
  processing notifications with 1-28 to 1-30
Apple event selectors 1-42 to 1-43
audio file property 1-45
automatic feedback gestures flag 1-45

B

background-blocking property 1-44
Backus-Naur Form 1-12
BNF. *See* Backus-Naur Form

C

callback property 1-44
callback routine parameter structures 1-30, 1-55
callback routines. *See* speech recognition callback routines
cancel during sound output flag 1-46
cleanup mode 1-41
container objects
  counting items in 1-75
  getting an indexed item in 1-76
  removing an indexed item from 1-78
  setting an indexed item in 1-77
containers. *See* container objects

D

data files
  getting language objects from 1-38
  putting language objects into 1-37

E

embedded language models 1-13
embedded speech commands 1-84
enabled flag, of language objects 1-52

F

feedback and listening modes 1-16, 1-40 to 1-41
feedback characters 1-15
  determining if speaking 1-86
  terminating speech by 1-86
feedback services
  defined 1-13 to 1-17
  routines for 1-82 to 1-89
feedback window 1-14
foreground-only property 1-44

G

`Gestalt` function
  and Speech Recognition Manager 1-17 to 1-18, 1-39

H

handles
  getting language objects from 1-38
  putting language objects into 1-37

K

key expected flag 1-47
key word mode. *See* toggle-listen mode
key word property 1-46

L

language models
  See also language objects
  active 1-7
  building 1-21 to 1-25
  counting subitems in 1-75
  creating 1-69
  defined 1-11 to 1-13
  embedded 1-13
  introduced 1-6
  top-level 1-13
language objects
  See also speech objects
  adding language objects to 1-72
  adding text to 1-71
  changing 1-73
  emptying 1-73
  getting from a data file 1-38, 1-81
  getting from a handle 1-38, 1-80
  getting from a resource 1-38
  introduced 1-12
  properties of 1-51 to 1-53
  putting into a data file 1-37, 1-79
  putting into a handle 1-37, 1-79
  putting into a resource 1-37
  removing language objects from 1-74
  routines for 1-66 to 1-82
  spelling of 1-51
  types of 1-51, 1-53
listening mode indicators 1-15
listen key combination property 1-46
listen key mode 1-46
listen key modes 1-49
listen key name property 1-46

M

microphone speech source 1-39
modal-blocking flag 1-44
`MySRCallBack` callback routine 1-89

N

notification flags 1-48
notification property 1-27, 1-43
notifications. *See* recognition notifications

O

object references. *See* references
objects. *See* speech objects
optional flag, of languyage objects 1-52
optional language objects 1-25
output text
  drawing 1-82
  introduced 1-15
  speaking and drawing 1-85

P

paths
  See also language objects
  counting subitems in 1-75
  creating 1-68
  introduced 1-12
phrases

See also language objects
counting words in 1-75
creating 1-68
introduced 1-12
PlainTalk 1-5
properties
introduced 1-10 to 1-11
routines for manipulating 1-63 to 1-65
property selectors
for language objects 1-51 to 1-53
for recognition results 1-49 to 1-50
for recognition systems 1-40 to 1-41
for recognizers 1-43 to 1-47
introduced 1-11
property types 1-10
property values
getting 1-63
introduced 1-10
setting 1-64
push-to-talk mode 1-15

R

recognition notifications
constants for 1-48
handling 1-27 to 1-33, 1-89 to 1-90
introduced 1-7
recognition results
introduced 1-7
properties of 1-49 to 1-50
recognition system IDs 1-40
recognition systems
introduced 1-6
opening 1-18 to 1-19, 1-56
properties of 1-40 to 1-41
routines for 1-56 to 1-57
recognized text
drawing 1-83
introduced 1-15
recognizer properties 1-43 to 1-47
recognizers
canceling recognition 1-61
continuing recognition 1-61
creating 1-20, 1-57
getting language model 1-59
granting time to 1-62
introduced 1-7
routines for 1-57 to 1-62
setting language model 1-25, 1-60
starting listening 1-27, 1-58
stopping feedback speech 1-86
stopping listening 1-27, 1-59
reference constants 1-11, 1-51
reference counts 1-10
references 1-9 to 1-10
rejectable 1-52
rejectable flag, of language objects 1-52
rejected words 1-41
rejection levels 1-52
repeatable flag, of language objects 1-52
resources
getting language objects from 1-38
resources, putting language objects into 1-37
result codes 1-108

S

sample routines
`MyBuildLanguageModel` 1-21
`MyCallPersonInPath` 1-36
`MyHandleSpeechDetected` 1-30
`MyHandleSpeechDone` 1-29
`MyHasSpeechRecognitionMgr` 1-18
`MyIdleCheckForSpeechResult` 1-33
`MyInitSpeechRecognition` 1-19
`MyInstallNotificationCallBack` 1-31
`MyNotificationCallBack` 1-32
`MyProcessRecognitionResult` 1-35
`MyRemoveNotificationCallBack` 1-31
`MySetRejectedWordRefCon` 1-26
`MyTerminateSpeechRecognition` 1-20
search status 1-44
search status flags 1-47 to 1-48
sound input volume 1-45
speech classes 1-8
speech class hierarchy 1-8

speech commands. *See* embedded speech commands
Speech control panel 1-15
Speech Manager. *See* Speech Synthesis Manager
speech objects
  getting new references to 1-65
  getting properties of 1-63
  introduced 1-8 to 1-9
  releasing 1-65
  routines for 1-62 to 1-78
  setting properties of 1-64
speech recognition 1-5
speech recognition callback routines 1-7, 1-30 to 1-33, 1-89 to 1-90
speech recognition callback structures 1-53 to 1-55
Speech Recognition Manager 1-5 to 1-109
  application-defined routines in 1-89 to 1-90
  checking for features of 1-17
  constants for 1-39 to 1-53
  data structures for 1-53 to 1-55
  defined 1-5
  limitations of 1-7
  result codes 1-108
  routines in 1-55 to 1-89
  sample code for 1-17 to 1-33
speech source IDs 1-20
Speech Synthesis Manager 1-5, 1-6, 1-84
SRAddLMObject function 1-72
SRAddText function 1-71
SRCallBackParam data type 1-55
SRCallBackStruct data type 1-53 to 1-55
SRCancelRecognition function 1-28, 1-61
SRChangeLMObject function 1-73
SRCloseRecognitionSystem function 1-57
SRContinueRecognition function 1-28, 1-61
SRCountItems function 1-75
SRDrawRecognizedText function 1-83
SRDrawText function 1-82
SREmptyLMObject function 1-73
SRGetIndexedItem function 1-76
SRGetLanguageModel function 1-59
SRGetProperty function 1-63
SRGetReference function 1-65
SRIdle function 1-62
SRLanguageObject data type 1-8
SRLoadLMObject function 1-81
SRNewLanguageModel function 1-69
SRNewLanguageObjectFromHandle function 1-80
SRNewPath function 1-68
SRNewPhrase function 1-68
SRNewRecognizer function 1-57
SRNewWord function 1-67
SROpenRecognitionSystem function 1-56
SRProcessBegin function 1-87
SRProcessEnd function 1-88
SRPutLanguageObjectIntoDataFile function 1-79
SRPutLanguageObjectIntoHandle function 1-79
SRRecognitionResult data type 1-8
SRRecognitionSystem data type 1-8, 1-19
SRRecognizer data type 1-8
SRReleaseObject function 1-9, 1-65
SRRemoveIndexedItem function 1-78
SRRemoveLMObject function 1-74
SRSetIndexedItem function 1-77
SRSetLanguageModel function 1-60
SRSetProperty function 1-64
SRSpeakAndDrawText function 1-85
SRSpeakText function 1-84
SRSpeechBusy function 1-86
SRSpeechObject data type 1-8, 1-9
SRSpeechSource data type 1-8
SRStartListening function 1-27, 1-58
SRStopListening function 1-27, 1-59
SRStopSpeech function 1-86
status pane 1-15
subitems
  counting 1-75
  getting by index 1-76
  removing by index 1-78
  setting by index 1-77

T

telephone speech source 1-39
text, speaking 1-84
text feedback flag 1-45

text-to-speech. *See* synthesized speech
top-level language models 1-13
transcript pane 1-15

V

values. *See* property values
voice recognition. *See* speech recognition

W

`WaitNextEvent` function, and `SRIdle` 1-62
words
  See also language objects
  counting in a phrase 1-75
  creating 1-67
  introduced 1-12

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages were created on an Apple LaserWriter IINTX printer. Final page negatives were output directly from text files on an Agfa Large-Format Imagesetter. Line art was created using Adobe™ Illustrator and Adobe Photoshop. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER
Tim Monroe

ILLUSTRATOR
Sandee Karr

PROJECT MANAGER
Patricia Eastman

Special thanks to Eric "Braz" Ford, Matt Pallakoff, Arlo Reeves, and Brent Schorsch.

Acknowledgments to Tavares Ford.