Speech Synthesis Manager Reference

User Experience > Speech Technologies

2009-04-08

Contents

Speech Synthesis Manager Reference

Framework:	ApplicationServices/ApplicationServices.h
Declared in	SpeechSynthesis.h

Overview

The Speech Synthesis Manager, formerly called the Speech Manager, is the part of the Mac OS that provides a standardized method for Macintosh applications to generate synthesized speech. For example, you may want your application to incorporate the capability to speak its dialog box messages to the user. A word-processing application might use the Speech Synthesis Manager to implement a command that speaks a selected section of a document to the user. Because sound samples can take up large amounts of room on disk, using text in place of sampled sound is extremely efficient, and so a multimedia application might use the Speech Synthesis Manager to provide a narration of a QuickTime movie instead of including sampled-sound data on a movie track.

Mac OS X v10.5 introduces native support for performing speech synthesis tasks using Core Foundation-based objects, such as speaking text represented as `CFString` objects and managing speech channel properties using a `CFDictionary`-based property dictionary. You should begin using the new, Core Foundation-based programming interfaces as soon as it's convenient, because future synthesizers will accept Core Foundation strings and data structures directly through the speech synthesis framework. In the meantime, existing buffer-based clients and synthesizers will continue to work as before, with strings and other data structures getting automatically converted as necessary.

Functions by Task

Changing Speech Attributes

`SetSpeechInfo` (page 28)
    Changes a setting of a particular speech channel.

`SetSpeechProperty` (page 29)
    Sets the value of the specified speech-channel property.

`SetSpeechPitch` (page 29)
    Sets the speech pitch on a designated speech channel.

`SetSpeechRate` (page 30)
    Sets the speech rate of a designated speech channel.

Converting Text To Phonemes

`TextToPhonemes` (page 37)
: Converts a buffer of textual data into phonemic data.

`CopyPhonemesFromText` (page 11)
: Converts the specified text string into its equivalent phonemic representation.

Installing a Pronunciation Dictionary

`UseDictionary` (page 38)
: Installs the designated dictionary into a speech channel.

`UseSpeechDictionary` (page 39)
: Registers a speech dictionary with a speech channel.

Managing Speech Channels

`DisposeSpeechChannel` (page 13)
: Disposes of an existing speech channel.

`NewSpeechChannel` (page 24)
: Creates a new speech channel.

Obtaining Information About Speech and Speech Channels

`CopySpeechProperty` (page 12)
: Gets the value associated with the specified property of a speech channel.

`GetSpeechInfo` (page 16)
: Gets information about a designated speech channel.

`GetSpeechPitch` (page 17)
: Gets a speech channel's current speech pitch.

`GetSpeechRate` (page 18)
: Gets a speech channel's current speech rate.

`SpeechBusy` (page 34)
: Determines whether any channels of speech are currently synthesizing speech.

`SpeechBusySystemWide` (page 34)
: Determines if any speech is currently being synthesized in your application or elsewhere on the computer.

`SpeechManagerVersion` (page 35)
: Determines the current version of the Speech Synthesis Manager installed in the system.

Getting Information About Voices

`CountVoices` (page 12)
: Determines how many voices are available.

`GetIndVoice` (page 16)
: Gets a voice specification structure for a voice by passing an index to the `GetIndVoice` function.

`GetVoiceDescription` (page 18)
: Gets a description of a voice by using the `GetVoiceDescription` function.

`GetVoiceInfo` (page 19)
: Gets the same information about a voice that the `GetVoiceDescription` function provides, or to determine in which file and resource a voice is stored.

`MakeVoiceSpec` (page 23)
: Sets the fields of a voice specification structure.

Starting, Stopping, and Pausing Speech

`ContinueSpeech` (page 10)
: Resumes speech paused by the `PauseSpeechAt` function.

`PauseSpeechAt` (page 27)
: Pauses speech on a speech channel.

`SpeakBuffer` (page 30)
: Speaks a buffer of text, using certain flags to control speech behavior.

`SpeakString` (page 32)
: Begins speaking a text string.

`SpeakCFString` (page 31)
: Begins speaking a string represented as a `CFString` object.

`SpeakText` (page 33)
: Begins speaking a buffer of text.

`StopSpeech` (page 35)
: Terminates speech immediately on the specified channel.

`StopSpeechAt` (page 36)
: Terminates speech delivery on a specified channel either immediately or at the end of the current word or sentence.

Creating, Invoking, and Disposing Universal Procedure Pointers

`DisposeSpeechDoneUPP` (page 13)
: Disposes of a universal procedure pointer (UPP) to a speech-done callback function.

`DisposeSpeechErrorUPP` (page 14)
: Disposes of a universal procedure pointer (UPP) to an error callback function.

`DisposeSpeechPhonemeUPP` (page 14)
: Disposes of a universal procedure pointer (UPP) to a phoneme callback function.

`DisposeSpeechSyncUPP` (page 15)
: Disposes of a universal procedure pointer (UPP) to a synchronization callback function.

`DisposeSpeechTextDoneUPP` (page 15)
: Disposes of a universal procedure pointer (UPP) to a text-done callback function.

`DisposeSpeechWordUPP` (page 15)
: Disposes of a universal procedure pointer (UPP) to a word callback function.

`InvokeSpeechDoneUPP` (page 20)
: Invokes your speech-done callback function.

`InvokeSpeechErrorUPP` (page 21)
: Invokes your error callback function.

`InvokeSpeechPhonemeUPP` (page 21)
: Invokes your phoneme callback function.

`InvokeSpeechSyncUPP` (page 22)
: Invokes your synchronization callback function.

`InvokeSpeechTextDoneUPP` (page 22)
: Invokes your text-done callback function.

`InvokeSpeechWordUPP` (page 23)
: Invokes your word callback function.

`NewSpeechDoneUPP` (page 24)
: Creates a new universal procedure pointer (UPP) to a speech-done callback function.

`NewSpeechErrorUPP` (page 25)
: Creates a new universal procedure pointer to an error callback function.

`NewSpeechPhonemeUPP` (page 25)
: Disposes of a universal procedure pointer (UPP) to a phoneme callback function.

`NewSpeechSyncUPP` (page 26)
: Creates a new universal procedure pointer (UPP) to a synchronization callback function.

`NewSpeechTextDoneUPP` (page 26)
: Creates a new universal procedure pointer (UPP) to a text-done callback function.

`NewSpeechWordUPP` (page 27)
: Creates a new universal procedure pointer (UPP) to a word callback function.

Functions

ContinueSpeech

Resumes speech paused by the `PauseSpeechAt` function.

```
OSErr ContinueSpeech (
   SpeechChannel chan
);
```

Parameters

`chan`
: The paused speech channel on which speech is to be resumed.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
At any time after the `PauseSpeechAt` function is called, the `ContinueSpeech` function can be called to continue speaking from the beginning of the word in which speech paused. Calling `ContinueSpeech` on a channel that is not currently in a paused state has no effect on the speech channel or on future calls to the `PauseSpeechAt` function. If you call `ContinueSpeech` on a channel before a pause is effective, `ContinueSpeech` cancels the pause.

If the `PauseSpeechAt` function stopped speech in the middle of a word, the Speech Synthesis Manager will start speaking that word from the beginning when you call `ContinueSpeech`.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

CopyPhonemesFromText

Converts the specified text string into its equivalent phonemic representation.

```
OSErr CopyPhonemesFromText (
   SpeechChannel chan,
   CFStringRef text,
   CFStringRef * phonemes
);
```

Parameters

chan
: A speech channel whose associated synthesizer and properties are to be used in the conversion process.

text
: The text from which to extract phonemic data.

phonemes
: On return, a `CFString` object that contains the extracted phonemic data. The caller is responsible for releasing this object.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
The `CopyPhonemesFromText` function is the Core Foundation-based equivalent of the `TextToPhonemes` (page 37) function.

Converting textual data into phonemic data is particularly useful during application development, when you might wish to adjust phrases that your application generates to produce smoother speech. By first converting the target phrase into phonemes, you can see what the synthesizer will try to speak. Then you need correct only the parts that would not have been spoken the way you want.

The data the `CopyPhonemesFromText` function stores in the `phonemes` parameter corresponds precisely to the phonemes that would be spoken had the input text been sent to `SpeakCFString` instead. All current property settings for the speech channel specified by `chan` are applied to the converted speech. No callbacks are generated while the `CopyPhonemesFromText` function is generating its output.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

CopySpeechProperty

Gets the value associated with the specified property of a speech channel.

```
OSErr CopySpeechProperty (
   SpeechChannel chan,
   CFStringRef property,
   CFTypeRef * object
);
```

Parameters

chan
The speech channel with which the specified property is associated.

property
A speech-channel property about which information is being requested. See "`Speech-Channel Properties`" (page 70) for information on the properties you can specify.

object
On return, a pointer to a Core Foundation object that holds the value of the specified property. The type of the object depends on the specific property passed in. For some properties, the value of *object* can be `NULL`. When the returned object is a `CFDictionary` object, you can use `CFDictionary` functions, such as `CFDictionaryGetValue`, to retrieve the values associated with the keys that are associated with the specified property.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
The `CopySpeechProperty` function is the Core Foundation-based equivalent of the `GetSpeechInfo` (page 16) function.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

CountVoices

Determines how many voices are available.

```
OSErr CountVoices (
   SInt16 *numVoices
);
```

Parameters

numVoices
On exit, a pointer to the number of voices that the application can use.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `CountVoices` function returns, in the `numVoices` parameter, the number of voices available. The application can then use this information to call the `GetIndVoice` function to obtain voice specification structures for one or more of the voices.

Each time `CountVoices` is called, the Speech Synthesis Manager searches for new voices.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechChannel

Disposes of an existing speech channel.

```
OSErr DisposeSpeechChannel (
   SpeechChannel chan
);
```

Parameters

`chan`

The speech channel to dispose of.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `DisposeSpeechChannel` function disposes of the speech channel specified in the `chan` parameter and releases all memory the channel occupies. If the speech channel specified is producing speech, then the `DisposeSpeechChannel` function immediately stops speech before disposing of the channel. If you have defined a text-done callback function or a speech-done callback function, the function will not be called before the channel is disposed of.

The Speech Synthesis Manager releases any speech channels that have not been explicitly disposed of by an application when the application quits. In general, however, your application should dispose of any speech channels it has created whenever it receives a suspend event. This ensures that other applications can take full advantage of Speech Synthesis Manager and Sound Manager capabilities.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechDoneUPP

Disposes of a universal procedure pointer (UPP) to a speech-done callback function.

```
void DisposeSpeechDoneUPP (
   SpeechDoneUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechErrorUPP

Disposes of a universal procedure pointer (UPP) to an error callback function.

```
void DisposeSpeechErrorUPP (
   SpeechErrorUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechPhonemeUPP

Disposes of a universal procedure pointer (UPP) to a phoneme callback function.

```
void DisposeSpeechPhonemeUPP (
   SpeechPhonemeUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechSyncUPP

Disposes of a universal procedure pointer (UPP) to a synchronization callback function.

```
void DisposeSpeechSyncUPP (
   SpeechSyncUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechTextDoneUPP

Disposes of a universal procedure pointer (UPP) to a text-done callback function.

```
void DisposeSpeechTextDoneUPP (
   SpeechTextDoneUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechWordUPP

Disposes of a universal procedure pointer (UPP) to a word callback function.

```
void DisposeSpeechWordUPP (
   SpeechWordUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

GetIndVoice

Gets a voice specification structure for a voice by passing an index to the `GetIndVoice` function.

```
OSErr GetIndVoice (
   SInt16 index,
   VoiceSpec *voice
);
```

Parameters

`index`
The index of the voice for which to obtain a voice specification structure. This number must range from `1` to the total number of voices, as returned by the `CountVoices` function.

`voice`
A pointer to the voice specification structure whose fields are to be filled in.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
The `GetIndVoice` function returns, in the voice specification structure pointed to by the `voice` parameter, a specification of the voice whose index is provided in the `index` parameter. Your application should make no assumptions about the order in which voices are indexed.

Your application should not add, remove, or modify a voice and then call the `GetIndVoice` function with an index value other than `1`. To allow the Speech Synthesis Manager to update its information about voices, your application should always either call the `CountVoices` function or call the `GetIndVoice` function with an index value of `1` after adding, removing, or modifying a voice or after a time at which the user might have done so.

If you specify an index value beyond the number of available voices, the `GetIndVoice` function returns a `voiceNotFound` error.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

GetSpeechInfo

Gets information about a designated speech channel.

```
OSErr GetSpeechInfo (
   SpeechChannel chan,
   OSType selector,
   void *speechInfo
);
```

Parameters

`chan`

The speech channel about which information is being requested.

`selector`

A speech information selector that indicates the type of information being requested.

For a complete list of speech information selectors, see "Speech-Channel Information Constants" (page 63). This list indicates how your application should set the `speechInfo` parameter for each selector.

`speechInfo`

A pointer whose meaning depends on the speech information selector specified in the `selector` parameter.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `GetSpeechInfo` function returns, in the data structure pointed to by the `speechInfo` parameter, the type of information requested by the `selector` parameter as it applies to the speech channel specified in the `chan` parameter.

The format of the data structure specified by the `speechInfo` parameter depends on the selector you choose. For example, a selector might require that your application allocate a block of memory of a certain size and pass a pointer to that block. Another selector might require that `speechInfo` be set to the address of a handle variable. In this case, the `GetSpeechInfo` function would allocate a relocatable block of memory and change the handle variable specified to reference the block.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

GetSpeechPitch

Gets a speech channel's current speech pitch.

```
OSErr GetSpeechPitch (
   SpeechChannel chan,
   Fixed *pitch
);
```

Parameters

`chan`

The speech channel whose pitch you wish to determine.

`pitch`

On return, a pointer to the current pitch of the voice in the speech channel, expressed as a fixed-point frequency value.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
Typical voice frequencies range from around 90 hertz for a low-pitched male voice to perhaps 300 hertz for a high-pitched child's voice. These frequencies correspond to approximate pitch values in the ranges of 30.000 to 40.000 and 55.000 to 65.000, respectively.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

GetSpeechRate

Gets a speech channel's current speech rate.

```
OSErr GetSpeechRate (
   SpeechChannel chan,
   Fixed *rate
);
```

Parameters

`chan`
The speech channel whose rate you wish to determine.

`rate`
On return, a pointer to the speech channel's speech rate in words per minute, expressed as an integer value.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

GetVoiceDescription

Gets a description of a voice by using the `GetVoiceDescription` function.

```
OSErr GetVoiceDescription (
   const VoiceSpec *voice,
   VoiceDescription *info,
   long infoLength
);
```

Parameters

voice

A pointer to the voice specification structure identifying the voice to be described, or `NULL` to obtain a description of the system default voice.

info

A pointer to a voice description structure. If this parameter is `NULL`, the function does not fill in the fields of the voice description structure; instead, it simply determines whether the `voice` parameter specifies an available voice and, if not, returns a `voiceNotFound` error.

infoLength

The length, in bytes, of the voice description structure. In the current version of the Speech Synthesis Manager, the voice description structure contains 362 bytes. However, you should always use the `SizeOf` function to determine the length of this structure.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `GetVoiceDescription` function fills out the voice description structure pointed to by the `info` parameter with the correct information for the voice specified by the `voice` parameter. It fills in the `length` field of the voice description structure with the number of bytes actually copied. This value will always be less than or equal to the value that your application passes in `infoLength` before calling `GetVoiceDescription`. This scheme allows applications targeted for the current version of the Speech Synthesis Manager to work on future versions that might have longer voice description structures; it also allows you to write code for future versions of the Speech Synthesis Manager that will also run on computers that support only the current version.

If the voice specification structure does not identify an available voice, `GetVoiceDescription` returns a `voiceNotFound` error.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

GetVoiceInfo

Gets the same information about a voice that the `GetVoiceDescription` function provides, or to determine in which file and resource a voice is stored.

```
OSErr GetVoiceInfo (
   const VoiceSpec *voice,
   OSType selector,
   void *voiceInfo
);
```

Parameters

voice
A pointer to the voice specification structure identifying the voice about which your application requires information, or `NULL` to obtain information on the system default voice.

selector
A specification of the type of data being requested. For current versions of the Speech Synthesis Manager, you should set this field either to `soVoiceDescription`, if you would like to use the `GetVoiceInfo` function to mimic the `GetVoiceDescription` function, or to `soVoiceFile`, if you would like to obtain information about the location of a voice on disk.

voiceInfo
A pointer to the appropriate data structure. If the selector is `soVoiceDescription`, then `voiceInfo` should be a pointer to a voice description structure, and the `length` field of the structure should be set to the length of the voice description structure. If the selector is `soVoiceFile`, then `voiceInfo` should be a pointer to a voice file information structure.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
This function is intended primarily for use by synthesizers, but an application can call it too.

The `GetVoiceInfo` function accepts a selector in the `selector` parameter that determines the type of information you wish to obtain about the voice specified in the `voice` parameter. The function then fills the fields of the data structure appropriate to the selector you specify in the `voiceInfo` parameter.

If the voice specification is invalid, `GetVoiceInfo` returns a `voiceNotFound` error. If there is not enough memory to load the voice into memory to obtain information about it, `GetVoiceInfo` returns the result code `memFullErr`.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechDoneUPP

Invokes your speech-done callback function.

```
void InvokeSpeechDoneUPP (
   SpeechChannel chan,
   SRefCon refCon,
   SpeechDoneUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechDoneUPP` function, because the system calls your speech-done callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechErrorUPP

Invokes your error callback function.

```
void InvokeSpeechErrorUPP (
   SpeechChannel chan,
   SRefCon refCon,
   OSErr theError,
   long bytePos,
   SpeechErrorUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechErrorUPP` function, because the system calls your error callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechPhonemeUPP

Invokes your phoneme callback function.

```
void InvokeSpeechPhonemeUPP (
   SpeechChannel chan,
   SRefCon refCon,
   SInt16 phonemeOpcode,
   SpeechPhonemeUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechPhonemeUPP` function, because the system calls your phoneme callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechSyncUPP

Invokes your synchronization callback function.

```
void InvokeSpeechSyncUPP (
   SpeechChannel chan,
   SRefCon refCon,
   OSType syncMessage,
   SpeechSyncUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechSyncUPP` function, because the system calls your synchronization callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechTextDoneUPP

Invokes your text-done callback function.

```
void InvokeSpeechTextDoneUPP (
   SpeechChannel chan,
   SRefCon refCon,
   const void **nextBuf,
   unsigned long *byteLen,
   SInt32 *controlFlags,
   SpeechTextDoneUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechTextDoneUPP` function, because the system calls your text-done callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechWordUPP

Invokes your word callback function.

```
void InvokeSpeechWordUPP (
   SpeechChannel chan,
   SRefCon refCon,
   unsigned long wordPos,
   UInt16 wordLen,
   SpeechWordUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechWordUPP` function, because the system calls your word callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

MakeVoiceSpec

Sets the fields of a voice specification structure.

```
OSErr MakeVoiceSpec (
   OSType creator,
   OSType id,
   VoiceSpec *voice
);
```

Parameters

`creator`
: The ID of the synthesizer that your application requires.

`id`
: The ID of the voice on the synthesizer specified by the `creator` parameter.

`voice`
: A pointer to the voice specification structure whose fields are to be filled in.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
A voice specification structure is a unique voice ID used by the Speech Synthesis Manager. Most voice management functions expect to be passed a pointer to a voice specification structure. When you already know the creator and ID for a voice, you should use the `MakeVoiceSpec` function to create such a structure rather than filling in the fields of one directly. On exit, the voice specification structure pointed to by the `voice` parameter contains the appropriate values. You should never set the fields of such a structure directly.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

NewSpeechChannel

Creates a new speech channel.

```
OSErr NewSpeechChannel (
   VoiceSpec *voice,
   SpeechChannel *chan
);
```

Parameters

`voice`

A pointer to the voice specification structure corresponding to the voice to be used for the new speech channel. Pass `NULL` to create a speech channel using the system default voice. Specifying a voice means the initial speaking rate is determined by the synthesizer's default speaking rate; passing `NULL` means the speaking rate is automatically set to the rate the user specifies in Speech preferences.

`chan`

On return, a pointer to a valid speech channel.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
The `NewSpeechChannel` function allocates memory for a speech channel structure and sets the speech channel variable pointed to by the `chan` parameter to point to this speech channel structure. The Speech Synthesis Manager automatically locates and opens a connection to the proper synthesizer for the voice specified by the `voice` parameter.

There is no predefined limit to the number of speech channels an application can create. However, system constraints on available RAM, processor loading, and number of available sound channels limit the number of speech channels actually possible.

Your application should not attempt to manipulate the data pointed to by a variable of type `SpeechChannel`. The internal format that the Speech Synthesis Manager uses for speech channel data is not documented and may change in future versions of system software.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

NewSpeechDoneUPP

Creates a new universal procedure pointer (UPP) to a speech-done callback function.

```
SpeechDoneUPP NewSpeechDoneUPP (
   SpeechDoneProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your speech-done callback function.

Return Value

A UPP to the speech-done callback function. See the description of the `SpeechDoneUPP` data type.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

NewSpeechErrorUPP

Creates a new universal procedure pointer to an error callback function.

```
SpeechErrorUPP NewSpeechErrorUPP (
   SpeechErrorProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your error callback function.

Return Value

A UPP to the error callback function. See the description of the `SpeechErrorUPP` data type.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

NewSpeechPhonemeUPP

Disposes of a universal procedure pointer (UPP) to a phoneme callback function.

```
SpeechPhonemeUPP NewSpeechPhonemeUPP (
   SpeechPhonemeProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your phoneme callback function.

Return Value

A UPP to the phoneme callback function. See the description of the `SpeechPhonemeUPP` data type.

Availability
Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

NewSpeechSyncUPP

Creates a new universal procedure pointer (UPP) to a synchronization callback function.

```
SpeechSyncUPP NewSpeechSyncUPP (
   SpeechSyncProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your synchronization callback function.

Return Value
A UPP to the synchronization callback function. See the description of the `SpeechSyncUPP` data type.

Availability
Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

NewSpeechTextDoneUPP

Creates a new universal procedure pointer (UPP) to a text-done callback function.

```
SpeechTextDoneUPP NewSpeechTextDoneUPP (
   SpeechTextDoneProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your text-done callback function.

Return Value
A UPP to the text-done callback function. See the description of the `SpeechTextDoneUPP` data type.

Availability
Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

NewSpeechWordUPP

Creates a new universal procedure pointer (UPP) to a word callback function.

```
SpeechWordUPP NewSpeechWordUPP (
   SpeechWordProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your word callback function.

Return Value

A UPP to the word callback function. See the description of the `SpeechWordUPP` data type.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

PauseSpeechAt

Pauses speech on a speech channel.

```
OSErr PauseSpeechAt (
   SpeechChannel chan,
   SInt32 whereToPause
);
```

Parameters

chan

The speech channel on which speech is to be paused.

whereToPause

A constant indicating when speech processing should be paused. Pass the constant `kImmediate` to pause immediately, even in the middle of a word. Pass `kEndOfWord` or `kEndOfSentence` to pause speech at the end of the current word or sentence, respectively.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `PauseSpeechAt` function makes speech production pause at a specified point in the text. `PauseSpeechAt` returns immediately, although speech output will continue until the specified point.

You can determine whether your application has paused speech output on a speech channel by obtaining a speech status information structure through the `GetSpeechInfo` function. While a speech channel is paused, the speech status information structure indicates that `outputBusy` and `outputPaused` are both `TRUE`.

If the end of the input text buffer is reached before the specified pause point, speech output pauses at the end of the buffer.

The `PauseSpeechAt` function differs from the `StopSpeech` and `StopSpeechAt` functions in that a subsequent call to `ContinueSpeech`, described next, causes the contents of the current text buffer to continue being spoken.

If you plan to continue speech synthesis from a paused speech channel, the text buffer being processed must remain available at all times and must not move while the channel is in a paused state.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SetSpeechInfo

Changes a setting of a particular speech channel.

```
OSErr SetSpeechInfo (
   SpeechChannel chan,
   OSType selector,
   const void *speechInfo
);
```

Parameters

`chan`
The speech channel for which your application wishes to change a setting.

`selector`
A speech information selector that indicates the type of information being changed.

For a complete list of speech information selectors, see "Speech-Channel Information Constants" (page 63). This list indicates how your application should set the `speechInfo` parameter for each selector.

`speechInfo`
A pointer whose meaning depends on the speech information selector specified in the `selector` parameter.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
The `SetSpeechInfo` function changes the type of setting indicated by the `selector` parameter in the speech channel specified by the `chan` parameter, based on the data your application provides via the `speechInfo` parameter.

The format of the data structure specified by the `speechInfo` parameter depends on the selector you choose. Ordinarily, a selector requires that `speechInfo` be a pointer to a data structure that specifies a new setting for the speech channel.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SetSpeechPitch

Sets the speech pitch on a designated speech channel.

```
OSErr SetSpeechPitch (
   SpeechChannel chan,
   Fixed pitch
);
```

Parameters

chan

The speech channel whose pitch you wish to set.

pitch

The new pitch for the speech channel, expressed as a fixed-point frequency value.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `SetSpeechPitch` function changes the current speech pitch on the speech channel specified by the `chan` parameter to the pitch specified by the `pitch` parameter. Typical voice frequencies range from around 90 hertz for a low-pitched male voice to perhaps 300 hertz for a high-pitched child's voice. These frequencies correspond to approximate pitch values in the ranges of 30.000 to 40.000 and 55.000 to 65.000, respectively. Although fixed-point values allow you to specify a wide range of pitches, not all synthesizers will support the full range of pitches. If your application specifies a pitch that a synthesizer cannot handle, it may adjust the pitch to fit within an acceptable range.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

SetSpeechProperty

Sets the value of the specified speech-channel property.

```
OSErr SetSpeechProperty (
   SpeechChannel chan,
   CFStringRef property,
   CFTypeRef object
);
```

Parameters

chan

The speech channel whose property to set.

property

The speech-channel property to set to the specified value.

object

The value to which the specified speech-channel property should be set. For some properties, this value can be `NULL`.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `SetSpeechProperty` function is the Core Foundation-based equivalent of the `SetSpeechInfo` (page 28) function.

See "`Speech-Channel Properties`" (page 70) for information on the properties you can specify.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`SpeechSynthesis.h`

SetSpeechRate

Sets the speech rate of a designated speech channel.

```
OSErr SetSpeechRate (
   SpeechChannel chan,
   Fixed rate
);
```

Parameters

`chan`

The speech channel whose rate you wish to set.

`rate`

The new speech rate in words per minute, expressed as an integer value.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `SetSpeechRate` function adjusts the speech rate on the speech channel specified by the `chan` parameter to the rate specified by the `rate` parameter. As a general rule, speaking rates range from around 150 words per minute to around 220 words per minute. It is important to keep in mind, however, that users will differ greatly in their ability to understand synthesized speech at a particular rate based upon their level of experience listening to the voice and their ability to anticipate the types of utterances they will encounter.

Note: the new speech rate should be expressed as an integer (not a fixed point decimal number as the data type implies).

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeakBuffer

Speaks a buffer of text, using certain flags to control speech behavior.

```
OSErr SpeakBuffer (
   SpeechChannel chan,
   const void *textBuf,
   unsigned long textBytes,
   SInt32 controlFlags
);
```

Parameters

chan

The speech channel through which speech is to be spoken.

textBuf

A pointer to the first byte of text to spoken.

textBytes

The number of bytes of text to spoken.

controlFlags

Control flags to customize speech behavior.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `SpeakBuffer` function behaves identically to the `SpeakText` function, but allows control of several speech parameters by setting values of the `controlFlags` parameter. The `controlFlags` parameter relies on specific constants, which may be applied additively. See "Control Flags Constants" (page 58).

Each constant specifies a flag bit of the `controlFlags` parameter, so by passing the constants additively you can enable multiple capabilities of `SpeakBuffer`. If you pass `0` in the `controlFlags` parameter, `SpeakBuffer` works just like `SpeakText`. By passing `kNoEndingProsody + kNoSpeechInterrupt` in the `controlFlags` parameter, `SpeakBuffer` works like `SpeakText` except that the `kNoEndingProsody` and `kNoSpeechInterrupt` features have been selected. Future versions of the Speech Synthesis Manager may define additional constants.

When the `controlFlags` parameter is set to `0`, `SpeakBuffer` behaves identically to `SpeakText`.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeakCFString

Begins speaking a string represented as a `CFString` object.

```
OSErr SpeakCFString (
   SpeechChannel chan,
   CFStringRef aString,
   CFDictionaryRef options
);
```

Parameters

chan

The speech channel through which speech is to be spoken.

aString

The string to be spoken, represented as a `CFString` object.

options

An optional dictionary of key-value pairs used to customize speech behavior. See "`Synthesizer Option Keys`" (page 76) for the available keys.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `SpeakCFString` function is the Core Foundation-based equivalent of the `SpeakBuffer` (page 30) function.

The `SpeakCFString` function converts the text string specified in *aString* into speech, using the voice and control settings in effect for the speech channel specified in *chan*. (Before you use `SpeakCFString`, therefore, be sure you've created a speech channel with the `NewSpeechChannel` (page 24) function.) The `SpeakCFString` function generates speech asynchronously, which means that control is returned to your application before speech has finished, perhaps even before the speech is first audible.

If `SpeakCFString` is called while the speech channel is currently speaking the contents of another text string, the speech stops immediately and the new text string is spoken as soon as possible.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`SpeechSynthesis.h`

SpeakString

Begins speaking a text string.

```
OSErr SpeakString (
   ConstStr255Param textToBeSpoken
);
```

Parameters

textToBeSpoken

The string to be spoken.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
The `SpeakString` function attempts to speak the Pascal-style text string contained in the string `textToBeSpoken`. Speech is produced asynchronously using the default system voice. When an application calls this function, the Speech Synthesis Manager makes a copy of the passed string and creates any structures required to speak it. As soon as speaking has begun, control is returned to the application. The synthesized speech is generated asynchronously to the application so that normal processing can continue while the text is being spoken. No further interaction with the Speech Synthesis Manager is required at this point, and the application is free to release the memory that the original string occupied.

If `SpeakString` is called while a prior string is still being spoken, the sound currently being synthesized is interrupted immediately. Conversion of the new text into speech is then begun. If you pass a zero-length string (or, in C, a `null` pointer) to `SpeakString`, the Speech Synthesis Manager stops any speech previously being synthesized by `SpeakString` without generating additional speech. If your application uses `SpeakString`, it is often a good idea to stop any speech in progress whenever your application receives a suspend event. Calling `SpeakString` with a zero-length string has no effect on speech channels other than the one managed internally by the Speech Synthesis Manager for the `SpeakString` function.)

The text passed to the `SpeakString` function may contain embedded speech commands.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeakText

Begins speaking a buffer of text.

```
OSErr SpeakText (
   SpeechChannel chan,
   const void *textBuf,
   unsigned long textBytes
);
```

Parameters

`chan`
: The speech channel through which speech is to be spoken.

`textBuf`
: A pointer to the first byte of text to spoken.

`textBytes`
: The number of bytes of text to spoken.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
Like `SpeakString`, the `SpeakText` function also generates speech, but through a speech channel through which you can exert control over the generated speech.

The `SpeakText` function converts the text stream specified by the `textBuf` and `textBytes` parameters into speech using the voice and control settings for the speech channel `chan`, which should be created with the `NewSpeechChannel` function. The speech is generated asynchronously. This means that control is returned to your application before the speech has finished (and probably even before it has begun). The maximum length of the text buffer that can be spoken is limited only by the available RAM.

If `SpeakText` is called while the channel is currently busy speaking the contents of a prior text buffer, it immediately stops speaking from the prior buffer and begins speaking from the new text buffer as soon as possible. If you pass a zero-length string (or, in C, a `null` pointer) to `SpeakText`, the Speech Synthesis Manager stops all speech currently being synthesized by the speech channel specified in the `chan` parameter without generating additional speech.

The text buffer must be locked in memory and must not move while the Speech Synthesis Manager processes it. This buffer is read at interrupt time, and moving it could cause a system crash. If your application defines a text-done callback function, then it can move the text buffer or dispose of it once the callback function is executed.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechBusy

Determines whether any channels of speech are currently synthesizing speech.

```
SInt16 SpeechBusy (
   void
);
```

Return Value
The number of speech channels that are currently synthesizing speech in the application. This is useful when you want to ensure that an earlier speech request has been completed before having the system speak again. Paused speech channels are counted among those that are synthesizing speech.

The speech channel that the Speech Synthesis Manager allocates internally in response to calls to the `SpeakString` function is counted in the number returned by `SpeechBusy`. Thus, if you use just `SpeakString` to initiate speech, `SpeechBusy` always returns `1` as long as speech is being produced. When `SpeechBusy` returns `0`, all speech has finished.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechBusySystemWide

Determines if any speech is currently being synthesized in your application or elsewhere on the computer.

```
SInt16 SpeechBusySystemWide (
   void
);
```

Return Value
The total number of speech channels currently synthesizing speech on the computer, whether they were initiated by your application or process's code or by some other process executing concurrently. Paused speech channels are counted among those channels that are synthesizing speech.

Discussion
This function is useful when you want to ensure that no speech is currently being produced anywhere on the Macintosh computer before initiating speech. Although the Speech Synthesis Manager allows different applications to produce speech simultaneously, this can be confusing to the user. As a result, it is often a good idea for your application to check that no other process is producing speech before producing speech itself. If the difference between the values returned by `SpeechBusySystemWide` and the `SpeechBusy` function is `0`, no other process is producing speech.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechManagerVersion

Determines the current version of the Speech Synthesis Manager installed in the system.

```
NumVersion SpeechManagerVersion (
   void
);
```

Return Value
The version of the Speech Synthesis Manager installed in the system, in the format of the first 4 bytes of a `'vers'` resource.

Discussion
Use this call to determine whether your program can access features of the Speech Synthesis Manager that are included in some Speech Synthesis Manager releases but not in earlier ones.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

StopSpeech

Terminates speech immediately on the specified channel.

```
OSErr StopSpeech (
   SpeechChannel chan
);
```

Parameters

chan

The speech channel on which speech is to be stopped.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `StopSpeech` function immediately terminates speech on the channel specified by the `chan` parameter. After returning from `StopSpeech`, your application can safely release any text buffer that the speech synthesizer has been using. You can call `StopSpeech` for an already idle channel without ill effect.

You can also stop speech by passing a zero-length string (or, in C, a `null` pointer) to one of the `SpeakString`, `SpeakText`, or `SpeakBuffer` functions. Doing this stops speech only in the specified speech channel (or, in the case of `SpeakString`, in the speech channel managed internally by the Speech Synthesis Manager).

Before calling the `StopSpeech` function, you can use the `SpeechBusy` function, which is described in `SpeechBusy` (page 34), to determine if a synthesizer is still speaking. If you are working with multiple speech channels, you can use the status selector with the function `GetSpeechInfo` which is described in `GetSpeechInfo` (page 16), to determine if a specific channel is still speaking.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

StopSpeechAt

Terminates speech delivery on a specified channel either immediately or at the end of the current word or sentence.

```
OSErr StopSpeechAt (
   SpeechChannel chan,
   SInt32 whereToStop
);
```

Parameters

chan

The speech channel on which speech is to be stopped.

whereToStop

A constant indicating when speech processing should stop. Pass the constant `kImmediate` to stop immediately, even in the middle of a word. Pass `kEndOfWord` or `kEndOfSentence` to stop speech at the end of the current word or sentence, respectively.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion

The `StopSpeechAt` function halts the production of speech on the channel specified by `chan` at a specified point in the text. This function returns immediately, although speech output continues until the specified point has been reached.

If you call the `StopSpeechAt` function before the Speech Synthesis Manager finishes processing input text, then the function might return before some input text has yet to be spoken. Thus, before disposing of the text buffer, your application should wait until its text-done callback function has been called (if one has been defined), or until it can determine (by, for example obtaining a speech status information structure) that the Speech Synthesis Manager is no longer processing input text.

If the end of the input text buffer is reached before the specified stopping point, the speech synthesizer stops at the end of the buffer without generating an error.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

TextToPhonemes

Converts a buffer of textual data into phonemic data.

```
OSErr TextToPhonemes (
   SpeechChannel chan,
   const void *textBuf,
   unsigned long textBytes,
   Handle phonemeBuf,
   long *phonemeBytes
);
```

Parameters

`chan`

A speech channel whose associated synthesizer and voice are to be used for the conversion process.

`textBuf`

A pointer to a buffer of text to be converted.

`textBytes`

The number of bytes of text to be converted.

`phonemeBuf`

A handle to a buffer to be used to store the phonemic data. The `TextToPhonemes` function may resize the relocatable block referenced by this handle.

`phonemeBytes`

On return, a pointer to the number of bytes of phonemic data written to the handle.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
Converting textual data into phonemic data is particularly useful during application development, when you might wish to adjust phrases that your application generates to produce smoother speech. By first converting the target phrase into phonemes, you can see what the synthesizer will try to speak. Then you need correct only the parts that would not have been spoken the way you want.

The `TextToPhonemes` function converts the `textBytes` bytes of textual data pointed to by the `textBuf` parameter to phonemic data, which it writes into the relocatable block specified by the `phonemeBuf` parameter. If necessary, `TextToPhonemes` resizes this relocatable block. The `TextToPhonemes` function sets the `phonemeBytes` parameter to the number of bytes of phonemic data actually written.

If the textual data is contained in a relocatable block, a handle to that block must be locked before the `TextToPhonemes` function is called.

The data returned by `TextToPhonemes` corresponds precisely to the phonemes that would be spoken had the input text been sent to `SpeakText` instead. All current mode settings for the speech channel specified by `chan` are applied to the converted speech. No callbacks are generated while the `TextToPhonemes` function is generating its output.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

UseDictionary

Installs the designated dictionary into a speech channel.

```
OSErr UseDictionary (
   SpeechChannel chan,
   Handle dictionary
);
```

Parameters

`chan`
The speech channel into which a dictionary is to be installed.

`dictionary`
A handle to the dictionary data. This is often a handle to a resource of type `'dict'`.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
The `UseDictionary` function attempts to install the dictionary data referenced by the `dictionary` parameter into the speech channel referenced by the `chan` parameter. The synthesizer will use whatever elements of the dictionary resource it considers useful to the speech conversion process. Some speech synthesizers might ignore certain types of dictionary entries.

After the `UseDictionary` function returns, your application is free to release any storage allocated for the dictionary handle. The search order for application-provided dictionaries is last-in, first-searched.

All details of how an application-provided dictionary is represented within the speech synthesizer are dependent on the specific synthesizer implementation and are private to the synthesizer.

Pronunciation dictionaries allow your application to override the default Speech Synthesis Manager pronunciations of individual words, such as names with unusual spellings.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

UseSpeechDictionary

Registers a speech dictionary with a speech channel.

```
OSErr UseSpeechDictionary (
   SpeechChannel chan,
   CFDictionaryRef speechDictionary
);
```

Parameters

`chan`
The speech channel with which the specified speech dictionary is to be registered.

`speechDictionary`
A speech dictionary to be registered with the specified speech channel, represented as a `CFDictionary` object. See "`Speech Dictionary Keys`" (page 82) for the keys you can use in the dictionary.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 83).

Discussion
The UseSpeechDictionary function is the Core Foundation-based equivalent of the `UseDictionary` (page 38) function.

The `UseSpeechDictionary` function registers the `CFDictionary` object referenced by the `speechDictionary` parameter with the speech channel referenced by the `chan` parameter. Speech dictionaries allow your application to override a synthesizer's default pronunciations of individual words, such as names with unusual spellings. A synthesizer will use whatever elements of the dictionary it considers useful in the speech conversion process. Some speech synthesizers might ignore certain types of dictionary entries.

Multiple dictionaries can be registered with a synthesizer. If the same word appears in multiple dictionaries, the synthesizer will use the one from the dictionary with the most recent date.

Note that because a speech dictionary is a `CFDictionary` object, it can be loaded from an XML-based property list file. An example of such a file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
```

```
    <key>LocaleIdentifier</key>
    <string>en_US</string>
    <key>ModificationDate</key>
    <string>2006-12-21 11:59:25 -0800</string>
    <key>Pronunciations</key>
    <array>
        <dict>
            <key>Phonemes</key>
            <string>_hEY_yUW</string>
            <key>Spelling</key>
            <string>Hello</string>
        </dict>
    </array>
    <key>Abbreviations</key>
    <array>
        <dict>
            <key>Phonemes</key>
            <string>_OW_sAEkz</string>
            <key>Spelling</key>
            <string>OSAX</string>
        </dict>
    </array>
</dict>
</plist>
```

After the `UseSpeechDictionary` function returns, your application is free to release the `CFDictionary` object referenced by the `speechDictionary` parameter.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

Callbacks

SpeechDoneProcPtr

Defines a pointer to a speech-done callback function which is called when the Speech Synthesis Manager finishes speaking a buffer of text.

```
typedef void (*SpeechDoneProcPtr) (
    SpeechChannel chan,
    SRefCon refCon
);
```

If you name your function `MySpeechDoneProc`, you would declare it like this:

```
void MySpeechDoneProc (
    SpeechChannel chan,
    long refCon
);
```

Parameters

chan

The speech channel that has finished processing input text.

refCon

The reference constant associated with the speech channel.

Discussion

If a speech-done callback function is installed in a speech channel, then the Speech Synthesis Manager calls this function when it finishes speaking a buffer of text.

You can specify a speech-done callback function by passing the `soSpeechDoneCallBack` selector to the `SetSpeechInfo` function.

You might use a speech-done callback function if you need to update some visual indicator that shows what text is currently being spoken. For example, suppose your application passes text buffers to the Speech Synthesis Manager one paragraph at a time. Your speech-done callback function might set a global flag variable to indicate to the application that the Speech Synthesis Manager has finished speaking a paragraph. When a function called by your application's main event loop checks the global flag variable and determines that it has been set, the function might ensure that the next paragraph of text is visible.

You might use a speech-done callback function to set a flag variable that alerts the application that it should pass a new buffer of text to the Speech Synthesis Manager. If you do so, however, there might be a noticeable pause as the Speech Synthesis Manager switches from processing one text buffer to another. Ordinarily, it is easier to achieve this goal by using a text-done callback function, as described earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechErrorCFProcPtr

Defines a pointer to an error callback function that handles syntax errors within commands embedded in a `CFString` object being processed by the Speech Synthesis Manager.

```
typedef void (*SpeechErrorCFProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    CFErrorRef theError
);
```

If you name your function `MySpeechErrorCFProc`, you would declare it like this:

```
void MySpeechErrorCFProc (
    SpeechChannel chan,
    long refCon,
    CFErrorRef theError
);
```

Parameters

chan

The speech channel that has finished processing input text.

refCon
: The reference constant associated with the speech channel.

theError
: The error that occurred in processing an embedded command.

Discussion

An error callback function defined by the `SpeechErrorCFProcPtr` is the Core Foundation-based equivalent of an error callback function defined by `SpeechErrorProcPtr` (page 42). The Speech Synthesis Manager calls a speech channel's error callback function whenever it encounters a syntax error within a command embedded in a `CFString` object it is processing. This can be useful during application debugging, to detect problems with commands that you have embedded in strings that your application speaks. It can also be useful if your application allows users to embed commands within strings. Your application might display an alert indicating that the Speech Synthesis Manager encountered a problem in processing an embedded command.

Ordinarily, the error information that the Speech Synthesis Manager provides the error callback function should be sufficient. However, if your application needs information about errors that occurred before the error callback function was enabled, the application (including the error callback function) can call the `CopySpeechProperty` (page 12) function with the `kSpeechErrorsProperty` property.

You can specify an error callback function by passing the `kSpeechErrorCFCallback` property to the `SetSpeechProperty` (page 29) function.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`SpeechSynthesis.h`

SpeechErrorProcPtr

Defines a pointer to an error callback function that handles syntax errors within commands embedded in a text buffer being processed by the Speech Synthesis Manager.

```
typedef void (*SpeechErrorProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    OSErr theError,
    long bytePos
);
```

If you name your function `MySpeechErrorProc`, you would declare it like this:

```
void MySpeechErrorProc (
    SpeechChannel chan,
    long refCon,
    OSErr theError,
    long bytePos
);
```

Parameters

chan
: The speech channel that has finished processing input text.

`refCon`

The reference constant associated with the speech channel.

`theError`

The error that occurred in processing an embedded command.

`bytePos`

The number of bytes from the beginning of the text buffer being spoken to the error encountered.

Discussion

The Speech Synthesis Manager calls a speech channel's error callback function whenever it encounters a syntax error within a command embedded in a text buffer it is processing. This can be useful during application debugging, to detect problems with commands that you have embedded in text buffers that your application speaks. It can also be useful if your application allows users to embed commands within text buffers. Your application might display an alert indicating that the Speech Synthesis Manager encountered a problem in processing an embedded command.

Ordinarily, the error information that the Speech Synthesis Manager provides the error callback function should be sufficient. However, if your application needs information about errors that occurred before the error callback function was enabled, the application (including the error callback function) can call the `GetSpeechInfo` function with the `soErrors` selector.

You can specify an error callback function by passing the `soErrorCallBack` selector to the `SetSpeechInfo` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechPhonemeProcPtr

Defines a pointer to a phoneme callback function that is called by the Speech Synthesis Manager before it pronounces a phoneme.

```
typedef void (*SpeechPhonemeProcPtr)
(
    SpeechChannel chan,
    SRefCon refCon,
    short phonemeOpcode
);
```

If you name your function `MySpeechPhonemeProc`, you would declare it like this:

```
void MySpeechPhonemeProc (
    SpeechChannel chan,
    long refCon,
    short phonemeOpcode
);
```

Parameters

`chan`

The speech channel that has finished processing input text.

refCon

The reference constant associated with the speech channel.

phonemeOpcode

The phoneme about to be pronounced.

Discussion

The Speech Synthesis Manager calls a speech channel's phoneme callback function just before it pronounces a phoneme. For example, your application might use such a callback function to enable mouth synchronization. In this case, the callback function would set a global flag variable to indicate that the phoneme being pronounced is changing and another global variable to `phonemeOpcode`. A function called by your application's main event loop could detect that the phoneme being pronounced is changing and update a picture of a mouth to reflect the current phoneme. In practice, providing a visual indication of the pronunciation of a phoneme requires several consecutive pictures of mouth movement to be rapidly displayed. Consult the linguistics literature for information on mouth movements associated with different phonemes.

You can specify a phoneme callback function by passing the `soPhonemeCallBack` selector to the `SetSpeechInfo` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechSyncProcPtr

Defines a pointer to a synchronization callback function that is called when the Speech Synthesis Manager encounters a synchronization command embedded in a text buffer.

```
typedef void (*SpeechSyncProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    OSType syncMessage
);
```

If you name your function `MySpeechSyncProc`, you would declare it like this:

```
void MySpeechSyncProc (
    SpeechChannel chan,
    long refCon,
    OSType syncMessage
);
```

Parameters

chan

The speech channel that has finished processing input text.

refCon

The reference constant associated with the speech channel.

syncMessage

The synchronization message passed in the embedded command. Usually, you use this message to distinguish between several different types of synchronization commands, but you can use it any way you wish.

Discussion

The Speech Synthesis Manager calls a speech channel's synchronization callback function whenever it encounters a synchronization command embedded in a text buffer. You might use the synchronization callback function to provide a callback not ordinarily provided. For example, you might inset synchronization commands at the end of every sentence in a text buffer, or you might enter synchronization commands after every numeric value in the text. However, to synchronize your application with phonemes or words, it makes more sense to use the built-in phoneme and word callback functions, defined in `SpeechPhonemeProcPtr` (page 43).

You can specify a synchronization callback function by passing the `soSyncCallBack` selector to the `SetSpeechInfo` function and embedding a synchronization command within a text buffer passed to the `SpeakText` or `SpeakBuffer` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechTextDoneProcPtr

Defines a pointer to a text-done callback function that is called when the Speech Synthesis Manager has finished processing a buffer of text.

```
typedef void (*SpeechTextDoneProcPtr)
(
    SpeechChannel chan,
    SRefCon refCon,
    void ** nextBuf,
    unsigned long * byteLen,
    long * controlFlags
);
```

If you name your function `MySpeechTextDoneProc`, you would declare it like this:

```
void MySpeechTextDoneProc (
    SpeechChannel chan,
    long refCon,
    void ** nextBuf,
    unsigned long * byteLen,
    long * controlFlags
);
```

Parameters

`chan`

The speech channel that has finished processing input text.

`refCon`

The reference constant associated with the speech channel.

`nextBuf`

On return, a pointer to the next buffer of text to process or `NULL` if your application has no additional text to be spoken. This parameter is mostly for internal use by the Speech Synthesis Manager.

byteLen

On return, a pointer to the number of bytes of the text buffer pointed to by the `nextBuf` parameter.

controlFlags

On return, a pointer to the control flags to be used in generating the next buffer of text.

Discussion

If a text-done callback function is installed in a speech channel, then the Speech Synthesis Manager calls this function when it finishes processing a buffer of text. The Speech Synthesis Manager might not yet have completed finishing speaking the text and indeed might not have started speaking it.

You can specify a text-done callback function by passing the `soTextDoneCallBack` selector to the `SetSpeechInfo` function.

A common use of a text-done callback function is to alert your application once the text passed to the `SpeakText` or `SpeakBuffer` function can be disposed of (or, when the text is contained within a locked relocatable block, when the relocatable block can be unlocked). The Speech Synthesis Manager copies the text you pass to the `SpeakText` or `SpeakBuffer` function into an internal buffer. Once it has finished processing the text, you may dispose of the original text buffer, even if speech is not yet complete. However, if you wish to write a callback function that executes when speech is completed, see the definition of a speech-done callback function below.

Although most applications will not need to, your callback function can indicate to the Speech Synthesis Manager whether there is another buffer of text to speak. If there is another buffer, your callback function should reference it by setting the `nextBuf` and `byteLen` parameters to appropriate values. (Your callback function might also change the control flags to be used to process the speech by altering the value in the `controlFlags` parameter.) Setting these parameters allows the Speech Synthesis Manager to generate uninterrupted speech. If there is no more text to speak, your callback function should set `nextBuf` to `NULL`. In this case, the Speech Synthesis Manager ignores the `byteLen` and `controlFlags` parameters.

If your text-done callback function does not change the values of the `nextBuf` and `byteLen` parameters, the text buffer just spoken will be spoken again.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechWordCFProcPtr

Defines a pointer to a Core Foundation-based word callback function that is called by the Speech Synthesis Manager before it pronounces a word.

```
typedef void (*SpeechWordCFProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    CFStringRef aString,
    CFRange wordRange
);
```

If you name your function `MySpeechWordCFProc`, you would declare it like this:

```
void MySpeechWordCFProc (
    SpeechChannel chan,
    SRefCon refCon,
    CFStringRef aString,
    CFRange wordRange
);
```

Parameters

chan
: The speech channel that has finished processing input text.

refCon
: The reference constant associated with the speech channel.

aString
: The original string passed to the speech synthesizer in the `SpeakCFString` (page 31) call.

wordRange
: The range of characters in `aString` that corresponds to the word.

Discussion

A word callback function defined by the `SpeechWordCFProcPtr` is the Core Foundation-based equivalent of a word callback function defined by `SpeechWordProcPtr` (page 47). The Speech Synthesis Manager calls a speech channel's word callback function just before it pronounces a word. You might use such a callback function, for example, to highlight the word about to be spoken in a window.

You can specify a word callback function by passing the `kSpeechWordCFCallBack` property to the`SetSpeechProperty` (page 29) function.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`SpeechSynthesis.h`

SpeechWordProcPtr

Defines a pointer to a word callback function that is called by the Speech Synthesis Manager before it pronounces a word.

```
typedef void (*SpeechWordProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    unsigned long wordPos,
    unsigned short wordLen
);
```

If you name your function `MySpeechWordProc`, you would declare it like this:

```
void MySpeechWordProc (
    SpeechChannel chan,
    long refCon,
    unsigned long wordPos,
    unsigned short wordLen
);
```

Parameters

chan
: The speech channel that has finished processing input text.

refCon
: The reference constant associated with the speech channel.

wordPos
: The number of bytes between the beginning of the text buffer and the beginning of the word about to be pronounced.

wordLen
: The length in bytes of the word about to be pronounced.

Discussion

The Speech Synthesis Manager calls a speech channel's word callback function just before it pronounces a word. You might use such a callback function, for example, to draw the word about to be spoken in a window. In this case, the callback function would set a global flag variable to indicate that the word being spoken is changing and another two global variables to `wordPos` and `wordLen`. A function called by your application's main event loop could detect that the word being spoken is changing and draw the word in a window.

You can specify a word callback function by passing the `soWordCallBack` selector to the `SetSpeechInfo` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

Data Types

DelimiterInfo

Defines a delimiter information structure.

```
struct DelimiterInfo {
    Byte startDelimiter[2];
    Byte endDelimiter[2];
};
typedef struct DelimiterInfo DelimiterInfo;
```

Fields

`startDelimiter`
: The start delimiter for an embedded command. By default, the start delimiter is "`[[`".

`endDelimiter`
: The end delimiter for an embedded command. By default, the end delimiter is "`]]`".

Discussion

A delimiter information structure defines the characters used to indicate the beginning and end of a command embedded in text. A delimiter can be one or two characters.

Ordinarily, applications that support embedded speech commands should not change the start or end delimiters. However, if for some reason you must change the delimiters, you can use the `SetSpeechInfo` function with the `soCommandDelimiter` selector. For example, you might do this if a text buffer naturally includes the delimiter strings. Before passing such a buffer to the Speech Synthesis Manager, you can change the delimiter strings to some two-character sequences not used in the buffer and then change the delimiter strings back once processing of the buffer is complete.

If a single-byte delimiter is desired, it should be followed by a `NULL` (0) byte. If the delimiter strings both consist of two `NULL` bytes, embedded command processing is disabled.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

PhonemeDescriptor

Defines a phoneme descriptor structure.

```
struct PhonemeDescriptor {
    SInt16 phonemeCount;
    PhonemeInfo thePhonemes[1];
};
typedef struct PhonemeDescriptor PhonemeDescriptor;
```

Fields

`phonemeCount`
: The number of phonemes that the current synthesizer defines. Typically, this will correspond to the number of phonemes in the language supported by the synthesizer.

`thePhonemes`
: An array of phoneme information structures.

Discussion
By calling the `GetSpeechInfo` (page 16) function with the `soPhonemeSymbols` selector, you can obtain a phoneme descriptor structure, which describes all phonemes defined for the current synthesizer.

A common use for a phoneme descriptor structure is to provide a graphical display to the user of all available phonemes. Such a list is used only for a user entering phonemic data directly rather than just entering text.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

PhonemeInfo

Defines a structure that stores information about a phoneme.

```
struct PhonemeInfo {
    SInt16 opcode;
    Str15 phStr;
    Str31 exampleStr;
    SInt16 hiliteStart;
    SInt16 hiliteEnd;
};
typedef struct PhonemeInfo PhonemeInfo;
```

Fields

`opcode`
: The opcode for the phoneme.

`phStr`
: The string used to represent the phoneme. The string does not necessarily have a phonetic connection to the phoneme, but might simply be an abstract textual representation of it.

`exampleStr`
: An example word that illustrates use of the phoneme.

`hiliteStart`
: The number of characters in the example word that precede the portion of that word representing the phoneme.

`hiliteEnd`
: The number of characters between the beginning of the example word and the end of the portion of that word representing the phoneme.

Discussion

Ordinarily, you use a phoneme information structure to show the user how to enter text to represent a particular phoneme when the `'PHON'` input mode is activated.

You might use the information contained in the `hiliteStart` and `hiliteEnd` fields to highlight the characters in the example word that represent the phoneme.

To obtain a phoneme information structure for an individual phoneme, you must obtain a list of phonemes through a phoneme descriptor structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechChannelRecord

Represents a speech channel.

```
struct SpeechChannelRecord {
    long data[1];
};
typedef struct SpeechChannelRecord SpeechChannelRecord;
typedef SpeechChannelRecord * SpeechChannel;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechDoneUPP

Defines a universal procedure pointer (UPP) to a speech-done callback function.

```
typedef SpeechDoneProcPtr SpeechDoneUPP;
```

Discussion
For more information, see the description of the `SpeechDoneProcPtr` (page 40) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechErrorInfo

Defines a speech error information structure.

```
struct SpeechErrorInfo {
    SInt16 count;
    OSErr oldest;
    long oldPos;
    OSErr newest;
    long newPos;
};
typedef struct SpeechErrorInfo SpeechErrorInfo;
```

Fields

`count`
: The number of errors that have occurred in processing the current text buffer since the last call to the `GetSpeechInfo` function with the `soErrors` selector. Of these errors, you can find information about only the first and last error that occurred.

`oldest`
: The error code of the first error that occurred after the previous call to the `GetSpeechInfo` function with the `soErrors` selector.

`oldPos`
: The character position within the text buffer being processed of the first error that occurred after the previous call to the `GetSpeechInfo` function with the `soErrors` selector.

`newest`
: The error code of the most recent error.

`newPos`
: The character position within the text buffer being processed of the most recent error.

Discussion
By calling the `GetSpeechInfo` (page 16) function with the `soErrors` selector, you can obtain a speech error information structure, which shows what Speech Synthesis Manager errors occurred while processing a text buffer on a given speech channel.

Speech error information structures never include errors that are returned by Speech Synthesis Manager functions. Instead, they reflect only errors encountered directly in the processing of text, and, in particular, in the processing of commands embedded within text.

The speech error information structure keeps track of only the most recent error and the first error that occurred after the previous call to the `GetSpeechInfo` function with the `soErrors` selector. If your application needs to keep track of all errors, then you should install an error callback function, `SpeechErrorProcPtr` (page 42).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechErrorUPP

Defines a universal procedure pointer (UPP) to an error callback function.

```
typedef SpeechErrorProcPtr SpeechErrorUPP;
```

Discussion
For more information, see the description of the `SpeechErrorProcPtr` (page 42) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechPhonemeUPP

Defines a universal procedure pointer (UPP) to a phoneme callback function.

```
typedef SpeechPhonemeProcPtr SpeechPhonemeUPP;
```

Discussion
For more information, see the description of the `SpeechPhonemeProcPtr` (page 43) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechStatusInfo

Defines a a speech status information structure, which stores information about the status of a speech channel.

```
struct SpeechStatusInfo {
    Boolean outputBusy;
    Boolean outputPaused;
    long inputBytesLeft;
    SInt16 phonemeCode;
};
typedef struct SpeechStatusInfo SpeechStatusInfo;
```

Fields

`outputBusy`
: Whether the speech channel is currently producing speech. A speech channel is considered to be producing speech even at some times when no audio data is being produced through the Macintosh speaker. This occurs, for example, when the Speech Synthesis Manager is processing an input buffer but has not yet initiated speech or when speech output is paused.

`outputPaused`
: Whether speech output in the speech channel has been paused by a call to the `PauseSpeechAt` (page 27) function.

`inputBytesLeft`
: The number of input bytes of the text that the speech channel must still process. When `inputBytesLeft` is 0, the buffer of input text passed to one of the `SpeakText` or `SpeakBuffer` functions may be disposed of. When you call the `SpeakString` function, the Speech Synthesis Manager stores a duplicate of the string to be spoken in an internal buffer; thus, you may delete the original string immediately after calling `SpeakString`.

`phonemeCode`
: The opcode for the phoneme that the speech channel is currently processing.

Discussion

By calling the `GetSpeechInfo` (page 16) function with the `soStatus` selector, you can find out information about the status of a speech channel.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechSyncUPP

Defines a universal procedure pointer (UPP) to a synchronization callback function.

```
typedef SpeechSyncProcPtr SpeechSyncUPP;
```

Discussion

For more information, see the description of the `SpeechSyncProcPtr` (page 44) callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechTextDoneUPP

Defines a universal procedure pointer (UPP) to a text-done callback function.

```
typedef SpeechTextDoneProcPtr SpeechTextDoneUPP;
```

Discussion

For more information, see the description of the `SpeechTextDoneProcPtr` (page 45) callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechVersionInfo

Defines a speech version information structure.

```
struct SpeechVersionInfo {
    OSType synthType;
    OSType synthSubType;
    OSType synthManufacturer;
    SInt32 synthFlags;
    NumVersion synthVersion;
};
typedef struct SpeechVersionInfo SpeechVersionInfo;
```

Fields

`synthType`

The general type of the synthesizer. For the current version of the Speech Synthesis Manager, this field always contains the value `kTextToSpeechSynthType`, indicating that the synthesizer converts text into speech.

`synthSubType`

The specific type of the synthesizer. Currently, no specific types of synthesizer are defined. If you define a new type of synthesizer, you should register the four-character code for your type with Developer Technical Support.

`synthManufacturer`

A unique identification of a synthesizer engine. If you develop synthesizers, then you should register a different four-character code for each synthesizer you develop with Developer Technical Support. The `creatorID` field of the voice specification structure and the `synthCreator` field of a speech extension data structure should each be set to the value stored in this field for the desired synthesizer.

`synthFlags`

A set of flags indicating which synthesizer features are activated. Specific constants define the bits in this field whose meanings are defined for all synthesizers.

`synthVersion`

The version number of the synthesizer.

Discussion

By calling the `GetSpeechInfo` (page 16) function with the `soSynthType` selector, you can obtain a speech version information structure, which provides information about the speech synthesizer currently being used.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechWordUPP

Defines a universal procedure pointer (UPP) to a word callback function.

```
typedef SpeechWordProcPtr SpeechWordUPP;
```

Discussion
For more information, see the description of the `SpeechWordProcPtr` (page 47) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechXtndData

Defines a speech extension data structure.

```
struct SpeechXtndData {
    OSType synthCreator;
    Byte synthData[2];
};
typedef struct SpeechXtndData SpeechXtndData;
```

Fields

`synthCreator`
The synthesizer's creator ID, identical to the value stored in the `synthManufacturer` field of a speech version information structure. You should set this field to the appropriate value before calling `GetSpeechInfo` or `SetSpeechInfo`.

`synthData`
Synthesizer-specific data. The size and format of the data in this field may vary.

Discussion
The speech extension data structure allows you to use the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions with selectors defined by particular synthesizers. By requiring that you pass to one of these functions a pointer to a speech extension data structure, synthesizers can permit the exchange of data in any format.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

VoiceDescription

Defines a voice description structure.

```
struct VoiceDescription {
    SInt32 length;
    VoiceSpec voice;
    SInt32 version;
    Str63 name;
    Str255 comment;
    SInt16 gender;
    SInt16 age;
    SInt16 script;
    SInt16 language;
    SInt16 region;
    SInt32 reserved[4];
};
typedef struct VoiceDescription VoiceDescription;
```

Fields

`length`
: The size of the voice description structure, in bytes.

`voice`
: A voice specification structure that uniquely identifies the voice.

`version`
: The version number of the voice.

`name`
: The name of the voice, preceded by a length byte. Names must be 63 characters or less.

`comment`
: Additional text information about the voice. Some synthesizers use this field to store a phrase that can be spoken.

`gender`
: The gender of the individual represented by the voice. See "Gender Constants" (page 59).

`age`
: The approximate age in years of the individual represented by the voice.

`script`
: In Mac OS X v10.4.7 and later, the encoding code of the text that the voice can process.

  Note that this field contains a 16-bit value. You can use any of the 16-bit values described in `External_String_Encodings` or `CFStringBuiltInEncodings`. However, if you need to use a 32-bit value, such as `kCFStringEncodingUTF8`, you pass the value in the first array element of the `reserved` field, and you also specify `-1` or `kCFStringEncodingInvalidId` in the `script` field.

`language`
: A code that indicates the language of voice output.

`region`
: A code that indicates the region represented by the voice.

`reserved`
: Reserved. May be used to hold a 32-bit encoding value, if necessary (see the description of the `script` field for more information).

Discussion

By calling the `GetVoiceDescription` (page 18) function, you can obtain information about a voice in a voice description structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

VoiceFileInfo

Defines a voice file information structure.

```
struct VoiceFileInfo {
    FSSpec fileSpec;
    SInt16 resID;
};
typedef struct VoiceFileInfo VoiceFileInfo;
```

Fields

`fileSpec`
: A file system specification structure that contains the volume, directory, and name of the file containing the voice. Generally, files containing a single voice are of type `kTextToSpeechVoiceFileType`, and files containing multiple voices are of type `kTextToSpeechVoiceBundleType`.

`resID`
: The resource ID of the voice in the file. Voices are stored in resources of type `kTextToSpeechVoiceType`.

Discussion
A voice file information structure specifies the file in which a voice is stored and the resource ID of the voice within that file. Use the `GetVoiceInfo` (page 19) function to obtain a voice file information structure for a voice.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

VoiceSpec

Defines a voice specification structure.

```
struct VoiceSpec {
    OSType creator;
    OSType id;
};
typedef struct VoiceSpec VoiceSpec;
typedef VoiceSpec * VoiceSpecPtr;
```

Fields

`creator`
: The synthesizer that is required to use the voice. This is equivalent to the value contained in the `synthManufacturer` field of a speech version information structure and that contained in the `synthCreator` field of a speech extension data structure. The set of `OSType` values specified entirely by space characters and lowercase letters is reserved.

`id`
: The voice ID of the voice for the synthesizer. Every voice on a synthesizer has a unique ID.

Discussion
A voice specification structure provides a unique specification that you must use to obtain information about a voice. You also must use a voice specification structure if you wish to create a speech channel that generates speech in a voice other than the current system default voice.

To ensure compatibility with future versions of the Speech Synthesis Manager, you should never fill in the fields of a voice specification structure yourself. Instead, you should create a voice specification structure by using the `MakeVoiceSpec` (page 23) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

Constants

Control Flags Constants

Flags that indicate which synthesizer features are active.

```
enum {
    kNoEndingProsody = 1,
    kNoSpeechInterrupt = 2,
    kPreflightThenPause = 4
};
```

Constants
`kNoEndingProsody`

Disables prosody at end of sentences. The `kNoEndingProsody` flag bit is used to control whether or not the speech synthesizer automatically applies ending prosody, the speech tone and cadence that normally occur at the end of a statement. Under normal circumstances (for example, when the flag bit is not set), ending prosody is applied to the speech when the end of the `textBuf` data is reached. This default behavior can be disabled by setting the `kNoEndingProsody` flag bit.

Some synthesizers do not speak until the `kNoEndingProsody` flag bit is reset, or they encounter a period in the text, or `textBuf` is full.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kNoSpeechInterrupt`
: Does not interrupt current speech. The `kNoSpeechInterrupt` flag bit is used to control the behavior of `SpeakBuffer` when called on a speech channel that is still busy. When the flag bit is not set, `SpeakBuffer` behaves similarly to `SpeakString` and `SpeakText`. Any speech currently being produced on the specified speech channel is immediately interrupted, and then the new text buffer is spoken. When the `kNoSpeechInterrupt` flag bit is set, however, a request to speak on a channel that is still busy processing a prior text buffer will result in an error. The new buffer is ignored and the error `synthNotReady` is returned. If the prior text buffer has been fully processed, the new buffer is spoken normally. One way of achieving continuous speech without using callback functions is to continually call `SpeakBuffer` with the `kNoSpeechInterrupt` flag bit set until the function returns `noErr`. The function will then execute as soon as the first text buffer has been processed.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechSynthesis.h`.

`kPreflightThenPause`
: Computes speech without generating.The `kPreflightThenPause` flag bit is used to minimize the latency experienced when the speech synthesizer is attempting to speak. Ordinarily, whenever a call to `SpeakString`, `SpeakText`, or `SpeakBuffer` is made, the speech synthesizer must perform a certain amount of initial processing before speech output is heard. This startup latency can vary from a few milliseconds to several seconds depending upon which speech synthesizer is being used. Recognizing that larger startup delays might be detrimental to certain applications, a mechanism exists to allow the synthesizer to perform any necessary computations at noncritical times. Once the computations have been completed, the speech is able to start instantly. When the `kPreflightThenPause` flag bit is set, the speech synthesizer will process the input text as necessary to the point where it is ready to begin producing speech output. At this point, the synthesizer will enter a paused state and return to the caller. When the application is ready to produce speech, it should call the `ContinueSpeech` function to begin speaking.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechSynthesis.h`.

Discussion

These constants are used in the `controlFlags` parameter of the `SpeakBuffer` (page 30) function and in the `synthFlags1` field of the `SpeechVersionInfo` (page 54) structure.

Declared In

`SpeechSynthesis.h`

Gender Constants

Constants that indicate the gender of the individual represented by a voice.

```
enum {
    kNeuter = 0,
    kMale = 1,
    kFemale = 2
};
```

Constants

`kNeuter`
: Neuter voice.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechSynthesis.h`.

`kMale`

Male voice.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kFemale`

Female voice.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Discussion

These constants are used in the `gender` field of the `VoiceDescription` (page 55) structure.

Declared In

`SpeechSynthesis.h`

Stop Speech Locations

Locations that indicate where speech should be paused or stopped.

```
enum {
    kImmediate = 0,
    kEndOfWord = 1,
    kEndOfSentence = 2
};
```

Constants

`kImmediate`

Speech should be paused or stopped immediately.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kEndOfWord`

Speech should be paused or stopped at the end of the word.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kEndOfSentence`

Speech should be paused or stopped at the end of the sentence.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Discussion

See the functions `PauseSpeechAt` (page 27) and `StopSpeechAt` (page 36) for more information.

Declared In

`SpeechSynthesis.h`

Speech Synthesis Manager Operating System Types

The `OSType` definitions used by the Speech Synthesis Manager.

```
enum {
    kTextToSpeechSynthType = 'ttsc',
    kTextToSpeechVoiceType = 'ttvd',
    kTextToSpeechVoiceFileType = 'ttvf',
    kTextToSpeechVoiceBundleType = 'ttvb'
};
```

Constants

`kTextToSpeechSynthType`

The type of a synthesizer component.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kTextToSpeechVoiceType`

The type of a voice resource.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kTextToSpeechVoiceFileType`

The type of a voice file. Typically. files containing a single voice are of type `kTextToSpeechVoiceFileType`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kTextToSpeechVoiceBundleType`

The type of a voice bundle file. Typically, files containing multiple voices are of type `kTextToSpeechVoiceBundleType`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Speech-Channel Modes

The available text-processing and number-processing modes for a speech channel.

```
enum {
    modeText = 'TEXT',
    modePhonemes = 'PHON',
    modeNormal = 'NORM',
    modeLiteral = 'LTRL'
};
```

Constants

`modeText`

Indicates that the speech channel is in text-processing mode.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`modePhonemes`

Indicates that the speech channel is in phoneme-processing mode. When in phoneme-processing mode, a text buffer is interpreted to be a series of characters representing various phonemes and prosodic controls.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`modeNormal`

Indicates that the synthesizer assembles digits into numbers (so that "12" is spoken as "twelve").

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`modeLiteral`

Indicates that each digit is spoken literally (so that "12" is spoken as "one, two").

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Speech-Channel Modes for Core Foundation-based Functions

The available text-processing and number-processing modes for a speech channel.

```
CFStringRef kSpeechModeText = CFSTR("TEXT");
CFStringRef kSpeechModePhoneme = CFSTR("PHON");
CFStringRef kSpeechModeNormal = CFSTR("NORM");
CFStringRef kSpeechModeLiteral = CFSTR("LTRL");
```

Constants

`kSpeechModeText`

Indicates that the speech channel is in text-processing mode.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechModePhoneme`

Indicates that the speech channel is in phoneme-processing mode. When in phoneme-processing mode, a text buffer is interpreted to be a series of characters representing various phonemes and prosodic controls.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechModeNormal`

Indicates that the synthesizer assembles digits into numbers (so that "12" is spoken as "twelve").

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechModeLiteral`

Indicates that each digit is spoken literally (so that "12" is spoken as "one, two").

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In
`SpeechSynthesis.h`

Voice Information Selectors

The types of voice data that can be requested by the `GetVoiceInfo` function.

```
enum {
    soVoiceDescription = 'info',
    soVoiceFile = 'fref'
};
```

Constants

`soVoiceDescription`

Get basic voice information.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soVoiceFile`

Get voice file reference information.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Declared In
`SpeechSynthesis.h`

Speech-Channel Information Constants

Selectors that can be passed to the `GetSpeechInfo` or `SetSpeechInfo` functions.

```
enum {
    soStatus = 'stat',
    soErrors = 'erro',
    soInputMode = 'inpt',
    soCharacterMode = 'char',
    soNumberMode = 'nmbr',
    soRate = 'rate',
    soPitchBase = 'pbas',
    soPitchMod = 'pmod',
    soVolume = 'volm',
    soSynthType = 'vers',
    soRecentSync = 'sync',
    soPhonemeSymbols = 'phsy',
    soCurrentVoice = 'cvox',
    soCommandDelimiter = 'dlim',
    soReset = 'rset',
    soCurrentA5 = 'myA5',
    soRefCon = 'refc',
    soTextDoneCallBack = 'tdcb',
    soSpeechDoneCallBack = 'sdcb',
    soSyncCallBack = 'sycb',
    soErrorCallBack = 'ercb',
    soPhonemeCallBack = 'phcb',
    soWordCallBack = 'wdcb',
    soSynthExtension = 'xtnd',
    soSoundOutput = 'sndo',
    soOutputToFileWithCFURL = 'opaf'
};
```

Constants

`soStatus`

Get a speech status information structure for the speech channel. The `speechInfo` parameter is a pointer to a speech status information structure, described in `SpeechStatusInfo` (page 52).

This selector works with the `GetSpeechInfo` (page 16) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soErrors`

Get saved error information for the speech channel and clear its error registers. This selector lets you poll for various run-time errors that occur during speaking, such as the detection of badly formed embedded commands. Errors returned directly by Speech Synthesis Manager functions are not reported here. If your application defines an error callback function, the callback should use the `soErrors` selector to obtain error information. The `speechInfo` parameter is a pointer to a speech error information structure, described in `SpeechErrorInfo` (page 51).

This selector works with the `GetSpeechInfo` (page 16) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soInputMode`

Get or set the speech channel's current text-processing mode. The returned value specifies whether the channel is currently in text input mode or phoneme input mode. The `speechInfo` parameter is a pointer to a variable of type `OSType`, which specifies a text-processing mode. The constants `modeText` and `modePhonemes` specify the available text-processing modes.

The `modeText` constant indicates that the speech channel is in text-processing mode. The `modePhonemes` constant indicates that the speech channel is in phoneme-processing mode. When in phoneme-processing mode, a text buffer is interpreted to be a series of characters representing various phonemes and prosodic controls. Some synthesizers might support additional input-processing modes and define constants for these modes.

This selector works with both the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soCharacterMode`

Get or set the speech channel's character-processing mode. Two constants are currently defined for the processing mode, `modeNormal` and `modeLiteral`. When the character-processing mode is `modeNormal`, input characters are spoken as you would expect to hear them. When the mode is `modeLiteral`, each character is spoken literally, so that the word "cat" would be spoken "C–A–T". The `speechInfo` parameter points to a variable of type `OSType`, which is the character-processing mode.

This selector works with both the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soNumberMode`

Get or set the speech channel's current number-processing mode. Two `OSType` constants are currently defined, `modeNormal` and `modeLiteral`. When the number-processing mode is `modeNormal`, the synthesizer assembles digits into numbers (so that 12 is spoken as "twelve"). When the mode is `modeLiteral`, each digit is spoken literally (so that 12 is spoken as "one, two"). The `speechInfo` parameter is a pointer to a variable of type `OSType`, which specifies the number-processing mode.

This selector works with both the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soRate`

Get or set a speech channel's speech rate. The `speechInfo` parameter is a pointer to a variable of type `Fixed`. The possible range of speech rates is from 0.000 to 65535.65535. The range of supported rates is not predefined by the Speech Synthesis Manager; each speech synthesizer provides its own range of speech rates. Average human speech occurs at a rate of 180 to 220 words per minute.

This selector works with both the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soPitchBase`
: Get or set the speech channel's baseline speech pitch. This selector is intended for use by the Speech Synthesis Manager; ordinarily, an application uses the `GetSpeechPitch` (page 17) and `SetSpeechPitch` (page 29) functions. The `speechInfo` parameter is a pointer to a variable of type `Fixed`.

  This selector works with both the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechSynthesis.h`.

`soPitchMod`
: Get or set a speech channel's pitch modulation. The `speechInfo` parameter is a pointer to a variable of type `Fixed`. Pitch modulation is also expressed as a fixed-point value in the range of 0.000 to 127.000. These values correspond to MIDI note values, where 60.000 is equal to middle C on a piano scale. The most useful speech pitches fall in the range of 40.000 to 55.000. A pitch modulation value of 0.000 corresponds to a monotone in which all speech is generated at the frequency corresponding to the speech pitch. Given a speech pitch value of 46.000, a pitch modulation of 2.000 would mean that the widest possible range of pitches corresponding to the actual frequency of generated text would be 44.000 to 48.000.

  This selector works with both the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechSynthesis.h`.

`soVolume`
: Get or set the speech volume for a speech channel. The `speechInfo` parameter is a pointer to a variable of type `Fixed`. Volumes are expressed in fixed-point units ranging from 0.0 through 1.0. A value of 0.0 corresponds to silence, and a value of 1.0 corresponds to the maximum possible volume. Volume units lie on a scale that is linear with amplitude or voltage. A doubling of perceived loudness corresponds to a doubling of the volume.

  This selector works with both the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechSynthesis.h`.

`soSynthType`
: Get a speech version information structure for the speech synthesizer being used on the specified speech channel. The `speechInfo` parameter is a pointer to a speech version information structure, described in `SpeechVersionInfo` (page 54).

  This selector works with the `GetSpeechInfo` (page 16) function.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechSynthesis.h`.

`soRecentSync`
: Get the message code for the most recently encountered synchronization command. If no synchronization command has been encountered, 0 is returned. The `speechInfo` parameter is a pointer to a variable of type `OSType`.

  This selector works with the `GetSpeechInfo` (page 16) function.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechSynthesis.h`.

`soPhonemeSymbols`

Get a list of phoneme symbols and example words defined for the speech channel's synthesizer. Your application might use this information to show the user what symbols to use when entering phonemic text directly. The `speechInfo` parameter is a pointer to a variable of type Handle that, on exit from the `GetSpeechInfo` function, is a handle to a phoneme descriptor structure, described in `PhonemeDescriptor` (page 49).

This selector works with the `GetSpeechInfo` (page 16) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soCurrentVoice`

Set the current voice on the current speech channel to the specified voice. The `speechInfo` parameter is a pointer to a voice specification structure. Your application should create the structure by calling `MakeVoiceSpec` (page 23). `SetSpeechInfo` will return an `incompatibleVoice` error if the specified voice is incompatible with the speech synthesizer associated with the speech channel. If you have a speech channel open using a voice from a particular synthesizer and you try to switch to a voice that works with a different synthesizer, you receive an `incompatibleVoice` error. You need to create a new channel to use with the new voice.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soCommandDelimiter`

Set the embedded speech command delimiter characters to be used for the speech channel. By default the opening delimiter is "`[[`" and the closing delimiter is "`]]`". Your application might need to change these delimiters temporarily if those character sequences occur naturally in a text buffer that is to be spoken. Your application can also disable embedded command processing by passing empty delimiters (2 `NULL` bytes). The `speechInfo` parameter is a pointer to a delimiter information structure, described in `DelimiterInfo` (page 48).

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soReset`

Set a speech channel back to its default state. For example, speech pitch and speech rate are set to default values. The `speechInfo` parameter should be set to `NULL`.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soCurrentA5`

Set the value that the Speech Synthesis Manager assigns to the A5 register before invoking any application-defined callback functions for the speech channel. The A5 register must be set correctly if the callback functions are to be able to access application global variables. The `speechInfo` parameter should be set to the pointer contained in the A5 register at a time when the application is not executing interrupt code or to `NULL` if your application wishes to clear a value previously set with the `soCurrentA5` selector.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soRefCon`

Set a speech channel's reference constant value. The reference constant value is passed to application-defined callback functions and might contain any value convenient for the application. The `speechInfo` parameter is a long integer containing the reference constant value. In contrast with other selectors, this selector does not require that the `speechInfo` parameter's value be a pointer value. Typically, however, an application does use this selector to pass a pointer or handle value to callback functions.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soTextDoneCallBack`

Set the callback function to be called when the Speech Synthesis Manager has finished processing speech being generated on the speech channel. The `speechInfo` parameter is a pointer to an application-defined text-done callback function, whose syntax is described in `SpeechTextDoneProcPtr` (page 45). Passing `NULL` in `speechInfo` disables the text-done callback function.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSpeechDoneCallBack`

Set the callback function to be called when the Speech Synthesis Manager has finished generating speech on the speech channel. The `speechInfo` parameter is a pointer to an application-defined speech-done callback function, whose syntax is described in `SpeechDoneProcPtr` (page 40). Passing `NULL` in `speechInfo` disables the speech-done callback function.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSyncCallBack`

Set the callback function to be called when the Speech Synthesis Manager encounters a synchronization command within an embedded speech command in text being processed on the speech channel. The `speechInfo` parameter is a pointer to an application-defined synchronization callback function, whose syntax is described in `SpeechSyncProcPtr` (page 44). Passing `NULL` in `speechInfo` disables the synchronization callback function.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soErrorCallBack`

Set the callback function to be called when an error is encountered during the processing of an embedded command. The callback function might also be called if other conditions (such as insufficient memory) arise during the speech conversion process. When a Speech Synthesis Manager function returns an error directly, the error callback function is not called. The callback function is passed information about the most recent error; it can determine information about the oldest pending error by using the speech information selector `soErrors`. The `speechInfo` parameter is a pointer to an application-defined error callback function. Passing `NULL` in `speechInfo` disables the error callback function, `SpeechErrorProcPtr` (page 42).

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soPhonemeCallBack`

Set the callback function to be called every time the Speech Synthesis Manager is about to generate a phoneme on the speech channel. The `speechInfo` parameter is a pointer to an application-defined phoneme callback function, whose syntax is described in `SpeechPhonemeProcPtr` (page 43). Passing `NULL` in `speechInfo` disables the phoneme callback function.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soWordCallBack`

Set the callback function to be called every time the Speech Synthesis Manager is about to generate a word on the speech channel. The `speechInfo` parameter is a pointer to an application-defined word callback function, whose syntax is described in `SpeechWordProcPtr` (page 47). Passing `NULL` in `speechInfo` disables the word callback function.

This selector works with the `SetSpeechInfo` (page 28) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSynthExtension`

Get or set synthesizer-specific information or settings. The `speechInfo` parameter is a pointer to a speech extension data structure, described in `SpeechXtndData` (page 55). Your application should set the `synthCreator` field of this structure before calling `GetSpeechInfo` (page 16) or `SetSpeechInfo` (page 28). Ordinarily, your application must pass additional information to the synthesizer in the `synthData` field.

This selector works with both the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSoundOutput`

Get or set the speech channel's current output channel. (**Deprecated.** Use `soOutputToFileWithCFURL` (page 70) instead.)

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soOutputToFileWithCFURL`
: Pass a `CFURLRef` to write to this file, `NULL` to generate sound.

  This selector works with the `SetSpeechInfo` (page 28) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

Discussion
See the `GetSpeechInfo` (page 16) and `SetSpeechInfo` (page 28) functions.

Declared In
`SpeechSynthesis.h`

Speech-Channel Properties

Properties used with `CopySpeechProperty` (page 12) or `SetSpeechProperty` (page 29) to get or set the characteristics of a speech channel.

```
CFStringRef kSpeechStatusProperty = CFSTR("stat");
CFStringRef kSpeechErrorsProperty = CFSTR("erro");
CFStringRef kSpeechInputModeProperty = CFSTR("inpt");
CFStringRef kSpeechCharacterModeProperty = CFSTR("char");
CFStringRef kSpeechNumberModeProperty = CFSTR("nmbr");
CFStringRef kSpeechRateProperty = CFSTR("rate");
CFStringRef kSpeechPitchBaseProperty = CFSTR("pbas");
CFStringRef kSpeechPitchModProperty = CFSTR("pmod");
CFStringRef kSpeechVolumeProperty = CFSTR("volm");
CFStringRef kSpeechSynthesizerInfoProperty = CFSTR("vers");
CFStringRef kSpeechRecentSyncProperty = CFSTR("sync");
CFStringRef kSpeechPhonemeSymbolsProperty = CFSTR("phsy");
CFStringRef kSpeechCurrentVoiceProperty = CFSTR("cvox");
CFStringRef kSpeechCommandDelimiterProperty = CFSTR("dlim");
CFStringRef kSpeechResetProperty = CFSTR("rset");
CFStringRef kSpeechOutputToFileURLProperty = CFSTR("opaf");
CFStringRef kSpeechRefConProperty = CFSTR("refc");
CFStringRef kSpeechTextDoneCallBack = CFSTR("tdcb");
CFStringRef kSpeechSpeechDoneCallBack = CFSTR("sdcb");
CFStringRef kSpeechSyncCallBack = CFSTR("sycb");
CFStringRef kSpeechPhonemeCallBack = CFSTR("phcb");
CFStringRef kSpeechErrorCFCallBack = CFSTR("eccb");
CFStringRef kSpeechWordCFCallBack = CFSTR("wccb");
```

Constants
`kSpeechStatusProperty`
: Get speech-status information for the speech channel.

  The value associated with this property is a `CFDictionary` object that contains speech-status information for the speech channel. See "`Speech Status Keys`" (page 77) for a description of the keys present in the dictionary.

  This property works with the `CopySpeechProperty` (page 12) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechErrorsProperty`

Get speech-error information for the speech channel.

The value associated with this property is a `CFDictionary` object that contains speech-error information. See "`Speech Error Keys`" (page 78) for a description of the keys present in the dictionary.

This property lets you get information about various run-time errors that occur during speaking, such as the detection of badly formed embedded commands. Errors returned directly by the Speech Synthesis Manager are not reported here. If your application defines an error callback function, the function can use this property to get error information.

This property works with the `CopySpeechProperty` (page 12) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechInputModeProperty`

Get or set the speech channel's current text-processing mode.

The value associated with this property is a `CFString` object that specifies whether the channel is currently in text input mode or phoneme input mode. The constants `kSpeechModeText` and `kSpeechModePhoneme` (defined in "`Speech-Channel Modes for Core Foundation-based Functions`" (page 62)) are the possible values of this string.

When in phoneme-processing mode, a text string is interpreted to be a series of characters representing various phonemes and prosodic controls. Some synthesizers might support additional input-processing modes and define constants for these modes.

This property works with the `CopySpeechProperty` (page 12) and `SetSpeechProperty` (page 29) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechCharacterModeProperty`

Get or set the speech channel's current character-processing mode.

The value associated with this property is a `CFString` object that specifies whether the speech channel is currently in normal or literal character-processing mode. The constants `kSpeechModeNormal` and `kSpeechModeLiteral` (defined in "`Speech-Channel Modes for Core Foundation-based Functions`" (page 62)) are the possible values of this string.

When the character-processing mode is `kSpeechModeNormal`, input characters are spoken as you would expect to hear them. When the mode is `kSpeechModeLiteral`, each character is spoken literally, so that the word "cat" is spoken "C–A–T".

This property works with the `CopySpeechProperty` (page 12) and `SetSpeechProperty` (page 29) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechNumberModeProperty`
: Get or set the speech channel's current number-processing mode.

  The value associated with this property is a `CFString` object that specifies whether the speech channel is currently in normal or literal number-processing mode. The constants `kSpeechModeNormal` and `kSpeechModeLiteral` (defined in "`Speech-Channel Modes for Core Foundation-based Functions`" (page 62)) are the possible values of this string.

  When the number-processing mode is `kSpeechModeNormal`, the synthesizer assembles digits into numbers (so that "12" is spoken as "twelve"). When the mode is `kSpeechModeLiteral`, each digit is spoken literally (so that "12" is spoken as "one, two").

  This property works with the `CopySpeechProperty` (page 12) and `SetSpeechProperty` (page 29) functions.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechRateProperty`
: Get or set a speech channel's speech rate.

  The value associated with this property is a `CFNumber` object that specifies the speech channel's speaking rate.

  The range of supported rates is not predefined by the Speech Synthesis Manager; each speech synthesizer provides its own range of speech rates. Average human speech occurs at a rate of 180 to 220 words per minute.

  This property works with the `CopySpeechProperty` (page 12) and `SetSpeechProperty` (page 29) functions.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechPitchBaseProperty`
: Get or set the speech channel's baseline speech pitch.

  The value associated with this property is a `CFNumber` object that specifies the speech channel's baseline speech pitch.

  Typical voice frequencies range from around 90 hertz for a low-pitched male voice to perhaps 300 hertz for a high-pitched child's voice. These frequencies correspond to approximate pitch values in the ranges of 30.000 to 40.000 and 55.000 to 65.000, respectively.

  This property works with the `CopySpeechProperty` (page 12) and `SetSpeechProperty` (page 29) functions.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechPitchModProperty`
: Get or set a speech channel's pitch modulation.

  The value associated with this property is a `CFNumber` object that specifies the speech channel's pitch modulation.

  Pitch modulation is also expressed as a floating-point value in the range of 0.000 to 127.000. These values correspond to MIDI note values, where 60.000 is equal to middle C on a piano scale. The most useful speech pitches fall in the range of 40.000 to 55.000. A pitch modulation value of 0.000 corresponds to a monotone in which all speech is generated at the frequency corresponding to the speech pitch. Given a speech pitch value of 46.000, a pitch modulation of 2.000 would mean that the widest possible range of pitches corresponding to the actual frequency of generated text would be 44.000 to 48.000.

  This property works with the `CopySpeechProperty` (page 12) and `SetSpeechProperty` (page 29) functions.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechVolumeProperty`
: Get or set the speech volume for a speech channel.

  The value associated with this property is a `CFNumber` object that specifies the speech channel's speech volume.

  Volumes are expressed in floating-point values ranging from 0.0 through 1.0. A value of 0.0 corresponds to silence, and a value of 1.0 corresponds to the maximum possible volume. Volume units lie on a scale that is linear with amplitude or voltage. A doubling of perceived loudness corresponds to a doubling of the volume.

  This property works with the `CopySpeechProperty` (page 12) and `SetSpeechProperty` (page 29) functions.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechSynthesizerInfoProperty`
: Get information about the speech synthesizer being used on the specified speech channel.

  The value associated with this property is a `CFDictionary` object that contains information about the speech synthesizer being used on the specified speech channel. See "`Speech Synthesizer Information Keys`" (page 79) for a description of the keys present in the dictionary.

  This property works with the `CopySpeechProperty` (page 12) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechRecentSyncProperty`
: Get the message code for the most recently encountered synchronization command.

  The value associated with this property is a `CFNumber` object that specifies the most recently encountered synchronization command. This property works with the `CopySpeechProperty` (page 12) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechPhonemeSymbolsProperty`
: Get a list of phoneme symbols and example words defined for the speech channel's synthesizer.

  The value associated with this property is a `CFDictionary` object that contains the phoneme symbols and example words defined for the current synthesizer. Your application might use this information to show the user what symbols to use when entering phonemic text directly. See "`Phoneme Symbols Keys`" (page 80) for a description of the keys present in the dictionary.

  This property works with the `CopySpeechProperty` (page 12) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechCurrentVoiceProperty`
: Set the current voice on the current speech channel to the specified voice.

  The value associated with this property is a `CFDictionary` object that contains the phoneme symbols and example words defined for the current synthesizer. Your application might use this information to show the user what symbols to use when entering phonemic text directly. See "`Phoneme Symbols Keys`" (page 80) for the keys you can use to specify values in this dictionary.

  This property works with the `SetSpeechProperty` (page 29) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechCommandDelimiterProperty`
: Set the embedded speech command delimiter characters to be used for the speech channel.

  By default, the opening delimiter is "`[[`" and the closing delimiter is "`]]`". Your application might need to change these delimiters temporarily if those character sequences occur naturally in a text buffer that is to be spoken. Your application can also disable embedded command processing by passing empty delimiters (as empty strings). The value associated with this property is a `CFDictionary` object that contains the delimiter information. See "`Command Delimiter Keys`" (page 81) for the keys you can use to specify values in this dictionary.

  This property works with the `SetSpeechProperty` (page 29) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechResetProperty`
: Set a speech channel back to its default state.

  You can use this function to, for example, set speech pitch and speech rate to default values. There is no value associated with this property; to reset the channel to its default state, set the string to `NULL`.

  This property works with the `SetSpeechProperty` (page 29) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechOutputToFileURLProperty`
: Set the speech output destination to a file or to the computer's speakers.

  The value associated with this property is a `CFURL` object. To write the speech output to a file, use the file's `CFURLRef`; to generate the sound through the computer's speakers, use `NULL`.

  This property works with the `SetSpeechProperty` (page 29) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechRefConProperty`

Set a speech channel's reference constant value.

The reference constant value is passed to application-defined callback functions and might contain any value convenient for the application. The value associated with this property is a `CFNumber` object that contains an integer value. For example, an application might set the value of the `CFNumber` object to an address in memory that contains a reference to an object or a pointer to a function.

This property works with the `SetSpeechProperty` (page 29) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechTextDoneCallBack`

Set the callback function to be called when the Speech Synthesis Manager has finished processing speech being generated on the speech channel.

The value associated with this property is a `CFNumber` object whose value is a pointer to an application-defined text-done callback function, whose syntax is described in `SpeechTextDoneProcPtr` (page 45). Passing a `CFNumber` object that contains the value `NULL` disables the text-done callback function.

This property works with the `SetSpeechProperty` (page 29) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechSpeechDoneCallBack`

Set the callback function to be called when the Speech Synthesis Manager has finished generating speech on the speech channel.

The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined speech-done callback function, whose syntax is described in `SpeechDoneProcPtr` (page 40). Passing `NULL` for the value of this property disables the speech-done callback function.

This property works with the `SetSpeechProperty` (page 29) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechSyncCallBack`

Set the callback function to be called when the Speech Synthesis Manager encounters a synchronization command within an embedded speech command in text being processed on the speech channel.

The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined synchronization callback function, whose syntax is described in `SpeechSyncProcPtr` (page 44). Passing a `CFNumber` object that contains the value `NULL` for the value of this property disables the synchronization callback function.

This property works with the `SetSpeechProperty` (page 29) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechPhonemeCallBack`
: Set the callback function to be called every time the Speech Synthesis Manager is about to generate a phoneme on the speech channel.

  The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined phoneme callback function, whose syntax is described in `SpeechPhonemeProcPtr` (page 43). Passing a `CFNumber` object that contains the value `NULL` for the value of this property disables the phoneme callback function.

  This property works with the `SetSpeechProperty` (page 29) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechErrorCFCallBack`
: Set the callback function to be called when an error is encountered during the processing of an embedded command.

  When a Speech Synthesis Manager function returns an error directly, the error callback function is not called. The callback function is passed information about the most recent error; it can determine information about the oldest pending error by using the speech information property `kSpeechErrorsProperty`. The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined error callback function, whose syntax is described in `SpeechErrorCFProcPtr` (page 41). Passing a `CFNumber` object that contains the value `NULL` for the value of this property disables the error callback function.

  This property works with the `SetSpeechProperty` (page 29) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechWordCFCallBack`
: Set the callback function to be called every time the Speech Synthesis Manager is about to generate a word on the speech channel.

  The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined word callback function, whose syntax is described in `SpeechWordCFProcPtr` (page 46). Passing a `CFNumber` object that contains the value `NULL` for the value of this property disables the word callback function.

  This property works with the `SetSpeechProperty` (page 29) function.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

Synthesizer Option Keys

Keys used to specify synthesizer options.

```
CFStringRef kSpeechNoEndingProsody = CFSTR("NoEndingProsody");
CFStringRef kSpeechNoSpeechInterrupt = CFSTR("NoSpeechInterrupt");
CFStringRef kSpeechPreflightThenPause = CFSTR("PreflightThenPause");
```

Constants

`kSpeechNoEndingProsody`

Disable prosody at the end of sentences.

The `kSpeechNoEndingProsody` key is used to indicate whether the speech synthesizer should automatically apply ending prosody, which is the speech tone and cadence that normally occur at the end of a sentence. When the key is not specified, ending prosody is applied to the speech at the end of `aString`. This behavior can be disabled by specifying the `kSpeechNoEndingProsody` key in the `options` dictionary.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechNoSpeechInterrupt`

Do not interrupt current speech.

The `kSpeechNoSpeechInterrupt` key is used to control the behavior of `SpeakCFString` (page 31) when it is called on a speech channel that is busy. When `kSpeechNoSpeechInterrupt` is not specified in the `options` dictionary, `SpeakCFString` immediately interrupts the speech currently being produced on the specified speech channel and the new `aString` text is spoken. When `kSpeechNoSpeechInterrupt` is specified in the `options` dictionary, the request to speak on a speech channel that is already busy causes the new `aString` text to be ignored and the `synthNotReady` error to be returned. As soon as the prior string has been fully processed, the new string is then spoken.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechPreflightThenPause`

Compute speech without generating it.

The `kSpeechPreflightThenPause` key is used to minimize the latency experienced when the speech synthesizer is attempting to speak.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

Declared In

`SpeechSynthesis.h`

Speech Status Keys

Keys used with the `kSpeechStatusProperty` property to specify the status of the speech channel.

```
CFStringRef kSpeechStatusOutputBusy = CFSTR("OutputBusy");
CFStringRef kSpeechStatusOutputPaused = CFSTR("OutputPaused");
CFStringRef kSpeechStatusNumberOfCharactersLeft = CFSTR("NumberOfCharactersLeft");
CFStringRef kSpeechStatusPhonemeCode = CFSTR("PhonemeCode");
```

Constants

`kSpeechStatusOutputBusy`

Indicates whether the speech channel is currently producing speech.

A speech channel is considered to be producing speech even at some times when no audio data is being produced through the computer's speaker. This occurs, for example, when the Speech Synthesis Manager is processing input, but has not yet initiated speech or when speech output is paused.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechStatusOutputPaused`

Indicates whether speech output in the speech channel has been paused by a call to the `PauseSpeechAt` (page 27) function.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechStatusNumberOfCharactersLeft`

The number of characters left in the input string of text.

When the value of this constant is zero, you can destroy the input string.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechStatusPhonemeCode`

The opcode for the phoneme that the speech channel is currently processing.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Speech Error Keys

Keys used with the `kSpeechErrorsProperty` property to describe errors encountered during speech processing and production.

```
CFStringRef kSpeechErrorCount = CFSTR("Count");
CFStringRef kSpeechErrorOldest = CFSTR("OldestCode");
CFStringRef kSpeechErrorOldestCharacterOffset = CFSTR("OldestCharacterOffset");
CFStringRef kSpeechErrorNewest = CFSTR("NewestCode");
CFStringRef kSpeechErrorNewestCharacterOffset = CFSTR("NewestCharacterOffset");
```

Constants

`kSpeechErrorCount`

The number of errors that have occurred in processing the current text string, since the last call to the `CopySpeechProperty` (page 12) function with the `kSpeechErrorsProperty` property.

Using the `kSpeechErrorOldest` keys and the `kSpeechErrorNewest` keys, you can get information about the oldest and most recent errors that occurred since the last call to `CopySpeechProperty` (page 12), but you cannot get information about any intervening errors.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechErrorOldest`

The error code of the first error that occurred since the last call to the `CopySpeechProperty` (page 12) function with the `kSpeechErrorsProperty` property.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechErrorOldestCharacterOffset`

The position in the text string of the first error that occurred since the last call to the `CopySpeechProperty` (page 12) function with the `kSpeechErrorsProperty` property.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechErrorNewest`

The error code of the most recent error that occurred since the last call to the `CopySpeechProperty` (page 12) function with the `kSpeechErrorsProperty` property.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechErrorNewestCharacterOffset`

The position in the text string of the most recent error that occurred since the last call to the `CopySpeechProperty` (page 12) function with the `kSpeechErrorsProperty` property.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Speech Synthesizer Information Keys

Keys used with the `kSpeechSynthesizerInfoProperty` property to get information about the synthesizer.

```
CFStringRef kSpeechSynthesizerInfoIdentifier = CFSTR("Identifier");
CFStringRef kSpeechSynthesizerInfoVersion = CFSTR("Version");
```

Constants

`kSpeechSynthesizerInfoIdentifier`
: The identifier of the speech synthesizer.

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

`kSpeechSynthesizerInfoVersion`
: The version of the speech synthesizer.

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Phoneme Symbols Keys

Keys used with the `kSpeechPhonemeSymbolsProperty` property to provide information about the phoneme being processed.

```
CFStringRef kSpeechPhonemeInfoOpcode = CFSTR("Opcode");
CFStringRef kSpeechPhonemeInfoSymbol = CFSTR("Symbol");
CFStringRef kSpeechPhonemeInfoExample = CFSTR("Example");
CFStringRef kSpeechPhonemeInfoHiliteStart = CFSTR("HiliteStart");
CFStringRef kSpeechPhonemeInfoHiliteEnd = CFSTR("HiliteEnd");
```

Constants

`kSpeechPhonemeInfoOpcode`
: The opcode of the phoneme.

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

`kSpeechPhonemeInfoSymbol`
: The symbol used to represent the phoneme.

  The symbol does not necessarily have a phonetic connection to the phoneme, but might simply be an abstract textual representation of it.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechPhonemeInfoExample`
: An example word that illustrates the use of the phoneme.

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

`kSpeechPhonemeInfoHiliteStart`
: The character offset into the example word that identifies the location of the beginning of the phoneme.

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

`kSpeechPhonemeInfoHiliteEnd`
: The character offset into the example word that identifies the location of the end of the phoneme.

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

Declared In
`SpeechSynthesis.h`

Current Voice Keys

Keys used with the `kSpeechCurrentVoiceProperty` property to specify information about the current voice.

```
CFStringRef kSpeechVoiceCreator = CFSTR("Creator");
CFStringRef kSpeechVoiceID = CFSTR("ID");
```

Constants

`kSpeechVoiceCreator`
: The synthesizer that is required to use the voice.

  Declared in `SpeechSynthesis.h`.

  Available in Mac OS X v10.5 and later.

`kSpeechVoiceID`
: The voice ID of the voice for the synthesizer (every voice on a synthesizer has a unique ID).

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

Declared In
`SpeechSynthesis.h`

Command Delimiter Keys

Keys used with the `kSpeechCommandDelimiterProperty` property to specify information about the command delimiter strings.

```
CFStringRef kSpeechCommandPrefix = CFSTR("Prefix");
CFStringRef kSpeechCommandSuffix = CFSTR("Suffix");
```

Constants

`kSpeechCommandPrefix`
: The command delimiter string that prefixes a command (by default, this is `[[`).

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

`kSpeechCommandSuffix`
: The command delimiter string that suffixes a command (by default, this is `]]`).

  Available in Mac OS X v10.5 and later.

  Declared in `SpeechSynthesis.h`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

Speech Dictionary Keys

Keys used in a speech dictionary to override the synthesizer's default pronunciation of a word.

```
CFStringRef kSpeechDictionaryLocaleIdentifier = CFSTR("LocaleIdentifier");
CFStringRef kSpeechDictionaryModificationDate = CFSTR("ModificationDate");
CFStringRef kSpeechDictionaryPronunciations = CFSTR("Pronunciations");
CFStringRef kSpeechDictionaryAbbreviations = CFSTR("Abbreviations");
CFStringRef kSpeechDictionaryEntrySpelling = CFSTR("Spelling");
CFStringRef kSpeechDictionaryEntryPhonemes = CFSTR("Phonemes");
```

Constants

`kSpeechDictionaryLocaleIdentifier`
The locale associated with the pronunciation.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechDictionaryModificationDate`
The date the dictionary was last modified.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechDictionaryPronunciations`
The set of custom pronunciations.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechDictionaryAbbreviations`
The set of custom pronunciations for abbreviations.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechDictionaryEntrySpelling`
The spelling of an entry.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechDictionaryEntryPhonemes`
The phonemic representation of an entry.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Discussion
The keys in a speech dictionary can determine how a synthesizer pronounces a word. After you've created a speech dictionary, you register it with a speech channel with the `UseSpeechDictionary` (page 39) function.

Declared In
`SpeechSynthesis.h`

Error Callback User-Information String

Specifies the string to speak to the user when an error occurs.

```
CFStringRef kSpeechErrorCallbackSpokenString = CFSTR("SpokenString");
```

Constants

`kSpeechErrorCallbackSpokenString`

The string to speak to the user when an error occurs.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Result Codes

The most common result codes returned by Speech Synthesis Manager are listed below.

Result Code	Value	Description
`noSynthFound`	-240	Could not find the specified speech synthesizer Available in Mac OS X v10.0 and later.
`synthOpenFailed`	-241	Could not open another speech synthesizer channel Available in Mac OS X v10.0 and later.
`synthNotReady`	-242	Speech synthesizer is still busy speaking Available in Mac OS X v10.0 and later.
`bufTooSmall`	-243	Output buffer is too small to hold result Available in Mac OS X v10.0 and later.
`voiceNotFound`	-244	Voice resource not found Available in Mac OS X v10.0 and later.
`incompatibleVoice`	-245	Specified voice cannot be used with synthesizer Available in Mac OS X v10.0 and later.
`badDictFormat`	-246	Pronunciation dictionary format error Available in Mac OS X v10.0 and later.
`badInputText`	-247	Raw phoneme text contains invalid characters Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Speech Synthesis Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Document Revision History

This table describes the changes to *Speech Synthesis Manager Reference*.

Date	Notes
2009-04-08	Clarified the use of the script field in the voice description structure.
2009-02-04	Made minor corrections and additions.
2007-05-15	Updated for Mac OS X v10.5.
2003-02-11	Revised existing Speech Synthesis Manager result codes and added new ones.
	Fixed typographical errors.
	Updated formatting.

Index