

# SMAPI Reference

---

IBM ViaVoice(tm) Software Developer's Kit  
Version 1.7

---

Printed in the USA

Note: Before using this information and the product it supports, be sure to read the general information under [Appendix C \[Notices\]](#), page 367.

First Edition (December 1999)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

© International Business Machines Corporation 1999. All Rights Reserved. Note to U.S. Government Users-Documents related to restricted rights- Use, duplication or disclosure is subject to restrictions set forth in GS ADP Schedule Contract with IBM Corp.

# Table of Contents

<b>About This Document</b> .....	<b>1</b>
Who Should Read This Document? .....	1
Related Publications .....	1
<b>1 SMAPI Overview</b> .....	<b>3</b>
1.1 Format of the Function Call Descriptions .....	3
1.1.1 Reply Structure Functions by Message Type .....	3
1.1.2 Task Related Functions and Callbacks .....	4
1.1.3 Reply Structure-Related Functions and Callbacks ..	4
1.2 Naming Conventions .....	4
1.3 SMAPI Function Calls by Group .....	4
1.3.1 Attribute Handling Functions .....	5
1.3.2 Callback and Dispatching Functions .....	5
1.3.3 Data Access Functions .....	5
1.3.4 Connection Functions .....	7
1.3.5 Session Functions .....	7
1.3.6 Database Functions .....	7
1.3.7 Administrative Functions .....	8
1.3.8 Speech Recognition Engine State Functions .....	8
1.3.9 Vocabulary Functions .....	8
1.3.10 Audio Functions .....	9
1.3.11 Parallel Session API Calls .....	9
<b>2 SMAPI Starter Set APIs</b> .....	<b>11</b>
2.1 Starter Set SMAPIs for Command and Control .....	11
2.1.1 Establishing a Recognition Session .....	12
2.1.2 Setting Up and Enabling Vocabularies .....	12
2.1.3 Directing the Engine to Process Speech .....	12
2.1.4 Processing Recognized Commands .....	13
2.1.5 Disconnecting from the Engine .....	13
2.2 Starter Set SMAPIs for Dictation .....	13
2.2.1 Establishing a Recognition Session .....	14
2.2.2 Setting Up and Enabling Vocabularies .....	14
2.2.3 Directing the Engine to Process Speech .....	14
2.2.4 Processing Recognized Text .....	15
2.2.5 Correcting Errors .....	15
2.2.6 Disconnecting from the Engine .....	16

<b>3</b>	<b>Function Calls to the Speech Recognition Engine</b>	<b>17</b>
3.1	SmAddCallback	17
3.2	SmAddEnrollid	18
3.3	SmAddPronunciation	20
3.4	SmAddToVocab	23
3.5	SmAddUser	25
3.6	SmApiVersionCheck	27
3.7	SmAutoComplete	28
3.8	SmCancelPlayback	30
3.9	SmClose	32
3.10	SmConnect	33
3.11	SmCorrectText	36
3.12	SmCorrectTextEx	38
3.13	SmDefineGrammar	40
3.14	SmDefineVocab	43
3.15	SmDefineVocabEx	46
3.16	SmDetachSessions	50
3.17	SmDisableVocab	51
3.18	SmDiscardData	53
3.19	SmDiscardSpeechData	54
3.20	SmDiscardUtterance	56
3.21	SmDisconnect	57
3.22	SmDispatch	59
3.23	SmEnableVocab	60
3.24	SmEventNotify	62
3.25	SmEventNotifyEx	64
3.26	SmEventTime	66
3.27	SmHaltRecognizer	67
3.28	SmMicOff	69
3.29	SmMicOn	71
3.30	SmNewContext	73
3.31	SmNewContextEx	75
3.32	SmOpen	77
3.33	SmPlayMessage	78
3.34	SmPlayUtterance	80
3.35	SmPlayWords	82
3.36	SmQuery	85
3.37	SmQueryAddedWords	90
3.38	SmQueryAddedWordsEx	92
3.39	SmQueryAlternates	94
3.40	SmQueryBinary	96
3.41	SmQueryDefault	98
3.42	SmQueryEnabledVocabs	100
3.43	SmQueryEnrollIds	102
3.44	SmQueryLanguages	104
3.45	SmQueryPhraseAlternatives	105

3.46	SmQueryPronunciation	107
3.47	SmQueryPronunciationEx	109
3.48	SmQueryPronunciations	111
3.49	SmQueryPronunciationsEx	113
3.50	SmQueryScripts	115
3.51	SmQuerySessions	117
3.52	SmQuerySpeechData	118
3.53	SmQuerySpeechDataEx	119
3.54	SmQuerySpeechUserSize	121
3.55	SmQueryTasks	123
3.56	SmQueryTopics	125
3.57	SmQueryUserDefault	127
3.58	SmQueryUserInfo	129
3.59	SmQueryUsers	131
3.60	SmQueryUtterances	133
3.61	SmQueryVocabs	135
3.62	SmQueryWord	137
3.63	SmReceiveMsg	139
3.64	SmRecognizeNextWord	140
3.65	SmReleaseFocus	142
3.66	SmRemoveCallback	143
3.67	SmRemoveEnrollid	144
3.68	SmRemoveFromVocab	146
3.69	SmRemovePronunciation	148
3.70	SmRemoveUser	150
3.71	SmRequestFocus	152
3.72	SmRequestMicOff	154
3.73	SmRequestMicOn	155
3.74	SmRequestNewEnrollid	156
3.75	SmRequestScriptText	158
3.76	SmRestoreSpeechData	160
3.77	SmRestoreSpeechDataEx	162
3.78	SmRestoreSpeechUser	164
3.79	SmSaveSpeechData	166
3.80	SmSaveSpeechDataEx	168
3.81	SmSaveSpeechUser	171
3.82	SmSelectScript	173
3.83	SmSet	175
3.84	SmSetArg	182
3.85	SmSetBinary	183
3.86	SmSetDefault	185
3.87	SmSetDirectory	187
3.88	SmSetUserDefault	189
3.89	SmSetUserInfo	191
3.90	SmSetUtteranceNumber	193
3.91	SmUndefineVocab	194
3.92	SmWordCorrection	196

<b>4</b>	<b>Data Access Functions</b>	<b>199</b>
4.1	SmGetAlphabets	199
4.2	SmGetAlternates	200
4.3	SmGetAnnotations	201
4.4	SmGetApplication	203
4.5	SmGetApplications	204
4.6	SmGetAudioLevel	205
4.7	SmGetBinaryItemValue	206
4.8	SmGetCodePage	207
4.9	SmGetComment	208
4.10	SmGetConfidenceScores	209
4.11	SmGetDescriptions	210
4.12	SmGetEngineState	211
4.13	SmGetEnrollId	213
4.14	SmGetEnrollIds	214
4.15	SmGetEventId	215
4.16	SmGetEventOptions	216
4.17	SmGetExpectedRecordingSpace	217
4.18	SmGetExpectedTrainingSpace	218
4.19	SmGetFirmWords	219
4.20	SmGetFlags	220
4.21	SmGetFocusChangeReason	221
4.22	SmGetFocusState	222
4.23	SmGetFreeSpace	223
4.24	SmGetGrammarPath	224
4.25	SmGetIncrements	225
4.26	SmGetItemValue	226
4.27	SmGetLanguages	227
4.28	SmGetMicState	228
4.29	SmGetMsgName	229
4.30	SmGetMsgType	230
4.31	SmGetNameValue	231
4.32	SmGetNextAlternate	232
4.33	SmGetNumberProcessed	233
4.34	SmGetNumberRecorded	234
4.35	SmGetNumberRequired	235
4.36	SmGetNumberUtterances	236
4.37	SmGetNumberWordMsgs	237
4.38	SmGetOptions	238
4.39	SmGetPercentages	239
4.40	SmGetPhoneticPronunciations	240
4.41	SmGetPhraseScore	241
4.42	SmGetPhraseState	242
4.43	SmGetPreferred	243
4.44	SmGetPronunciations	244
4.45	SmGetRc	245
4.46	SmGetRcDescription	246

4.47	SmGetRcName .....	247
4.48	SmGetRequiredTrainingSpace .....	248
4.49	SmGetSampleRates .....	249
4.50	SmGetScriptFlags .....	250
4.51	SmGetScripts .....	251
4.52	SmGetService .....	252
4.53	SmGetSessionId .....	253
4.54	SmGetSeverity .....	254
4.55	SmGetSizes .....	255
4.56	SmGetSpeechDataArchive .....	256
4.57	SmGetSpeechDataOptions .....	257
4.58	SmGetSpeechDataSize .....	258
4.59	SmGetSpeechDataVersion .....	259
4.60	SmGetSpelling .....	260
4.61	SmGetSpellings .....	261
4.62	SmGetStates .....	262
4.63	SmGetStatus .....	263
4.64	SmGetTagOffset .....	265
4.65	SmGetTags .....	266
4.66	SmGetTask .....	267
4.67	SmGetTaskFlags .....	268
4.68	SmGetTasks .....	269
4.69	SmGetTimes .....	270
4.70	SmGetTopics .....	271
4.71	SmGetTrained .....	272
4.72	SmGetUserId .....	273
4.73	SmGetUserIds .....	274
4.74	SmGetUsers .....	275
4.75	SmGetUtteranceList .....	276
4.76	SmGetUtteranceNumber .....	277
4.77	SmGetVocabList .....	278
4.78	SmGetVocabName .....	279
4.79	SmGetVocabPath .....	280
4.80	SmGetVocWords .....	281
4.81	SmGetWords .....	282
4.82	SmGetWordTimes .....	283
4.83	SmReturnRc .....	285
4.84	SmReturnRcDescription .....	286
4.85	SmReturnRcName .....	287

## 5 Reply Message Structures and Callbacks .. 289

5.1	Reply Message Structures Received from the Speech Recognition Engine .....	289
5.2	Callbacks .....	299
5.3	Reply Structure Functions for Unsolicited Callbacks .....	303

<b>6</b>	<b>Data Types .....</b>	<b>305</b>
6.1	SmArg data type.....	305
6.2	SmArgVal data type.....	306
6.3	SmHandler data type.....	307
6.4	SM_ANNOTATION data type .....	308
6.5	SM_MSG data type .....	309
6.6	SM_VOCWORD data type.....	310
6.7	SM_WORD data type .....	311
<b>7</b>	<b>SMAPI Attributes.....</b>	<b>313</b>
7.1	System Dependent Definition for Argument Lists .....	313
7.2	Argument Attribute List.....	313
7.2.1	Application Information Attributes .....	313
7.2.2	Requested Services .....	313
7.2.3	Options Flags .....	314
7.2.4	External Notifier .....	314
7.2.5	User Definition Arguments .....	315
<b>8</b>	<b>SMAPI Grammar Compiler API Overview</b>	
	.....	<b>317</b>
8.1	Format of the Function Call Descriptions .....	317
8.2	Grammar Compiler API Function Calls.....	317
<b>9</b>	<b>SMAPI Grammar Compiler API Function</b>	
	<b>Calls.....</b>	<b>319</b>
9.1	VtAddArg.....	319
9.2	VtCompileGrammar.....	320
9.3	VtGetMessage .....	321
9.4	VtGetTranslation .....	322
9.5	VtLoadFSG .....	323
9.6	VtSetArg.....	324
9.7	VtUnloadFSG .....	325
<b>10</b>	<b>SMAPI Grammar Compiler Data Types</b>	
	.....	<b>327</b>
10.1	VtArg.....	327
<b>Appendix A SMAPI Return Codes, Messages,</b>		
<b>and Message Types .....</b>		<b>329</b>
A.1	SMAPI Return and Status Codes .....	329
A.1.1	SMAPI Return Codes .....	329
A.1.2	SMAPI Status Codes.....	332
A.2	SMAPI Message Types.....	333
A.3	SMAPI Message Explanations .....	337
A.4	SMAPI Logging .....	363



<b>Appendix B</b>	<b>Speech Recognition Engine Error</b>	
<b>Messages</b>		<b>365</b>
<b>Appendix C</b>	<b>Notices</b>	<b>367</b>
C.1	Trademarks	367
<b>Index</b>		<b>369</b>



## About This Document

This document provides detailed information on developing speech-aware applications using IBM ViaVoice(tm) Software Developer's Kit (SDK) and Speech Manager Application Programming Interfaces (SMAPI) from IBM. The ViaVoice SDK provides three sets of APIs: Speech Manager APIs (SMAPI), Dictation Macro APIs (DMAPI), and Grammar Compiler APIs. This document covers only SMAPI and the Grammar Compiler. Refer to the Speech Developer's Tools Guide for information about DMAPI.

## Who Should Read This Document?

Read this document if you are a software developer interested in writing applications using the ViaVoice SDK APIs.

## Related Publications

Refer to the following publications for additional programming, reference, and design information:

- SMAPI Developer's Guide

- SAPI Reference ( Windows developers only )

- ActiveX Developer's Guide ( Window developers only )

Refer to the following sources for additional programming, reference, and design information:

ViaVoice Developer's Corner website at:

- [http://www.ibm.com/viavoice/dev\\_home.html](http://www.ibm.com/viavoice/dev_home.html)

IBM ViaVoice SDK Web Channel at:

- <http://www.software.ibm.com/viavoice/subscribe.html>

OLE Automation Reference from Microsoft

- ( Windows developers only )



# 1 SMAPI Overview

This chapter describes the format of the function calls that are presented in Chapter 3 "Function Calls to the Speech Recognition Engine" and Chapter 4 "Data Access Functions". Chapter 3 contains the function calls that go directly to the speech recognition engine. Chapter 4 contains the function calls that do not interact with the engine; they provide local access to the logical contents of a message that has already been received.

This chapter also lists the calls by functional group.

## 1.1 Format of the Function Call Descriptions

The description of each function call contains the following information:

*Function Name*

The name of the function call.

*Purpose*

The purpose and description of the function call.

*Syntax*

The syntax of the function as declared in SMAPI.H.

*Parameters*

Definitions of the parameters.

*Return Values*

Return values are listed in two groups:

- Return values that are set by the SMAPI or as a result of an unsuccessful connection to the speech recognition engine. These values are always returned by the called function whether it was called synchronously or asynchronously.
- Return values that are set from within the speech recognition engine. These values are returned by the called function only if it was called synchronously. If it was called asynchronously, the speech-aware application must retrieve this type of return value from within the appropriate callback by using the SmGetRc function.

For detailed descriptions of SMAPI errors, reference <Pantone 2718 CVP>"SMAPI Return Codes and Messages" on page 353<Default Paragraph Font>.

### 1.1.1 Reply Structure Functions by Message Type

A list of all reply message structures that can be received from the speech recognition engine; and for each listed reply message structure, the SmGet functions that can extract data from it.

### 1.1.2 Task Related Functions and Callbacks

A list of functions and callbacks broadly related by task to this function. This is applicable to functions that do not return a reply structure. For example, task related groups include:

- Vocabulary Processing
- User Processing
- Application Initiation
- Speech Processing
- Application Termination

This section also lists related unsolicited callbacks.

### 1.1.3 Reply Structure-Related Functions and Callbacks

Clarifies when the SmGet functions can be called. If the function prototype SmXxx is listed as related for any SmGet function, then that function can be called as follows:

- Immediately after a synchronous SmXxx call
- From SmNXxxCallback after an asynchronous SmXxx call
- From the message reply structure after an asynchronous SmXxx call.

## 1.2 Naming Conventions

Use the following conventions when creating various IDs and names passed to the speech recognition engine:

- User IDs can consist of only lowercase alphanumeric characters plus the underscore and hyphen characters.
- Descriptions cannot contain a new line character.
- An ID must contain at least one character.

The following values, found in the file SMLIMITS.H, shows the maximum lengths of key variables:

```
#define SM_MAX_USERID_LEN 8
#define SM_MAX_ENROLLID_LEN 8
#define SM_MAX_TASKNAME_LEN 8
#define SM_MAX_SCRIPTNAME_LEN 8
```

## 1.3 SMAPI Function Calls by Group

The following information lists the SMAPI calls by functional group.

### 1.3.1 Attribute Handling Functions

These functions are implemented locally within the application's address space by the SMAPI layer and do not require any interaction with the speech recognition engine. Consequently, they can be made at any time, independent of the speech focus.

- SmSetArg

### 1.3.2 Callback and Dispatching Functions

These functions are also implemented locally in the application's address space by the SMAPI layer.

- SmAddCallback
- SmDispatch
- SmReceiveMsg
- SmRemoveCallback

### 1.3.3 Data Access Functions

The data access functions manipulate data received by the application. They are independent of the speech focus.

- SmGetAlphabets
- SmGetAlternates
- SmGetAnnotations
- SmGetApplication
- SmGetApplications
- SmGetAudioLevel
- SmGetBinaryItemValue
- SmGetCodepage
- SmGetDefaultTopics
- SmGetDescriptions
- SmGetEngineState
- SmGetEnrollId
- SmGetEnrollIds
- SmGetEventId
- SmGetEventOptions
- SmGetFirmWords
- SmGetFlags
- SmGetFocusState

- SmGetIncrements
- SmGetItemValue
- SmGetLanguages
- SmGetMicState
- SmGetMsgName
- SmGetMsgType
- SmGetNameValue
- SmGetNumberWordMsgs
- SmGetOptions
- SmGetPercentages
- SmGetPhraseScore
- SmGetPreferred
- SmGetPronunciations
- SmGetRc
- SmGetRcDescription
- SmGetRcName
- SmGetSampleRates
- SmGetScriptFlags
- SmGetScripts
- SmGetService
- SmGetSessionId
- SmGetSeverity
- SmGetSizes
- SmGetSpeechDataArchive
- SmGetSpeechDataOptions
- SmGetSpeechDataSize
- SmGetSpeechDataVersion
- SmGetSpelling
- SmGetSpellings
- SmGetStates
- SmGetStatus
- SmGetTags
- SmGetTask
- SmGetTaskFlags
- SmGetTasks
- SmGetTimes
- SmGetTopics
- SmGetTrained



- SmGetUserId
- SmGetUserIds
- SmGetUsers
- SmGetUtteranceNumber
- SmGetVocabList
- SmGetVocabName
- SmGetVocabPath
- SmGetVocWords
- SmGetWords
- SmGetWordTimes
- SmReturnRc
- SmReturnRcDescription
- SmReturnRcName

### 1.3.4 Connection Functions

These functions enable an application to connect to, or disconnect from, the speech recognition engine.

- SmApiVersionCheck
- SmOpen
- SmConnect
- SmDisconnect
- SmClose

### 1.3.5 Session Functions

The session-sharing related calls are appropriate when an application is in a particular state.

- SmDetachSessions
- SmReleaseFocus
- SmRequestFocus
- SmRequestMicOff
- SmRequestMicOn

### 1.3.6 Database Functions

These functions access database information. Although they do divert some engine resources, they do not interfere with the application's use of the speech recognition engine. Consequently, they are permitted at any time after calling SmConnect. Some returned information pertains to a specific session; some pertains to the speech recognition engine as a whole.

- SmQueryAddedWords
- SmQueryAddedWordsEx
- SmQueryAlternates
- SmQueryDefaults
- SmQueryEnabledVocabs
- SmQueryEnrollIds
- SmQueryLanguages
- SmQueryPronunciation
- SmQueryPronunciations
- SmQueryPronunciationsEx
- SmQuerySessions
- SmQueryTasks
- SmQueryTopics
- SmQueryUserDefault
- SmQueryUserInfo
- SmQueryUsers
- SmQueryVocabs
- SmQueryWord

### 1.3.7 Administrative Functions

The following functions are administrative. They do not change the state of the engine for the application with focus:

- SmSetDefault
- SmSetUserDefault
- SmSetUserInfo

### 1.3.8 Speech Recognition Engine State Functions

These functions set and query the state of the speech recognition engine.

- SmSet
- SmQuery

### 1.3.9 Vocabulary Functions

These functions change the state of the active, dynamically specified vocabularies, and they are handled independently for each session.

- SmAddPronunciation

- SmAddToVocab
- SmCorrectText
- SmDefineVocab
- SmDefineVocabEx
- SmDisableVocab
- SmDiscardData
- SmEnableVocab
- SmEventNotify
- SmHaltRecognizer
- SmNewContext
- SmRecognizeNextWord
- SmRemoveFromVocab
- SmRemovePronunciation
- SmUndefineVocab
- SmWordCorrection

### 1.3.10 Audio Functions

These functions change the state of the audio system. Given a single audio source, this changes the engine state for the application with focus. Therefore, these functions can only be called by the application with focus.

- SmCancelPlayback
- SmMicOn
- SmMicOff
- SmPlayMessage
- SmPlayUtterance
- SmPlayWords

### 1.3.11 Parallel Session API Calls

The feature of having multiple engine connections from a single application is known as parallel sessions. If you want more than one connection from your application, you must use the parallel session calls. The name of a parallel session call is the same as the regular function call except for two things: the name of the call has the characters Ses inserted after the Sm, for example the parallel session disconnect call is SmSesDisconnect. Secondly, each parallel session call takes one additional parameter, which is the session ID. This is always the first parameter. So for the disconnect example, the call is SmSesDisconnect(hSession), where hSession is the session ID. The session ID is returned in the first parameter of the SmSesOpen call. The SmSesOpen returns SM\_RC\_NO\_MORE\_CONNECTIONS if SM\_MAX\_SESSIONS (8) is exceeded. Currently you are allowed to have up to 8 parallel connections from one application.



## 2 SMAPI Starter Set APIs

To develop a full-function speech application, you do not need to use all of the functions provided in the SMAPI. You can use a subset of calls, known as the Starter Set SMAPIs. These calls do not use separate header or library files.

**Please Note:**

The ViaVoice Run Time Kit contains several user interface applications that you must include with your speech-aware application. (For more information, consult the SMAPI Developer's Guide documentation.) Inclusion of these applications ensures optimal recognition performance for your users. They also serve as reusable applications, so that you don't have to develop these same functions yourself (setting up the microphone, changing ViaVoice system parameters, enrollment, and managing the user's personal vocabulary).

### 2.1 Starter Set SMAPIs for Command and Control

When writing your command and control application, you must first establish a connection with the speech recognition engine. Next, you direct the engine to start processing speech. Your application interacts with the engine to set up and activate vocabularies and grammars. Most of your work is involved in processing the recognized speech. When you are done, you disconnect from the engine. Here are the Starter Set SMAPIs for command and control applications:

- SmOpen
- SmConnect
- SmDefineGrammar
- SmDefineVocab
- SmDefineVocabEx
- SmEnableVocab
- SmDisableVocab
- SmRequestFocus
- SmReleaseFocus
- SmMicOn
- SmMicOff
- SmRecognizeNextWord
- SmReceiveMsg
- SmGetFirmWords
- SmGetAnnotations
- SmDisconnect
- SmClose

### 2.1.1 Establishing a Recognition Session

To establish a recognition session with the engine, use `SmOpen` to set up local parameters for starting the engine and `SmConnect`, which actually establishes the link between your application and the engine. You can use `SmSetArg` to set the input parameters to `SmOpen` and `SmConnect` (such as language, user ID, and enroll ID), or you can specify to take the defaults using the default parameters, `SM_DEFAULT_USERID`.

### 2.1.2 Setting Up and Enabling Vocabularies

A vocabulary is a list of words and/or phrases that the engine uses to match speech input and translate it into text. Your command and control application specifies the set of active words by defining and enabling one or more vocabularies. A vocabulary can be predefined in a grammar file, or it can be created at runtime as a dynamic command vocabulary. For more information on grammars, refer to the SMAPI Developer's Guide.

By passing the name of your compiled grammar file to `SmDefineGrammar`, the words and phrases of your grammar are defined to the engine. `SmDefineVocab` allows you to build command vocabularies dynamically by passing a list of words and phrases that make up that vocabulary. You would also use this call to define the names for any external references in your grammar file. The function `SmDefineVocabEx` is an extension called by a client application to define a vocabulary. `SmEnableVocab` enables a vocabulary. When decoding speech, the engine searches all of the enabled vocabularies to translate speech to text. You can have more than one vocabulary active in your application at a time.

To improve performance (both recognition speed and accuracy), your application should narrow the possibilities by enabling and disabling vocabularies as they are needed. You disable a vocabulary by using the `SmDisableVocab` call.

### 2.1.3 Directing the Engine to Process Speech

The speech recognition engine supports both shared and parallel sessions, which means that there can be several applications connected to the speech recognition engine at the same time, and that a single application can have more than one connection to the speech recognition engine at the same time, too. You should design your application so that it cooperates with other speech-aware applications for control of the microphone. This is known as "having speech focus." An application must have speech focus to receive recognized text; this may or may not equate to input focus. You need to ensure that your application has speech focus when it needs it. This is done by using `SmRequestFocus` to request speech focus and `SmReleaseFocus` to release speech focus when you're done. Once you have speech focus, your application must tell the engine to start processing audio data. `SmMicOn` engages the engine to start processing audio data, and `SmMicOff` tells the engine to stop processing audio data.

### 2.1.4 Processing Recognized Commands

The bulk of the work in your application involves getting the recognized commands from the engine and determining what to do with it. For a command and control application, the engine waits for your application to direct it to start recognizing words. Use `SmRecognizeNextWord` to start the engine looking for words to decode. When the engine recognizes a word from a dynamic command vocabulary, it returns the word with alternates in the `SM_RECOGNIZED_WORD` message. When a phrase is recognized from a grammar, the engine returns the set of words and annotations in the `SM_RECOGNIZED_PHRASE` message. Use `SmReceiveMsg` to accept and process these messages. In either case, the engine stops decoding until your application makes another `SmRecognizeNextWord` call. This gives you a chance to change active vocabularies for recognizing the next command. While halted, however, the engine continues to capture and process audio, so no words are lost.

### 2.1.5 Disconnecting from the Engine

When your application is closing down, you must disconnect and close the session with the speech recognition engine. To do this, use `SmDisconnect` and `SmClose`.

## 2.2 Starter Set SMAPIs for Dictation

The Dictation Starter Set SMAPIs will enable you to develop a dictation application that supports commands, dictation, word-error playback and correction, and language model cache updates.

- `SmOpen`
- `SmConnect`
- `SmDefineGrammar`
- `SmDefineVocab`
- `SmDefineVocabEx`
- `SmEnableVocab`
- `SmDisableVocab`
- `SmUndefineVocab`
- `SmMicOn`
- `SmMicOff`
- `SmRequestFocus`
- `SmReleaseFocus`
- `SmNewContext`
- `SmRecognizeNextWord`
- `SmGetFirmWords`

- SmHaltRecognizer
- SmEventNotify
- SmReceiveMsg
- SmQueryAlternates
- SmPlayWords
- SmWordCorrection
- SmDiscardData
- SmDisconnect
- SmClose

### 2.2.1 Establishing a Recognition Session

To establish a recognition session with the engine, use SmOpen to set up local parameters for starting the engine and SmConnect, which actually establishes the link between your application and the engine. You can use SmSetArg to set the input parameters to SmOpen and SmConnect (such as language, user ID, and enroll ID), but you'll probably want to take the defaults using SM\_DEFAULT\_\* parameters.

### 2.2.2 Setting Up and Enabling Vocabularies

A vocabulary is a list of words and/or phrases that the engine uses to match speech input and translate it into text. Your application specifies the set of active words by defining and enabling one or more vocabularies. A dictation application must first enable a predefined domain (such as the general office vocabulary) before the engine can use it to decode text. It does so by using the SmEnableVocab function. To have command words recognized during dictation (such as "stop dictation"), the application must enable the dictation vocabulary first, and then define the command vocabulary. Grammar vocabularies cannot be active at the same time as dictation vocabularies, although you can support both dictation and command and control within the same application. SmDefineGrammar defines a grammar vocabulary to the engine, and SmDefineVocab defines a dynamic command vocabulary. SmEnableVocab enables all types of vocabularies (dictation, grammar, and dynamic command). To improve performance (both recognition speech and accuracy), your application should narrow the possibilities by enabling and disabling vocabularies as they are needed. You disable a vocabulary by using the SmDisableVocab call. You can also use SmUndefineVocab to delete a dynamic command vocabulary, but it must have been disabled first.

### 2.2.3 Directing the Engine to Process Speech

The speech recognition engine supports both shared and parallel sessions, which means that there can be several applications connected to the speech recognition engine at the same



time, and that a single application can have more than one connection to the speech recognition engine at the same time, too. You should design your application so that it cooperates with other speech-aware applications for control of the microphone. This is known as "having speech focus." An application must have speech focus to receive recognized text; this may or may not equate to input focus. You need to ensure that your application has speech focus when it needs it. This is done by using `SmRequestFocus` to request speech focus and `SmReleaseFocus` to release speech focus when you're done. Once you have speech focus, your application must tell the engine to start processing audio data. `SmMicOn` engages the engine to start processing audio data, and `SmMicOff` tells the engine to stop processing audio data.

### 2.2.4 Processing Recognized Text

The bulk of the work in your application involves getting the recognized text from the engine and determining what to do with it. For a dictation application, the engine waits for your application to direct it to start recognizing words. Use `SmRecognizeNextWord` to start the engine looking for words to decode. If the engine recognizes a word from a dictation vocabulary, the available decoded words are sent to the application in an `SM_RECOGNIZED_TEXT` reply message. The `SM_RECOGNIZED_TEXT` reply message provides a list of firm words. Use `SmReceiveMsg` to receive messages from the engine. The engine continues decoding words as they are spoken. `SmEventNotify` requests that your application be notified when the engine completes decoding all of the audio up to the point of the call.

Commands are handled as described in the preceding section, "Processing Recognized Commands".

When a word is recognized, there is a reply structure generated which you can access via function calls. When your application is notified that a word is recognized, use `SmGetFirmWords` to retrieve the list of words from the reply message. The engine continues decoding speech after a word is recognized during dictation. There are times your application may need to halt the engine during dictation recognition; for example, while setting new context or adding pronunciations. To do this, use `SmHaltRecognizer`, which will temporarily halt recognition. A subsequent `SmRecognizeNextWord` call restarts the recognition process.

`SmNewContext` is called when dictation begins at a new location in the document. Context refers to the previous and following words, which are used to decode the current word.

If the engine recognizes a word from a command vocabulary, the word is sent to the application in an `SM_RECOGNIZED_WORD` reply message. The engine halts and waits for further instruction from the application. Use `SmGetFirmWords` on `SM_RECOGNIZED_WORD` to get the recognized text from the engine.

### 2.2.5 Correcting Errors

Correcting misrecognized words and improving recognition by updating personal data files are key functions of dictation applications. Use `SmPlayWords` to play back previously

spoken words or phrases to the user. This can be of great help to the user in the correction process. Use `SmQueryAlternates` to provide alternative choices when incorrectly recognized words have been detected by the user. `SmWordCorrection` corrects a misrecognized word. It notifies the engine that a word or sequence of words was corrected by the user. It can be used to update the user's text vocabulary, word-usage model, and pronunciation pool. Since the data that the engine stores can be resource intensive, use `SmDiscardData` to discard audio and error correction data for recognized words that are not referenced again. This can conserve speech recognition engine disk space requirements.

### 2.2.6 Disconnecting from the Engine

When your application is closing down, you must disconnect and close the session with the speech recognition engine. To do this, use `SmDisconnect` and `SmClose`.

## 3 Function Calls to the Speech Recognition Engine

This chapter lists and describes the function calls that go directly to the speech recognition engine.

### 3.1 SmAddCallback

#### Purpose

SmAddCallback adds a single callback routine for the specified message.

#### Syntax

```
int SmAddCallback ( char      *reply_name,
                    SmHandler *handler,
                    void      *client_data );
```

#### Parameters

*reply\_name*

input - The name of the type of message.

*handler*

input - The function name of the routine that handles the message.

*client\_data*

input - Data passed back to the handler when it is called.

#### Return Values

SM\_RC\_EALLOC  
SM\_RC\_ENOMEM  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_OK  
SM\_RC\_SM\_NOT\_OPEN

#### Task Related Functions and Callbacks

SmDispatch  
SmRemoveCallback

## 3.2 SmAddEnrollid

### Purpose

SmAddEnrollid adds another enrollid for a particular userid. This enrollid can be used to enroll another acoustic environment or sample rate for that userid.

### Syntax

```
int SmAddEnrollid ( char    *user_id,
                   char    *enroll_id,
                   char    *description,
                   char    *language,
                   short    sample_rate,
                   SM_MSG  *reply );
```

### Parameters

*user\_id*      input - Identifies the user to which the enrollid is added. Must be no more than SM\_MAX\_USERID\_LEN in length

*enroll\_id*    input - Identifies the enrollid to be added. Must be no more than SM\_MAX\_ENROLLID\_LEN in length.

*description*      input - A description of the enrollid.

*language*      input - The language used by this enrollid. Must be no more than SM\_MAX\_LANGUAGE\_LEN in length.

*sample\_rate*      input - The sample rate used for recording by this enrollid. Valid values 8, 11, 22

*reply*          input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
```

SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_USERID  
SM\_RC\_BAD\_ENROLLID  
SM\_RC\_ENROLLID\_EXISTS  
SM\_RC\_BAD\_DESCRIPTION

## **Reply Structure Functions by Message Type**

SM\_ADD\_ENROLLID\_REPLY

## **Task Related Functions and Callbacks**

SmAddUser  
SmRemoveEnrollid  
SmRemoveUser  
SmRequestNewEnrollid  
SmNaddEnrollidCallback

### 3.3 SmAddPronunciation

#### Purpose

SmAddPronunciation adds a new pronunciation. This function provides the method for associating a pronunciation with a spelling. The following restrictions apply when calling SmAddPronunciation:

- The engine must be halted.
- The utterance used to build the pronunciation must be complete.
- If an SmAddPronunciation call fails and the call is immediately repeated with exactly the same parameters, the engine will automatically apply more lenient threshold parameters the second time.

For a detailed description of the SM\_REJECTION\_THRESHOLD parameter see [Section 3.83 \[SmSet\], page 175](#).

**Please Note:** For this call to be valid, the application must first enable audio saving through:

```
SM_MSG reply;
SmSet( SM_SAVE_AUDIO, SM_SAVE_AUDIO_DEFAULT, &reply );
```

#### Syntax

```
int SmAddPronunciation ( char    *spelling,
                        char    *pronunciation,
                        long     uttno,
                        short    repetitions,
                        long     options,
                        SM_MSG *reply );
```

#### Parameters

*spelling*      input - A null-terminated character string containing the spelling for the added pronunciation.

*pronunciation*  
input - A null-terminated character string containing an optional sounds-like spelling that matches the acoustics and is used to help find the pronunciation that best matches both the spelling and the speech. If omitted, the value of the spelling parameter is used to help the search.

*uttno*          input - Utterance number of the utterance in which a recording of the word is stored.

*repetitions*   input - Number of repetitions must be set to 1.

*options*       input - The options are:

- `SM_ADD_PRONUNCIATION_ADD` - The pronunciation is added to any existing ones for this word, up to a maximum of `SM_MAX_ADDED_PRONUNCIATIONS`. This is the default.
- `SM_ADD_PRONUNCIATION_REPLACE` - Delete existing pronunciations and replace them with this new one.
- `SM_ADD_PRONUNCIATION_PHONETIC` - An acoustic utterance is not used, and the pronunciation parameter must hold a string of blank separated phone spellings. The string may be preceded by an optional sounds-like spelling surrounded by ~ characters.
- `SM_ADD_PRONUNCIATION_ACOUSTIC` - Only the acoustics are used to construct the pronunciation, the spelling is ignored during the search.
- `SM_ADD_PRONUNCIATION_PERMANENT` - This option forces the added pronunciation to be saved in the personal pool so it is available for later sessions.

*reply*      input/output - A pointer to a reply structure or to `SmAsynchronous`, indicating that the call is made asynchronously.

## Return Values

`SM_RC_DEALLOCATING_SH_MEM`  
`SM_RC_EALLOCS`  
`SM_RC_EBADHANDLE`  
`SM_RC_ENOCONN`  
`SM_RC_ENOMEM`  
`SM_RC_ENOMSG`  
`SM_RC_ENOSERVER`  
`SM_RC_EUNEXP`  
`SM_RC_SM_NOT_OPEN`  
`SM_RC_OK`  
`SM_RC_INVALID_PARM_MAX_LEN`  
`SM_RC_ADDWORD_LIMIT_EXCEEDED`  
`SM_RC_BAD_ACOUSTICS`  
`SM_RC_BAD_ADDWORD`  
`SM_RC_ILLEGAL_SOUNDLIKE`  
`SM_RC_ILLEGAL_SPELLING`  
`SM_RC_MISMATCHED_ACOUSTICS`  
`SM_RC_NOT_ADDED`  
`SM_RC_NOT_VALID_REQUEST`  
`SM_RC_NOT_YET`

SM\_RC\_SERVER\_ERROR  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR  
SM\_RC\_SERVER\_FILE\_READ\_ERROR  
SM\_RC\_SERVER\_FILE\_WRITE\_ERROR  
SM\_RC\_SERVER\_MALLOC\_ERROR

## **Reply Structure Functions by Message Type**

SM\_ADD\_PRONUNCIATION\_REPLY  
SmGetRc  
SmGetSpelling  
SmGetSpellings

## **Task Related Functions and Callbacks**

SmQueryPronunciation  
SmQueryPronunciations  
SmRemovePronunciation  
SmNaddPronunciationCallback  
SmNqueryPronunciationCallback  
SmNqueryPronunciationsCallback  
SmNremovePronunciationCallback



### 3.4 SmAddToVocab

#### Purpose

SmAddToVocab adds words to a vocabulary. This function adds words to predefined vocabularies or to a vocabulary previously created by SmDefineVocab. SmAddToVocab can be used to dynamically change command vocabularies within an application. If any of the specified words do not have an existing pronunciation, the call returns the list of words without pronunciations. Missing pronunciations must be added by using the SmAddPronunciation function. Pronunciations exist for all words in the predefined vocabularies and for words added by the user. For a predefined vocabulary the words are added in the user's personal pronunciation area of the vocabulary.

This call is valid only when the speech recognition engine is not decoding speech to text.

If a specified word does not exist in the current pool, the backup dictionary will be searched. If the word is found in the backup dictionary, it will be added to the temporary pool, except that when SmAddToVocab specifies a dictation vocabulary such as "text," the word will be added to the personal pool.

#### Syntax

```
int SmAddToVocab ( char      *vocab,
                  short      nvocwords,
                  SM_VOCWORD *vocwords[],
                  SM_MSG      *reply );
```

#### Parameters

- vocab*        input - The name of the vocabulary to which words are added.
- nvocwords*   input - The number of words added to the vocabulary, up to the limit defined by SM\_MAX\_WORDS.
- vocwords*    input - The spellings of the words added to the vocabulary.
- reply*        input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
```

SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_VOCAB  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR  
SM\_RC\_SERVER\_FILE\_WRITE\_ERROR

## Reply Structure Functions by Message Type

SM\_ADD\_TO\_VOCAB\_REPLY  
SmGetRc  
SmGetVocabName  
SmGetVocWords

## Task Related Functions and Callbacks

SmDefineVocab  
SmDisableVocab  
SmEnableVocab  
SmQueryAddedWords  
SmQueryEnabledVocabs  
SmQueryVocabs  
SmQueryWord  
SmRemoveFromVocab  
SmUndefineVocab  
SmNaddToVocabCallback  
SmNdefineVocabCallback  
SmNdisableVocabCallback  
SmNenableVocabCallback  
SmNqueryAddedWordsCallback  
SmNqueryEnabledVocabsCallback  
SmNqueryVocabsCallback  
SmNqueryWordsCallback  
SmNremoveFromVocabCallback  
SmNundefineVocabCallback

## 3.5 SmAddUser

### Purpose

SmAddEnrollid adds another speech user to the system.

### Syntax

```
int SmAddUser ( char    *user_id,
                char    *user_name,
                char    *description,
                char    *password,
                SM_MSG *reply );
```

### Parameters

- user\_id*      input - Short name to identify the user. Must be no more than SM\_MAX\_USERID\_LEN in length. The following characters must not appear in a userid: : \ / \n \r \t " ' ! # \$ % & ( ) \* , . ; < = > ? [ ] ^ \_ { | } ~
- user\_name*    input - Full name of the user.
- description*   input - A description of the user.
- password*    input - Password used to limit access to userid. ( Password access disabled on Windows ). Must be no more than SM\_MAX\_PASSWORD\_LEN in length. The following characters must not appear in a password: : \ / \n \r \t " ' ! # \$ % & ( ) \* , . ; < = > ? [ ] ^ \_ { | } ~
- reply*        input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
 SM\_RC\_EALLOCS  
 SM\_RC\_EBADHANDLE  
 SM\_RC\_ENOCONN  
 SM\_RC\_ENOMEM  
 SM\_RC\_ENOMSG  
 SM\_RC\_ENOSERVER  
 SM\_RC\_EUNEXP  
 SM\_RC\_SM\_NOT\_OPEN

SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_USERID  
SM\_RC\_USERID\_EXISTS  
SM\_RC\_BAD\_PASSWORD

## **Reply Structure Functions by Message Type**

SM\_ADD\_USER\_REPLY

## **Task Related Functions and Callbacks**

SmAddEnrollid  
SmRemoveEnrollid  
SmRemoveUser  
SmRequestNewEnrollid  
SmNaddUserCallback

## 3.6 SmApiVersionCheck

### Purpose

SmApiVersionCheck verifies the current version of the SMAPI. This function checks whether the version of the SMAPI used to compile the speech-aware application is compatible with the API currently installed on the system. The reply message contains the return code indicating the current status.

### Syntax

```
int SmApiVersionCheck ( char *caller_version,  
                        char **sm_version );
```

### Parameters

*caller\_version*

input - Indicates the version used to compile the application. The constant SM\_API\_VERSION\_STRING is used for the comparison check.

*sm\_version*

output - The version of the SMAPI currently installed on the system.

### Return Values

SM\_RC\_OK

SM\_RC\_WRONG\_SM\_VERSION

## 3.7 SmAutoComplete

### Purpose

SmAutoComplete is called by a client application to request completions of a specified string. SmAutoComplete will search active vocabularies, including topics, for completions of the given string. If the number requested cannot be satisfied from the active vocabularies, the backup dictionary will be searched.

### Syntax

```
int SmAutoComplete ( unsigned long  flags,
                    int             max_depth,
                    char            *spelling,
                    SM_MSG          *reply);
```

### Parameters

*flags*           input - reserved

*max\_depth*   input - maximum number of completions to return. Limited to SM\_MAX\_VOCWORDS.

*spelling*      input - string for which completions are requested

*reply*          input - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
SM_RC_SERVER_FILE_OPEN_ERROR
```

SM\_RC\_SERVER\_FILE\_READ\_ERROR

SM\_RC\_SERVER\_MALLOC\_ERROR

## **Reply Structure Functions by Message Type**

SM\_AUTO\_COMPLETE\_REPLY

SmGetSpelling

SmGetSpellings

## **Task Related Functions and Callbacks**

SmNautoCompleteCallback

## 3.8 SmCancelPlayback

### Purpose

SmCancelPlayback cancels the request to play back a message, utterance, or words. This function cancels a play request from the SmPlayMessage, SmPlayUtterance, and SmPlayWords functions.

### Syntax

```
int SmCancelPlayback ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_NOT\_VALID\_REQUEST

### Reply Structure Functions by Message Type

SM\_CANCEL\_PLAYBACK\_REPLY  
SmGetRc



## Task Related Functions and Callbacks

SmPlayMessage  
SmPlayUtterance  
SmPlayWords  
SmNcancelPlaybackCallback  
SmNplayMessageCallback  
SmNplayMessageStatusCallback  
SmNplayUtteranceCallback  
SmNplayUtteranceStatusCallback  
SmNplayWordsCallback  
SmNplayWordsStatusCallback

## 3.9 SmClose

### Purpose

SmClose closes the SMAPI connection. If the speech-aware application has not already called the SmDisconnect function to terminate the connection with the speech recognition engine, the speech API does so before executing SmClose.

### Syntax

```
int SmClose ( );
```

### Parameters

None.

### Return Values

SM\_RC\_OK  
SM\_RC\_SM\_NOT\_OPEN

### Task Related Functions and Callbacks

SmDisconnect  
SmNdisconnectCallback

### 3.10 SmConnect

#### Purpose

SmConnect connects to the speech recognition engine. This function establishes a session with the speech recognition engine. The desired type of session and other necessary information are provided by setting SMAPI attributes. Once a session is established the session type cannot be changed. To change the session type, call SmDisconnect then call SmConnect again.

#### Syntax

```
int SmConnect ( int      nargs,
                SmArg   *Args,
                SM_MSG *reply );
```

#### Parameters

- nargs*        input - The number of arguments in the accompanying argument list.
- Args*        input - A set of arguments that indicate the parameters used to connect to the speech recognition engine. Speech API arguments can also be set prior to an SmConnect call by using the SmSetArg function, which specifies the parameter or attribute name and its value. The attributes passed to this function determine the session type. For details on establishing a speech session, see the SMAPI Developer's Guide. For details on the attributes, see [Chapter 7 \[Attributes\]](#), page 313.
- reply*        input - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
```

SM\_RC\_OK  
SM\_RC\_INCOMPATIBLE\_ENROLLMENT  
SM\_RC\_NAVIGATOR\_ALREADY\_DEFINED  
SM\_RC\_ALREADY\_CONNECTED  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_ADDWORD  
SM\_RC\_BAD\_AP  
SM\_RC\_BAD\_AUDIO  
SM\_RC\_BAD\_DECO  
SM\_RC\_BAD\_DESCRIPTION  
SM\_RC\_BAD\_ENROLLID  
SM\_RC\_BAD\_PASSWORD  
SM\_RC\_BAD\_SCRIPT  
SM\_RC\_BAD\_TASKID  
SM\_RC\_BAD\_USERID  
SM\_RC\_ENROLLID\_EXISTS  
SM\_RC\_ENROLLID\_RUNNING  
SM\_RC\_ENROLLMENT\_NOT\_COMPLETE  
SM\_RC\_MISMATCHED\_ALPHABET  
SM\_RC\_MISMATCHED\_LANGUAGE  
SM\_RC\_MISMATCHED\_SCRIPT  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_ERROR  
SM\_RC\_SEVER\_FILE\_OPEN\_ERROR

## Reply Structure Functions by Message Type

SM\_CONNECT\_REPLY  
SmGetMsgType  
SmGetRc  
SmGetService  
SmGetSessionId  
SmGetTask  
SmGetTasks  
SmGetUserId

Only recognition sessions:

SmGetCodepage  
SmGetEnrollId

SmGetEnrollIds  
SmGetLanguages  
SmGetSessionId  
SmGetUserId  
SmGetUserIds

## **Task Related Functions and Callbacks**

SmOpen  
SmSetArg  
SmNconnectCallback

## 3.11 SmCorrectText

### Purpose

SmCorrectText updates the user's voice model with a sequence of text the user considers correct.

This function provides the speech recognition engine with a sequence of correctly recognized text. This text can be used to adapt the user's voice model, thereby improving future recognition of dictated text.

### Syntax

```
int SmCorrectText ( short    nwords,
                   SM_WORD *words[],
                   SM_MSG  *reply );
```

### Parameters

*nwords*      input - The number of words in the text sequence.

*words*        input - An array of pointers to the words of the text.

*reply*        input/output - The pointer to a reply structure indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR  
SM\_RC\_SERVER\_FILE\_WRITE\_ERROR

## Reply Structure Functions by Message Type

SM\_CORRECT\_TEXT\_REPLY

SmGetRc

## Task Related Functions and Callbacks

SmNewContext

SmRemovePronunciation

SmWordCorrection

SmNcorrectTextCallback

SmNnewContextCallback

SmNremovePronunciationCallback

SmNwordCorrectionCallback

### 3.12 SmCorrectTextEx

#### Purpose

SmCorrectTextEx updates the user's voice model with a sequence of text the user considers correct. This function is an extension of the function SmCorrectText. This function provides the speech recognition engine with a sequence of correctly recognized text for a particular document. This text can be used to adapt the user's voice model, thereby improving future recognition of dictated text. Since the api will only support specifying up to SM\_MAX\_WORDS in a request this function may be called multiple times for a specific document.

#### Syntax

```
int SmCorrectTextEx ( unsigned long  flags,
                      int           docID,
                      short          nwords,
                      SM_WORD        *words[],
                      SM_MSG          *reply );
```

#### Parameters

*flags* input - Control flags. Specify only one of the first four. The last one can be specified with any of the others. Valid values:

*SM\_CORRECT\_TEXT\_OPEN*

Set reference point if words already in cache, otherwise clear reference point.

*SM\_CORRECT\_TEXT\_UPDATE*

Update reference point ( undo previous one and update cache again. )

*SM\_CORRECT\_TEXT\_CLOSE*

Discard reference point

*SM\_CORRECT\_TEXT\_DISCARD*

Undo any updates made for this document

*SM\_CORRECT\_TEXT\_COMPLETE*

Specifies this request contains the end of the document.

*docID* input - Application defined document identifier.

*nwords* input - The number of words in the text sequence.

*words* input - An array of pointers to the words of the text.

*reply* input/output - The pointer to a reply structure indicating that the call is made asynchronously.



## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_DOCUMENT\_CACHED  
SM\_RC\_BAD\_NAME

## Reply Structure Functions by Message Type

SM\_CORRECT\_TEXT\_REPLY  
SmGetRc

## Task Related Functions and Callbacks

SmCorrectText  
SmNewContext  
SmRemovePronunciation  
SmWordCorrection  
SmNcorrectTextCallback  
SmNnewContextCallback  
SmNremovePronunciationCallback  
SmNwordCorrectionCallback

### 3.13 SmDefineGrammar

#### Purpose

SmDefineGrammar defines a grammar-based vocabulary.

This function dynamically defines a new vocabulary that can be enabled through the SmEnableVocab function, along with dynamic command vocabularies defined through SmDefineVocab. The vocabulary content is specified by a precompiled FSG file, produced by the grammar compiler.

Like SmDefineVocab, SmDefineGrammar returns a list of grammar words that don't have pronunciations. Unlike SmDefineVocab, if any pronunciation is missing, SmDefineGrammar fails with a return code of SM\_RC\_NOT\_INVOCAB, since recognition of a grammar network with missing pronunciations is not well defined. Use SmGetVocWords to retrieve the list of missing words.

If external lists are missing (for example, there was no SmDefineVocab before the SmDefineGrammar), SmDefineGrammar will also fail. This time, the return code is SM\_RC\_MISSING\_EXTERN. The names of the external lists are returned as if they were words without pronunciations, and can also be obtained with SmGetVocWords. Since missing externs is really a program logic error, these are checked and returned before determining if any words are missing pronunciations. So, if both externs and words are missing, you first get SM\_RC\_MISSING\_EXTERN. When that is fixed, you get SM\_RC\_NOT\_INVOCAB.

Note that pronunciations can be found in the user's personal pool, application-specific pools (created by Dictionary Builder) or the base domain pool. If a specified word does not exist in the current pool, the backup dictionary will be searched. If the word is found in the backup dictionary, it will be added to the temporary pool, except that when SmAddToVocab specifies a dictation vocabulary such as "text," the word will be added to the personal pool.

#### Syntax

```
int SmDefineGrammar ( char    *vocab,
                      char    *grammar,
                      long     options,
                      SM_MSG  *reply );
```

#### Parameters

*vocab*        input - The name of the new grammar.

*grammar*    input - The fully qualified path name of the FSG file containing the compiled grammar.

*options*     flags include:

              SM\_PHRASE\_ALLOW\_SILENCES  
                  Allow inter-word silences within phrase.

*SM\_PHRASE\_SHOW\_SILENCES*

Silence indicated in returned phrase. Silences are returned via the word spelling "" in the recognized phrase message.

*SM\_PHRASE\_NO\_SILENCES*

Don't allow inter-word silences

*SM\_PHRASE\_ALLOW\_INSERTIONS*

Allow insertions within phrase

*SM\_PHRASE\_SHOW\_INSERTIONS*

Insertions indicated in returned phrase. Insertions are returned via an empty word spelling "" in the recognized phrase message.

*SM\_PHRASE\_NO\_INSERTIONS*

Don't allow insertions

Note: These run-time flags override settings compiled in the FSG file.

*reply* input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

## Return Values

SM\_RC\_OK

SM\_RC\_MISSING\_EXTERN

SM\_RC\_NOT\_VALID\_REQUEST

SM\_RC\_NOT\_INVOCAB

SM\_RC\_BAD\_VOCAB

SM\_RC\_SERVER\_FILE\_MALLOC\_ERROR

SM\_RC\_SERVER\_FILE\_OPEN\_ERROR

SM\_RC\_SERVER\_FILE\_READ\_ERROR

## Reply Structure Functions by Message Type

SM\_DEFINE\_GRAMMAR\_REPLY

SmNdefineGrammarCallback

SmGetVocabName

SmGetGrammarPath

SmGetVocWords

## Task Related Functions and Callbacks

SmEnableVocab

SmUndefineGrammar

SmDisableVocab

## Chapter 3: Function Calls to the Speech Recognition Engine

SmNdisableVocabCallback  
SmNdenableVocabCallback  
SmNundefineVocabCallback

### 3.14 SmDefineVocab

#### Purpose

SmDefineVocab defines a new vocabulary.

This function dynamically creates a new vocabulary that can later be used by calling the SmEnableVocab function. The vocabulary created by SmDefineVocab consists only of the words specified in the call with all words receiving an equal voice model weighting. This function can be used to dynamically create command vocabularies in an application. If any of the specified words do not have an existing pronunciation, the call returns a list of words without pronunciations. Pronunciations can be found in the predefined vocabulary and in the user's personal vocabulary. Predefined vocabularies are not dynamic. Dynamic vocabularies are intended for command vocabulary recognition.

SmDefineVocab takes more time to execute than SmEnableVocab and SmDisableVocab; therefore, it is more efficient to define a vocabulary once and enable/disable frequently rather than to define multiple times.

This call is valid only when the speech recognition engine is not decoding speech to text. For examples of conditions when the engine is not decoding speech to text, see "Setting Up Vocabularies" in the SMAPI Developer's Guide. If a specified word does not exist in the current pool, the backup dictionary will be searched. If the word is found in the backup dictionary, it will be added to the temporary pool, except that when SmAddToVocab specifies a dictation vocabulary such as "text," the word will be added to the personal pool.

#### Syntax

```
int SmDefineVocab ( char      *vocab,
                   short      nvocwords,
                   SM_VOCWORD *vocwords[],
                   SM_MSG      *reply );
```

#### Parameters

- vocab*            input - The name of the new vocabulary.
- nvocwords*      input - The number of words in the new vocabulary. Limited to SM\_MAX\_VOCWORDS.
- vocwords*       input - The spellings of the words in the new vocabulary.
- reply*           input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_VOCAB  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR  
SM\_RC\_SERVER\_FILE\_WRITE\_ERROR

## Reply Structure Functions by Message Type

SM\_DEFINE\_VOCAB\_REPLY  
SmGetRc  
SmGetVocabName  
SmGetVocWords

## Task Related Functions and Callbacks

SmAddToVocab  
SmDisableVocab  
SmEnableVocab  
SmQueryAddedWords  
SmQueryEnabledVocabs  
SmQueryVocabs  
SmQueryWord  
SmRemoveFromVocab  
SmUndefineVocab  
SmNaddToVocabCallback  
SmNdefineVocabCallback  
SmNdisableVocabCallback

SmNenableVocabCallback  
SmNqueryAddedWordsCallback  
SmNqueryEnabledVocabsCallback  
SmNqueryVocabsCallback  
SmNqueryWordsCallback  
SmNremoveFromVocabCallback  
SmNundefineVocabCallback

### 3.15 SmDefineVocabEx

#### Purpose

SmDefineVocabEx is an extension of the SMAPI function SmDefineVocab. This function is called by a client application to define a vocabulary.

If a specified word does not exist in the current pool, the backup dictionary will be searched. If the word is found in the backup dictionary, it will be added to the temporary pool, except that when SmAddToVocab specifies a dictation vocabulary such as "text," the word will be added to the personal pool.

Calling SmDefineVocabEx with options SM\_VOCAB\_VOCWORDS and SM\_VOCAB\_COMMAND set is equivalent to calling SmDefineVocab. Calling SmDefineVocabEx with options SM\_VOCAB\_FILE and SM\_VOCAB\_FSG set is equivalent to calling SmDefineGrammar. Calling SmDefineVocabEx with options SM\_VOCAB\_WORDS and SM\_VOCAB\_PHRASE set is equivalent to calling SmDefineGrammar with SM\_PHRASE\_LITERAL set. Only a few combinations of format and type option flags are currently supported:

- SM\_VOCAB\_VOCWORDS + SM\_VOCAB\_COMMAND
- SM\_VOCAB\_VOCWORDS + SM\_VOCAB\_TEXT
- SM\_VOCAB\_FILE + SM\_VOCAB\_FSB
- SM\_VOCAB\_WORDS + SM\_VOCAB\_PHRASE

#### Syntax

```
int SmDefineVocabEx ( char    *vocab,
                     void    *data,
                     int     length,
                     long    options,
                     char    *acoustic_id,
                     char    *poolname,
                     SM_MSG  *reply );
```

#### Parameters

<i>vocab</i>	input - The name of the vocabulary being defined.
<i>data</i>	input - Data used to define the new vocabulary. The form of the data depends on the value of the options argument.
<i>length</i>	input - Length of data in bytes or number of elements
<i>options</i>	input - Options which specify how vocabulary is defined. These options are valid for almost all vocabularies:



*SM\_VOCAB\_REPLACE*

Replace any currently defined vocabulary with the same name. This eliminates the need to undefine the current vocabulary before defining the new one.

*SM\_VOCAB\_GLOBAL*

Active even if the client does not have focus.

*SM\_VOCAB\_NOHALT*

Engine does not halt after returning a command.

These options specify the format of the data:

*SM\_VOCAB\_BINARY*

The data is a buffer in binary format.

*SM\_VOCAB\_FILE*

The data is the name of a file.

*SM\_VOCAB\_WORDS*

The data is a list of words in ASCII format.

*SM\_VOCAB\_VOCWORDS*

The data is an array of pointers to SM\_VOCWORD structures and the length field specifies the number of elements in the array. This length must be limited to SM\_MAX\_VOCWORDS.

These options specify the type of vocabulary:

*SM\_VOCAB\_COMMAND*

Regular command vocabulary.

*SM\_VOCAB\_TEXT*

Acts like the "text" vocabulary but with no LM.

*SM\_VOCAB\_FSG*

Regular compiled grammar.

*SM\_VOCAB\_PHRASE*

Single-phrase grammar (for example, for Enrollment).

*SM\_VOCAB\_SELECT*

Select-phrase grammar (verb + any phrase)

*SM\_VOCAB\_VERBFINAL*

Verb follows selected phrase

In addition, the existing SM\_PHRASE\_ options may be set for grammars.

*acoustic\_id*

input - Reserved: must be set to NULL

*poolname*

input - Reserved: must be set to NULL

*reply*

input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_VOCAB  
SM\_RC\_MISSING\_EXTERN  
SM\_RC\_NOT\_INVOCAB  
SM\_RC\_NOT\_VALID\_REQUEST

## Reply Structure Functions by Message Type

SM\_DEFINE\_VOCABULARY\_REPLY  
SmGetVocabName  
SmGetVocabPath  
SmGetVocWords  
SmGetEventOptions  
SmGetOptions

## Task Related Functions and Callbacks

SmAddToVocab  
SmDefineVocab  
SmDefineGrammar  
SmDisableVocab  
SmEnableVocab  
SmQueryAddedWords  
SmQueryEnabledVocabs  
SmQueryVocabs  
SmQueryWord  
SmRemoveFromVocab  
SmUndefineVocab

SmNaddToVocabCallback  
SmNdefineVocabCallback  
SmNdisableVocabCallback  
SmNenableVocabCallback  
SmNqueryAddedWordsCallback  
SmNqueryEnabledVocabsCallback  
SmNqueryVocabsCallback  
SmNqueryWordsCallback  
SmNremoveFromVocabCallback  
SmNundefineVocabCallback  
SmNdefineVocabExCallback

## 3.16 SmDetachSessions

### Purpose

SmDetachSessions requests that one or more sessions detach. This function requests all applications connected to the speech recognition engine to disconnect from it. SmQuerySessions can be used to monitor the number of attached applications. The disconnect request is in the form of the SM\_REQUEST\_DETACH reply message structure. The engine does not force a session to disconnect and does not wait for a reply to the disconnect request. This function can be used for switching users (or user ID, enroll ID, task).

### Syntax

```
int SmDetachSessions ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

SM\_RC\_OK

### Reply Structure Functions by Message Type

SM\_DETACH\_SESSIONS\_REPLY

SmGetRc

### Task Related Functions and Callbacks

SmQuerySessions

SmNdetachRequestedCallback

SmNdetachSessionsCallback

SmNquerySessionsCallback

### 3.17 SmDisableVocab

#### Purpose

SmDisableVocab disables a defined vocabulary. This function disables a vocabulary so it is no longer used by the speech recognition engine to decode speech to text during a recognition session. Only the specified vocabulary is disabled. Any other enabled vocabularies remain active. This function is valid only when the speech recognition engine is not decoding speech to text. For examples of conditions when the engine is not decoding speech to text, see "Setting Up Vocabularies" in the SAPI Developer's Guide.

#### Syntax

```
int SmDisableVocab ( char    *vocab,
                    SM_MSG *reply );
```

#### Parameters

*vocab*           input - The name of the vocabulary to be disabled.

*reply*           input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOC
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_BAD_VOCAB
SM_RC_NOT_VALID_REQUEST
```

## Reply Structure Functions by Message Type

SM\_DISABLE\_VOCAB\_REPLY

SmGetRc

SmGetVocabName

## Task Related Functions and Callbacks

SmAddToVocab

SmDefineVocab

SmEnableVocab

SmQueryAddedWords

SmQueryEnabledVocabs

SmQueryVocabs

SmQueryWord

SmRemoveFromVocab

SmUndefineVocab

SmNaddToVocabCallback

SmNdefineVocabCallback

SmNdisableVocabCallback

SmNenableVocabCallback

SmNqueryAddedWordsCallback

SmNqueryEnabledVocabsCallback

SmNqueryVocabsCallback

SmNqueryWordsCallback

SmNremoveFromVocabCallback

SmNundefineVocabCallback

### 3.18 SmDiscardData

#### Purpose

SmDiscardData discards audio and error-correction data. This function discards audio data and error-correction information for recognized words that are not referenced again, thus conserving speech recognition engine disk space. ( Note that this data can also be deleted by specifying the SmNdiscardSessionData attribute when calling SmDisconnect. See [Section 3.21 \[SmDisconnect\], page 57.](#) )

SmDiscardData works on the granularity of utterances (between SmMicOn and SmMicOff) and discards utterance files owned by the application. Ownership is established by the application that turned the microphone on to create the utterance. An application can guarantee ownership of its data by toggling the microphone Off and On when it receives focus.

#### Syntax

```
int SmDiscardData ( SM_MSG *reply );
```

#### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOC
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_NOT_VALID_REQUEST
```

#### Reply Structure Functions by Message Type

```
SM_DISCARD_DATA_REPLY
SmGetRc
```

### 3.19 SmDiscardSpeechData

#### Purpose

SmDiscardSpeechData will be called by a client application to specify audio data that can be deleted. The total number of tags which may be specified is limited to SM\_MAX\_API\_TAGS (16000). If the tags specified cover the entire utterance, the files associated with that utterance will be deleted. When the speech data is saved (SmSaveSpeechData) those sections of remaining .wav files associated with the tags marked as discarded will not be saved. Any tags marked as discarded will no longer be usable.

#### Syntax

```
int SmDiscardSpeechData ( unsigned long flags,
                          short ntags,
                          long stags[],
                          long etags[],
                          SM_MSG *reply );
```

#### Parameters

<i>flags</i>	input - Reserved, must be set to 0.
<i>ntags</i>	input - Specifies number of tags in tag arrays. Valid values 1 - SM_MAX_API_TAGS
<i>stags</i>	input - Array of starting tags
<i>etags</i>	input - Array of ending tags
<i>reply</i>	input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOC
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
```



## Reply Structure Functions by Message Type

SM\_DISCARD\_SPEECH\_DATA\_REPLY

SmGetStatus

SmGetFlags

## Task Related Functions and Callbacks

SmQuerySpeechData

SmRestoreSpeechData

SmSaveSpeechData

SmNdiscardSpeechDataCallback

SmNquerySpeechDataCallback

SmNrestoreSpeechDataCallback

SmNsaveSpeechDataCallback

## 3.20 SmDiscardUtterance

### Purpose

SmDiscardUtterance deletes all audio files associated with an utterance.

### Syntax

```
int SmDiscardUtterance ( long    uttno,
                        SM_MSG *reply );
```

### Parameters

*uttno*        input - Utterance number to be deleted

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_ENROLLMENT\_NOT\_COMPLETE  
SM\_RC\_BAD\_UTTNO

### Reply Structure Functions by Message Type

SM\_DISCARD\_UTTERANCE\_REPLY

### Task Related Functions and Callbacks

SmQueryUtterances  
SmNdiscardUtteranceCallback

## 3.21 SmDisconnect

### Purpose

SmDisconnect disconnects from the speech recognition engine. This function closes the communication connection with the speech recognition engine. Any asynchronous engine messages left in the application message queue after the SmDisconnect() call returns are no longer valid and should not be used in any further SMAPI calls.

If the application needs to reconnect to the engine after a call to SmDisconnect(), then any engine messages remaining in the application message queue for the previously connected session must be removed before calling SmConnect().

### Syntax

```
int SmDisconnect ( int      nargs,
                  SmArg  *Args,
                  SM_MSG *reply );
```

### Parameters

*nargs*        input - The number of arguments in the accompanying argument list.

*Args*        input - A set of arguments that indicate default parameters for disconnecting from the speech recognition engine. Different attributes can be specified depending on the type of session and the disposition of current data. Arguments include:

In recognition sessions:

*SmNdiscardSessionAdaptation*

Reset the user's voice model to the state it was in before the decoding session.

*SmNdiscardSessionData*

Delete any data for this session.

*SmNsaveSessionData*

Keep the session data until the user re-initializes. (Default)

*SmNsaveSessionAdaptation*

Keep words added to the word-usage model during this session. (Default)

In enrollment sessions:

*SmNcompleteEnrollment*

Start the training program after disconnecting from the enrollment session.

*SmNsuspendEnrollment*

Do not start the training program after disconnecting from the enrollment session.

*reply*      input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_BAD\_ENROLLID  
SM\_BAD\_SCRIPT  
SM\_RC\_BAD\_VALUE  
SM\_RC\_ENROLLID\_EXISTS  
SM\_RC\_ENROLLID\_RUNNING  
SM\_RC\_NO\_SPACE\_TERM\_ENROLL  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_ERROR

## Reply Structure Functions by Message Type

SM\_DISCONNECT\_REPLY  
SmGetMsgType  
SmGetRc  
SmGetService

## Task Related Functions and Callbacks

SmClose  
SmNdisconnectCallback

## 3.22 SmDispatch

### Purpose

SmDispatch receives one message and dispatches the callbacks. This function provides the method through which callback functions are executed by Windows applications. More specifically, this function receives one message from the speech recognition engine and dispatches the appropriate callback routines, which were previously registered using SmAddCallback, for the default connection.

### Syntax

```
int SmDispatch ( unsigned long ap_val );
```

### Parameters

*ap\_val*        input - Windows applications need to specify lParam here for ap\_val. Non-Windows applications should set this parameter to 0.

### Return Values

SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_EINVAL  
SM\_RC\_EMSGSIZE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOHANDLES  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_ETIMEOUT  
SM\_RC\_EUNEXP  
SM\_RC\_EUNKMSG  
SM\_RC\_NOT\_OPEN  
SM\_RC\_OK

### Task Related Functions and Callbacks

SmAddCallback  
SmRemoveCallback

## 3.23 SmEnableVocab

### Purpose

SmEnableVocab enables a defined vocabulary. This function enables a vocabulary to be used by the speech recognition engine to decode speech to text during a recognition session. Currently enabled vocabularies are not disabled by this function but remain enabled along with the newly enabled vocabulary. If currently enabled vocabularies are to be disabled first, then an SmDisableVocab call must be made beforehand for each vocabulary to be disabled.

This call is valid only when the speech recognition engine is not decoding speech to text.

### Syntax

```
int SmEnableVocab ( char *vocab,
                   SM_MSG *reply );
```

### Parameters

*vocab*            input - The name of the vocabulary to be enabled.

*reply*           input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_VOCAB  
SM\_RC\_NOT\_VALID\_REQUEST

## Reply Structure Functions by Message Type

SM\_ENABLE\_VOCAB\_REPLY

SmGetRc

SmGetVocabName

## Task Related Functions and Callbacks

SmAddToVocab

SmDefineVocab

SmDisableVocab

SmQueryAddedWords

SmQueryEnabledVocabs

SmQueryVocabs

SmQueryWord

SmRemoveFromVocab

SmUndefineVocab

SmNaddToVocabCallback

SmNdefineVocabCallback

SmNdisableVocabCallback

SmNenableVocabCallback

SmNqueryAddedWordsCallback

SmNqueryEnabledVocabsCallback

SmNqueryVocabsCallback

SmNqueryWordsCallback

SmNremoveFromVocabCallback

SmNundefineVocabCallback

## 3.24 SmEventNotify

### Purpose

SmEventNotify requests notification when the speech recognition engine completes decoding all the audio dictated up to the time SmEventNotify was called. This function causes the callback associated with the SmNeventSynchCallback to be called as soon as decoding is completed for all the words dictated up to the time SmEventNotify was called. If a callback is not used, the application is notified with the reply message structure SM\_EVENT\_SYNCH sent by the speech recognition engine.

### Syntax

```
int SmEventNotify ( long    event_id,
                   long    options,
                   SM_MSG *reply );
```

### Parameters

*event\_id* input - The event ID.

*options* input - The options for the event, which can be logically OR'ed:

- SM\_EVENT\_HALT\_RECOGNITION - Halt and switch to command recognition when event is encountered (Default)
- SM\_EVENT\_CONTINUE\_RECOGNITION - Notify application that event was encountered and continue with recognition
- SM\_EVENT\_FIRM\_UP - Process audio preceding event
- SM\_EVENT\_DISCARD - Discard audio preceding event (Default)

**Please Note:**

- SM\_EVENT\_HALT\_RECOGNITION and SM\_EVENT\_CONTINUE\_RECOGNITION cannot be specified together.
- SM\_EVENT\_FIRM\_UP and SM\_EVENT\_DISCARD cannot be specified together.

*reply* input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOC
SM_RC_EBADHANDLE
```



SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_NOT\_VALID\_REQUEST

## **Reply Structure Functions by Message Type**

SM\_EVENT\_NOTIFY\_REPLY  
SmGetEventId  
SmGetRc  
SM\_EVENT\_SYNCH  
SmGetEventId  
SmGetEventOptions  
SmGetRc

## **Task Related Functions and Callbacks**

SmNeventNotifyCallback  
SmNeventSynchCallback

## 3.25 SmEventNotifyEx

### Purpose

SmEventNotifyEx is an extension of the function SmEventNotify. It allows an application to request notification for an arbitrary time.

This function can be used by an application to create bookmarks. The application can first call the SmEventTime function to get the current time. It can then call SmEventNotifyEx passing it the time returned by SmEventTime. The engine will send notification when speech has been processed which corresponds to this time.

### Syntax

```
int SmEventNotifyEx ( long          event_id,
                     unsigned long  flags,
                     unsigned long  time,
                     SM_MSG         *reply );
```

### Parameters

*event\_id*     input - The event ID.

*flags*        input - The options for the event, which can be logically OR'ed:

- SM\_EVENT\_HALT\_RECOGNITION - Halt and switch to command recognition when event is encountered (Default)
- SM\_EVENT\_CONTINUE\_RECOGNITION - Notify application that event was encountered and continue with recognition
- SM\_EVENT\_FIRM\_UP - Process audio following event
- SM\_EVENT\_DISCARD - Discard audio preceding event (Default)

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOC
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
```

SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN

## Reply Structure Functions by Message Type

SM\_EVENT\_NOTIFY\_REPLY  
SmGetEventId  
SmGetTimes  
SmGetFlags

## Task Related Functions and Callbacks

SmEventNotify  
SmEventTime  
SmNeventNotifyCallback

## 3.26 SmEventTime

### Purpose

SmEventTime allows an application to get the current time.

### Syntax

```
int SmEventTime ( long    event_id,  
                  long    options,  
                  SM_MSG *reply );
```

### Parameters

*flags* input - Reserved.

*reply* input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN

### Reply Structure Functions by Message Type

SM\_EVENT\_TIME\_REPLY  
SmGetTimes  
SmGetFlags

### Task Related Functions and Callbacks

SmEventNotify  
SmEventNotifyEx  
SmNeventTimeCallback

## 3.27 SmHaltRecognizer

### Purpose

SmHaltRecognizer temporarily halts recognition. This function can be called when the speech recognition engine is decoding speech to text. Halting recognition may be necessary, for example, to define and/or enable vocabularies to be used by the engine during recognition. Recognition can be restarted with SmRecognizeNextWord.

### Syntax

```
int SmHaltRecognizer ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to the reply structure or to the SmAsynchronous value.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_ERROR

### Reply Structure Functions by Message Type

SM\_HALT\_RECOGNIZER\_REPLY  
SmGetNumberWordMsgs  
SmGetRc

## Task Related Functions and Callbacks

SmMicOff  
SmMicOn  
SmQuery  
SmRecognizeNextWord  
SmSet  
SmNhaltRecognizerCallback  
SmNmicOffCallback  
SmNmicOnCallback  
SmNqueryCallback  
SmNrecognizeNextWordCallback  
SmNrecognizedTextCallback  
SmNrecognizedWordCallback  
SmNsetCallback  
SmNutteranceCompletedCallback

## 3.28 SmMicOff

### Purpose

SmMicOff turns off the microphone. This function turns off the microphone; however, in a recognition session, after the microphone is turned off, the speech recognition engine continues speech to text decoding of words already spoken. Depending on the speed and what is said before the microphone is turned off, this process can take several seconds to complete.

### Syntax

```
int SmMicOff ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_MIC\_ALREADY\_OFF  
SM\_RC\_MIC\_OFF\_PENDING  
SM\_RC\_NOT\_VALID\_REQUEST

### Reply Structure Functions by Message Type

SM\_MIC\_OFF\_REPLY  
SmGetRc

## Task Related Functions and Callbacks

SmHaltRecognizer  
SmMicOn  
SmQuery  
SmRecognizeNextWord  
SmSet  
SmNhaltRecognizerCallback  
SmNmicOffCallback  
SmNmicOnCallback  
SmNqueryCallback  
SmNrecognizeNextWordCallback  
SmNrecognizedTextCallback  
SmNrecognizedWordCallback  
SmNsetCallback  
SmNutteranceCompletedCallback



## 3.29 SmMicOn

### Purpose

SmMicOn turns on the microphone. In a recognition session, this function turns on the microphone and starts the audio stream. The application must issue an SmRecognized-NextWord to start the engine decoding.

### Syntax

```
int SmMicOn ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to the reply structure or to the SmAsynchronous value.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOC  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEMS  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_AP  
SM\_RC\_BAD\_DECO  
SM\_RC\_BUSY\_LAST\_UTTERANCE  
SM\_RC\_BUSY\_WORD\_CORRECTION  
SM\_RC\_ENROLLMENT\_NOT\_COMPLETE  
SM\_RC\_MIC\_ALREADY\_ON  
SM\_RC\_MIC\_OFF\_PENDING  
SM\_RC\_MIC\_ON\_PENDING  
SM\_RC\_NO\_SPACE\_MIC\_ON  
SM\_RC\_NOT\_VALID\_REQUEST

## Reply Structure Functions by Message Type

SM\_MIC\_ON\_REPLY

SmGetRc

SmGetUtteranceNumber

During a recognition session:

SM\_RECOGNIZED\_TEXT

SmGetFirmWords

SmGetRc

SmGetTimes

## Task Related Functions and Callbacks

SmHaltRecognizer

SmMicOff

SmQuery

SmRecognizeNextWord

SmSet

SmNhaltRecognizerCallback

SmNmicOffCallback

SmNmicOnCallback

SmNqueryCallback

SmNrecognizeNextWordCallback

SmNrecognizedTextCallback

SmNrecognizedWordCallback

SmNsetCallback

SmNutteranceCompletedCallback

### 3.30 SmNewContext

#### Purpose

SmNewContext sends a new text context. This function allows a change in text context sent to the speech recognition engine. This function is usually called when text dictation begins at a new location in a document and can only be called when the speech recognition engine is halted. Context refers to previous and following words, which are used to decode the current word in a recognition session.

#### Syntax

```
int SmNewContext ( short nwords,
                  SM_WORD *words[],
                  SM_MSG *reply );
```

#### Parameters

- nwords*      input - The number of words of context sent. Zero means no left context.
- words*        input - An array of pointers to the context words, ordered from left to right in the text.
- reply*        input/output - The pointer to the reply structure or to the SmAsynchronous value indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
```

## Reply Structure Functions by Message Type

SM\_NEW\_CONTEXT\_REPLY

SmGetRc

## Task Related Functions and Callbacks

SmCorrectText

SmCorrectTextCancel

SmRemovePronunciation

SmWordCorrection

SmNcorrectTextCallback

SmNcorrectTextCancelCallback

SmNnewContextCallback

SmNremovePronunciationCallback

SmNwordCorrectionCallback

### 3.31 SmNewContextEx

#### Purpose

SmNewContextEx sends a new text context. This function is an extension of the function SmNewContext.

This function allows a change in text context sent to the speech recognition engine. This function is usually called when text dictation begins at a new location in a document and can only be called when the speech recognition engine is halted. Context refers to previous (left) and following (right) words, which are used to decode the current word in a recognition session. This function allows an application to specify left and right context, as well as a list of words between the left and right context to be ignored by the recognition engine when setting the context

#### Syntax

```
int SmNewContextEx ( const char *vocab,
                    int          numleft,
                    const char *lefwords[],
                    int          numexclude,
                    const char *excludephrases[],
                    int          numright,
                    const char *rightwords[],
                    SM_MSG      *reply );
```

#### Parameters

- vocab*        input - Specifies the vocabulary of the new context.
- numleft*     input - The number of left context words.
- lefwords*    input - An array of pointers to the left context words, ordered from left to right in the text.
- numexclude*   input - The number of exclude phrases.
- excludephrases* input - An array of pointers to the excluded phrases.
- numright*    input - The number of right context words.
- rightwords*   input - An array of pointers to the right context words, ordered from left to right in the text.
- reply*        input/output - The pointer to the reply structure or to the SmAsynchronous value indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_VALID\_REQUEST

## Reply Structure Functions by Message Type

SM\_NEW\_CONTEXT\_REPLY  
SmGetRc

## Task Related Functions and Callbacks

SmCorrectText  
SmCorrectTextCancel  
SmRemovePronunciation  
SmWordCorrection  
SmNcorrectTextCallback  
SmNcorrectTextCancelCallback  
SmNnewContextCallback  
SmNremovePronunciationCallback  
SmNwordCorrectionCallback

## 3.32 SmOpen

### Purpose

SmOpen opens the SMAPI. This function establishes a SMAPI connection and initializes values in a connection structure.

### Syntax

```
int SmOpen ( int nargs,
             SmArg *Args );
```

### Parameters

- nargs*        input - The number of arguments in the accompanying argument list.
- Args*        input - A set of arguments that indicate default parameters for the SMAPI connection. They are stored in a connection structure for the speech-aware application. The values in the structure are used to set up the session with the speech recognition engine. SMAPI attributes can also be set by using the SmSetArg function, which specifies the attribute name and its value. For details on establishing a speech session, see the SMAPI Developer's Guide. For details on the attributes, see [Chapter 7 \[Attributes\]](#), page 313.

### Return Values

SM\_RC\_EALLOC  
SM\_RC\_ENOMEM  
SM\_RC\_NAVIGATOR\_ALREADY\_DEFINED  
SM\_RC\_OK  
SM\_RC\_ALREADY\_CONNECTED  
SM\_RC\_ALREADY\_OPENED  
SM\_RC\_OPEN\_SYNCH\_QUEUE\_FAILED  
SM\_RC\_NOT\_VALID\_REQUEST

### Task Related Functions and Callbacks

SmConnect  
SmSetArg  
SmNconnectCallback

### 3.33 SmPlayMessage

#### Purpose

SmPlayMessage plays back a prerecorded audio file. This function allows the user to play a prerecorded audio file. After the speech-aware application requests a playback, it can call SmGetStatus to retrieve the last playback status, which can be one of the following:

- SM\_STAT\_BAD\_AUDIO - The connection to the audio source was lost during playback.
- SM\_STAT\_PLAY\_START - The message has started playing.
- SM\_STAT\_PLAY\_STOP - The message has stopped playing.

Possible return values passed to the SmNplayMessageStatusCallback are:

- SM\_RC\_OK
- SM\_RC\_BAD\_AUDIO
- SM\_RC\_PLAY\_OPEN\_ERROR

The following restrictions apply when calling SmPlayMessage:

- Microphone must be off.
- Speech recognition engine must not be decoding speech to text. In other words, the application must have received SM\_UTTERANCE\_COMPLETED from the engine.

#### Syntax

```
int SmPlayMessage ( char    *message_name,
                    char    *language,
                    SM_MSG *reply );
```

#### Parameters

*message\_name*

input - The fully qualified name of an audio file in IBM ViaVoice format, which is 8, 11, or 22 kHz ADPCM WAV file format. These audio files can be created by dictating the message and saving to a file in the IBM ViaVoice user path.

*language*    input - The name of the language played.

*reply*        input/output - The pointer to the reply structure or to the SmAsynchronous value indicating that the call is made asynchronously.



## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_LANGUAGE  
SM\_RC\_BAD\_MESSAGE  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_NOT\_WHILE\_MIC\_ON

## Reply Structure Functions by Message Type

SM\_PLAY\_MESSAGE\_REPLY  
SmGetRc  
SM\_PLAY\_MESSAGE\_STATUS  
SmGetMsgName  
SmGetRc  
SmGetStatus

## Task Related Functions and Callbacks

SmCancelPlayback  
SmPlayUtterance  
SmPlayWords  
SmNcancelPlaybackCallback  
SmNplayMessageCallback  
SmNplayMessageStatusCallback  
SmNplayUtteranceCallback  
SmNplayUtteranceStatusCallback  
SmNplayWordsCallback  
SmNplayWordsStatusCallback

### 3.34 SmPlayUtterance

#### Purpose

SmPlayUtterance plays back a spoken utterance. This function allows an utterance or a portion of an utterance previously spoken during dictation or enrollment to be played back to the user as an error-correction aid. For a monitored enrollment session, an utterance corresponds to a sentence of an enrollment script and, for a recognition session, it corresponds to any word spoken between SmMicOn and subsequent SmMicOff.

After the speech-aware application requests a playback it can call SmGetStatus to retrieve the last playback status, which can be one of the following:

- SM\_STAT\_BAD\_AUDIO - The connection to the audio source was lost during playback.
- SM\_STAT\_PLAY\_START - The utterance has started playing.
- SM\_STAT\_PLAY\_STOP - The utterance has stopped playing.

Possible return values passed to the SmNplayUtteranceStatusCallback are:

- SM\_RC\_OK
- SM\_RC\_BAD\_AUDIO
- SM\_RC\_PLAY\_OPEN\_ERROR

The following restrictions apply when calling SmPlayUtterance:

- Microphone must be off.
- Speech recognition engine must not be decoding speech to text. In other words, the application must have received SM\_UTTERANCE\_COMPLETED from the engine.

**Please note:** For this call to be valid, the application must first enable audio saving through SmSet(SM\_SAVE\_AUDIO, SM\_SAVE\_AUDIO\_DEFAULT, &reply).

#### Syntax

```
int SmPlayUtterance( long    uttno,
                    long    begtime,
                    long    endtime,
                    SM_MSG *reply );
```

#### Parameters

<i>uttno</i>	input - The utterance number to play back. The current utterance number can be extracted from SM_MIC_ON_REPLY with SmGetUtteranceNumber.
<i>begtime</i>	input - Reserved. Must be set to 0.
<i>endtime</i>	input - Reserved. Must be set to 0.
<i>reply</i>	input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_NOT\_WHILE\_MIC\_ON

## Reply Structure Functions by Message Type

SM\_PLAY\_UTTERANCE\_REPLY  
SmGetRc  
SM\_PLAY\_UTTERANCE\_STATUS  
SmGetRc  
SmGetStatus  
SmGetUtteranceNumber

## Task Related Functions and Callbacks

SmCancelPlayback  
SmPlayMessage  
SmPlayWords  
SmNcancelPlaybackCallback  
SmNplayMessageCallback  
SmNplayMessageStatusCallback  
SmNplayUtteranceCallback  
SmNplayUtteranceStatusCallback  
SmNplayWordsCallback  
SmNplayWordsStatusCallback

### 3.35 SmPlayWords

#### Purpose

SmPlayWords plays back spoken words. This function allows a previously spoken word or collection of words to be played back to the user as an error correction aid.

The SmPlayWords call generates one or more SM\_PLAY\_WORDS\_STATUS messages. When this message is received, the application can call SmGetStatus to retrieve the status of the last playback, which can be one of the following:

- SM\_STAT\_BAD\_AUDIO - The connection to the audio source was lost during playback.
- SM\_STAT\_BAD\_TAG - An invalid word (tag) was specified for the played word.
- SM\_STAT\_PLAY\_START - The word has started playing.
- SM\_STAT\_PLAY\_STOP - The word has stopped playing.

Possible return values passed to the SmNplayWordsStatusCallback are:

- SM\_RC\_OK
- SM\_RC\_BAD\_TAG
- SM\_RC\_PLAY\_OPEN\_ERROR

The following restrictions apply when calling SmPlayWords:

- Microphone must be off.
- Speech recognition engine must not be decoding speech to text. In other words, the application must have received SM\_UTTERANCE\_COMPLETED from the engine.

**Please note:** For this call to be valid, the application must first enable audio saving through SmSet(SM\_SAVE\_AUDIO, SM\_SAVE\_AUDIO\_DEFAULT, &reply).

#### Syntax

```
int SmPlayWords ( short   ntags,
                  long    tags[],
                  long    options,
                  SM_MSG *reply );
```

#### Parameters

*ntags*        input - The number of words (tags) to play back.

*tags*        input - The array of word tags.

*options*     input - One of the following:

*SM\_PLAY\_WORDS\_CONTIGUOUS*

When playing multiple words, return a single SM\_PLAY\_WORDS\_STATUS message at the beginning (SM\_STAT\_PLAY\_START) of each word

played, and return a single message (SM\_STAT\_PLAY\_STOP) after all words have been played. This is the default.

*SM\_PLAY\_WORDS\_SAVE\_WAVFILE*

Instead of sending audio to the output device, this flag saves the audio data as a wave(WAV) file. The name of the wave file must be set via an SmSetDirectory call prior to calling the SmPlayWords function. If the SM\_PLAY\_WORDS\_WITH\_SILENCE flag is also set, then the entire utterance(s) corresponding to the input tag array will be written to the wave file.

*SM\_PLAY\_WORDS\_SEPARATE*

When playing multiple words, return separate SM\_PLAY\_WORDS\_STATUS messages at the beginning (SM\_STAT\_PLAY\_START) of each word played and a single message (SM\_STAT\_PLAY\_STOP) when the entire list of tags has been played. If playback is canceled, the SM\_STAT\_PLAY\_STOP is also sent to indicate completion of the interrupted playback request.

*SM\_PLAY\_WORDS\_WITH\_SILENCE*

When playing multiple words, playback the silence between words. The default is to play only the non-silence portion of recognized words.

*reply*      input/output - The pointer to a reply structure or to SmAsynchronous, indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
 SM\_RC\_EALLOCS  
 SM\_RC\_EBADHANDLE  
 SM\_RC\_ENOCONN  
 SM\_RC\_ENOMEM  
 SM\_RC\_ENOMSG  
 SM\_RC\_ENOSERVER  
 SM\_RC\_EUNEXP  
 SM\_RC\_SM\_NOT\_OPEN  
 SM\_RC\_OK  
 SM\_RC\_INVALID\_PARM\_MAX\_LEN  
 SM\_RC\_BAD\_TAG  
 SM\_RC\_NOT\_VALID\_REQUEST  
 SM\_RC\_NOT\_WHILE\_MIC\_ON

## Reply Structure Functions by Message Type

SM\_PLAY\_WORDS\_REPLY  
SmGetRc  
SM\_PLAY\_WORDS\_STATUS  
SmGetRc  
SmGetStatus  
SmGetTags

## Task Related Functions and Callbacks

SmCancelPlayback  
SmPlayMessage  
SmPlayUtterance  
SmQueryWord  
SmWordCorrection  
SmNcancelPlaybackCallback  
SmNplayMessageCallback  
SmNplayMessageStatusCallback  
SmNplayUtteranceCallback  
SmNplayUtteranceStatusCallback  
SmNplayWordsCallback  
SmNplayWordsStatusCallback  
SmNqueryAddedWordsCallback

### 3.36 SmQuery

#### Purpose

SmQuery queries a speech recognition engine parameter. This function queries the value of the specified speech recognition engine parameter. Call the SmGetItemValue function to extract the queried value.

#### Syntax

```
int SmQuery ( short item,
              SM_MSG *reply );
```

#### Parameters

*item*            input - The parameter queried, which can be one of the following:

*SM\_API\_DEBUG*

Values include the following: 0 is debugging off, 1 to 5 provide progressively more debug information. A value of 1 or 2 provides flow of control information.

*SM\_API\_DISPLAY*

Reserved

*SM\_API\_LOG*

Reserved

*SM\_API\_TIMING*

Reserved

*SM\_AUDIO\_CONFIGURATION*

(Not applicable to Windows. Instead use Audio Setup program.)  
Query the input source of the audio hardware where audio data is collected. The returned value is bit mapped and can be any combination of the following values:

SM\_AUDIO\_INPUT\_MIC\_HI\_GAIN

SM\_AUDIO\_INPUT\_MIC\_LO\_GAIN

SM\_AUDIO\_INPUT\_LINE\_LEFT

SM\_AUDIO\_INPUT\_LINE\_RIGHT

SM\_AUDIO\_INPUT\_VARIABLE\_GAIN

SM\_AUDIO\_OUTPUT\_LINE\_LEFT

SM\_AUDIO\_OUTPUT\_LINE\_RIGHT

SM\_AUDIO\_OUTPUT\_INTERNAL\_SPEAKER

SM\_AUDIO\_OUTPUT\_VARIABLE\_GAIN

Refer to the SMLIMITS.H file for bounds on input and output values.

#### *SM\_AUDIO\_DEVICE*

Query the type of audio hardware. Return values include the following:

- SM\_ACPA\_AUDIO - Reserved
- SM\_PERSONAL\_DICTATION\_AUDIO - ???
- SM\_SYSTEM\_AUDIO - Native audio system for hardware/operating systems.

#### *SM\_AUDIO\_INPUT\_MODE*

(Not applicable to Windows. Instead use Audio Setup program.) Query the input source of the audio hardware where audio data is collected. We suggest you use the supplied AUDIO.DLL instead. The returned value is bit mapped and can be any one of the following values:

SM\_AUDIO\_INPUT\_LINE\_LEFT  
 SM\_AUDIO\_INPUT\_LINE\_RIGHT  
 SM\_AUDIO\_INPUT\_MIC\_HI\_GAIN  
 SM\_AUDIO\_INPUT\_MIC\_LO\_GAIN

#### *SM\_AUDIO\_INPUT\_GAIN*

(Not applicable to Windows. Instead use Audio Setup program.) If supported by underlying audio, this parameter allows you to query current gain setting.

#### *SM\_AUDIO\_OUTPUT\_GAIN*

(Not applicable to Windows. Instead use Audio Setup program.) If supported by underlying audio, this parameter allows you to query current gain setting.

#### *SM\_AUDIO\_OUTPUT\_MODE*

(Not applicable to Windows. Instead use Audio Setup program.) If supported by underlying audio, this bit value indicates the current output destination. We suggest you use the supplied audio.dll instead. The returned value can be any one of the following values:

SM\_AUDIO\_OUTPUT\_LINE\_LEFT  
 SM\_AUDIO\_OUTPUT\_LINE\_RIGHT  
 SM\_AUDIO\_OUTPUT\_LINE\_INTERNAL\_SPEAKER

#### *SM\_COMPLETE\_COMMAND\_TIMEOUT*

These timeout parameters are used to control the behavior of finite state grammars. In particular, they specify how much silence is needed at the end of commands before the engine will make a recognition decision to accept or reject the command. The amount



of silence required to accept or reject a phrase that is unambiguously complete. Default value is 25 (csecs). Can be overridden by task dependent .par values.

*SM\_DELAY\_EXIT*

Query the amount of time the engine waits before terminating after the last client disconnects. Valid values: -1 - MAX\_LONG. If set to -1 the engine will not terminate after the last client disconnects. If set to 0 the engine will terminate immediately after the last client disconnects. For values of x where  $0 < x \leq \text{MAX\_LONG}$ , the engine will wait x seconds before terminating after the last client disconnects.

*SM\_ENABLE\_EXCLUSIVE\_VOCABS*

Query the calling application's vocabularies. The value is either TRUE (1) for enabled, or FALSE (0) for disabled.

*SM\_REJECTION\_THRESHOLD*

Query the engine's threshold for rejecting a recognized phrase. Returns value from SM\_MIN\_REJECTION\_THRESHOLD to SM\_MAX\_REJECTION\_THRESHOLD.

*SM\_NOTIFY\_AUDIO\_LEVEL*

Query the returning of audio-level data during recognition or enrollment. The value is either TRUE (1) or FALSE (0).

*SM\_NOTIFY\_COMMAND\_WORD*

Query whether the application is notified when a command word is recognized by having the engine send a SM\_COMMAND\_WORD reply message. The value is either TRUE (1) or FALSE (0).

*SM\_NOTIFY\_ENGINE\_STATE*

Query whether the application is notified of a speech recognition engine state change. The value is either TRUE (1) or FALSE (0).

*SM\_NOTIFY\_FOCUS\_STATE*

Query whether the application is notified of a speech focus state change. The value is either TRUE (1) or FALSE (0).

*SM\_NOTIFY\_MIC\_STATE*

Query whether the application is notified of a microphone state change. The value is either TRUE (1) or FALSE (0).

*SM\_PARTIAL\_COMMAND\_TIMEOUT*

These timeout parameters are used to control the behavior of finite state grammars. In particular, they specify how much silence is needed at the end of commands before the engine will make a recognition decision to accept or reject the command. The amount of silence required to accept or reject a phrase that is both partially complete and complete. Default value is 250 (csecs). It can be overridden by task dependent .par value. For example, consider the grammar: command = move up 3 move up five Now consider

the following acoustic scenarios: 1.MOVE 2.MOVE UP 3.MOVE UP FIVE 1.The engine will accept silences up to 0.5 seconds, waiting for the UP, before rejecting the phrase as incomplete. 2.The engine will accept silences up to 0.5 seconds, waiting for the optional FIVE, before accepting the phrase MOVE UP. 3.Then engine will wait only 0.1 seconds after the FIVE to accept the phrase. For the example given, it seems that you would like to increase the `SM_PARTIAL_COMMAND_TIMEOUT` in order to allow longer pauses within phrases. Note that increasing this parameter will also increase the decision time for "ambiguously complete" phrases.

#### *SM\_PHRASE\_ALTERNATIVES*

Queries the maximum number of alternatives which can be returned by the engine when querying phrase alternatives. See [Section 3.45 \[SmQueryPhraseAlternatives\]](#), page 105.

#### *SM\_REDUCED\_CPU\_MODE*

Query the CPU mode of the speech recognition engine. The value is either TRUE (1) for reduced CPU mode, or FALSE (0) for normal CPU mode. See [Section 4.12 \[SmGetEngineState\]](#), page 211.

#### *SM\_SAVE\_AUDIO*

Query whether recorded audio is being saved during recognition. The value is either TRUE (1) or FALSE (0). The returned value is a 32 bit variable containing one or more of the following flags set:

##### *SM\_SAVE\_AUDIO\_DEFAULT*

(`SM_SAVE_AUDIO_PLAYBACK`, `SM_SAVE_AUDIO_ALTERNATES`, `SM_SAVE_AUDIO_TRAINWORD`)

##### *SM\_SAVE\_AUDIO\_PLAYBACK*

Saves the files required for playback.

##### *SM\_SAVE\_AUDIO\_ALTERNATES*

Saves the files required for querying alternates.

##### *SM\_SAVE\_AUDIO\_TRAINWORD*

Saves the files required for adding a pronunciation.

##### *SM\_SAVE\_AUDIO\_ADAPTATION*

Saves the files required for training.

#### *SM\_SILENCE\_DETECTION*

Query silence detection setting. Valid values: 0, 1. If set to 1, silence detection enabled, if set to 0, silence detection disabled.

*reply* input/output - The pointer to a reply structure or to `SmAsynchronous` indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_BAD\_MODE  
SM\_RC\_NOT\_YET  
SM\_RC\_EXISTS\_IN\_NOTIFY  
SM\_RC\_NOT\_IN\_NOTIFY  
SM\_RC\_NOT\_VALID\_REQUEST

## Reply Structure Functions by Message Type

SM\_QUERY\_REPLY  
SmGetItemValue  
SmGetRc

## Task Related Functions and Callbacks

SmHaltRecognizer  
SmMicOff  
SmMicOn  
SmRecognizeNextWord  
SmSet  
SmNhaltRecognizerCallback  
SmNmicOffCallback  
SmNmicOnCallback  
SmNqueryCallback  
SmNrecognizeNextWordCallback  
SmNrecognizedTextCallback  
SmNrecognizedWordCallback  
SmNsetCallback  
SmNutteranceCompletedCallback

## 3.37 SmQueryAddedWords

### Purpose

SmQueryAddedWords queries added words. This function requests a list of all words that have been added to vocabularies of the currently active session.

### Syntax

```
int SmQueryAddedWords ( char *vocab,  
                        SM_MSG *reply );
```

### Parameters

*vocab*           input - The name of the vocabulary to query.

*reply*           input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_VOCAB  
SM\_RC\_NOT\_VALID\_REQUEST

### Reply Structure Functions by Message Type

SM\_QUERY\_ADDED\_WORDS\_REPLY  
SmGetRc  
SmGetWords

## Task Related Functions and Callbacks

SmAddToVocab  
SmDefineVocab  
SmDisableVocab  
SmEnableVocab  
SmNaddToVocabCallback  
SmNdefineVocabCallback  
SmNdisableVocabCallback  
SmNenableVocabCallback  
SmNplayWordsCallback  
SmNplayWordsStatusCallback  
SmNqueryAddedWordsCallback  
SmNqueryEnabledVocabsCallback  
SmNqueryVocabsCallback  
SmNqueryWordsCallback  
SmNremoveFromVocabCallback  
SmNundefineVocabCallback  
SmNwordCorrectionCallback  
SmPlayWords  
SmQueryAddedWords  
SmQueryEnabledVocabs  
SmQueryVocabs  
SmQueryWord  
SmRemoveFromVocab  
SmUndefineVocab  
SmWordCorrection

### 3.38 SmQueryAddedWordsEx

#### Purpose

SmQueryAddedWordsEx extends SmQueryAddedWords. This new function accepts an argument which specifies a starting point. It will return words beginning from this starting point up to the limit SM\_MAX\_WORDS (500). If the specified starting point is greater than the number of added words, no words will be returned. Calling SmQueryAddedWords is equivalent to calling SmQueryAddedWordsEx with a start argument of 0. If there are more added words than the limit SM\_MAX\_WORDS, this function may be called repeatedly, specifying a start argument of 0 the first time and incrementing the start argument by the number of words returned. When 0 words are returned, all added words have been retrieved.

#### Syntax

```
int SmQueryAddedWordsEx ( unsigned long  flags,
                          unsigned long  start,
                          char            *vocab,
                          SM_MSG         *reply );
```

#### Parameters

*flags*        Reserved.

*start*       Specifies O-origin starting place.

*vocab*       Specifies the name of the vocabulary to query.

*reply*       SmAsynchronous or the pointer to a reply structure.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
SM_RC_BAD_VOCAB
```

## Reply Structure Functions by Message Type

SM\_QUERY\_ADDED\_WORDS\_REPLY

SmGetWords

## Task Related Functions and Callbacks

SmAddToVocab

SmDefineVocab

SmDisableVocab

SmEnableVocab

SmNaddToVocabCallback

SmNdefineVocabCallback

SmNdisableVocabCallback

SmNenableVocabCallback

SmNplayWordsCallback

SmNplayWordsStatusCallback

SmNqueryAddedWordsCallback

SmNqueryAddedWordsExCallback

SmNqueryEnabledVocabsCallback

SmNqueryVocabsCallback

SmNqueryWordsCallback

SmNremoveFromVocabCallback

SmNundefineVocabCallback

SmNwordCorrectionCallback

SmPlayWords

SmQueryAddedWords

SmQueryEnabledVocabs

SmQueryVocabs

SmQueryWord

SmRemoveFromVocab

SmUndefineVocab

SmWordCorrection

### 3.39 SmQueryAlternates

#### Purpose

SmQueryAlternates requests a list of alternative words. This function requests a list of alternative choices for a firm word or a group of firm words that has been incorrectly recognized. Errors are usually one-for-one substitutions, in which case only one tag needs to be specified on the call to SmQueryAlternates. A word may, however, be recognized incorrectly as two or more words. In that case, two or more tags might need to be specified in order to get an alternative list containing the correct word. The following restrictions apply when calling SmQueryAlternates:

- Speech recognition engine must not be decoding speech to text. In other words, the application must have received SM\_UTTERANCE\_COMPLETED from the engine.
- SM\_SAVE\_AUDIO must be enabled with SmSet.

#### Syntax

```
int SmQueryAlternates ( short   ntags,
                       long    tags[],
                       SM_MSG *reply );
```

#### Parameters

*ntags*        input - The number of words (tags) for which alternatives are requested.

*tags*        input - The tags of the word for which alternatives are requested.

*reply*       input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
```



## **Reply Structure Functions by Message Type**

SM\_QUERY\_ALTERNATES\_REPLY

SmGetAlternates

SmGetNextAlternate

SmGetRc

SmGetTags

## **Task Related Functions and Callbacks**

SmNqueryAlternatesCallback

### 3.40 SmQueryBinary

#### Purpose

SmQueryBinary queries the value of a specified speech recognition engine parameter. This function is used instead of SmQuery to query values of arbitrary length and data type.

#### Syntax

```
int SmQueryBinary ( short  item,
                   SM_MSG *reply );
```

#### Parameters

- item*           input - The parameter to be queried, which can be any of the following:
- SM\_AUDIO\_SOURCE*  
Specifies querying value in audio library. Value returned in reply depends on audio library implementation.
  - SM\_MNR\_VALUE*  
Specifies querying mnv value. Value returned in reply is mnv value.
  - SM\_SIGNAL\_NOISE*  
Specifies querying signal/noise ration values. Value returned in reply is character string with the following blank delimited signal/noise ratio values: signal to noise ratio in decibels, signal level in decibels, fraction of samples clipped.
- reply*           input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_NOT_VALID_REQUEST
```

## Reply Structure Functions by Message Type

SM\_QUERY\_BINARY\_REPLY

SmGetRc

SmGetBinaryItemValue

## Task Related Functions and Callbacks

SmSetBinary

SmNsetBinaryCallback

### 3.41 SmQueryDefault

#### Purpose

SmQueryDefault queries the default value for a user, enrollment, task ID speech attribute, or the default topics.

The default enrollid, taskid, and topics are stored on a per-user basis. SmQueryDefault called with an item value of SM\_DEFAULT\_USERID will return the value of the default userid. It can be accessed in the reply structure using the function SmGetUserId or SmGetUserIds. SmQueryDefault called with other item values will return default enrollid, taskid, or topics for the default userid. The value of the default enrollid can be accessed in the reply structure using the functions SmGetEnrollId or SmGetEnrollIds. The value of the default taskid can be accessed in the reply structure using the functions SmGetTask or SmGetTasks. The value of the default topics can be accessed in the reply structure using the function SmGetDefaultTopics. Since multiple default topics can be associated with a userid the default topics will be returned concatenated in a blank delimited string. To query the default enrollid, task or topics for a user other than the default user, the function SmQueryUserDefault is provided. See [Section 3.57 \[SmQueryUserDefault\]](#), page 127.

#### Syntax

```
int SmQueryDefault ( long    item,
                    SM_MSG *reply );
```

#### Parameters

- |              |   |
|--------------|---|
| <i>item</i>  | input - Type of default ID speech attribute. Valid values include the following: <ul style="list-style-type: none"> <li>• SM_DEFAULT_USERID</li> <li>• SM_DEFAULT_ENROLLID</li> <li>• SM_DEFAULT_TASK</li> <li>• SM_DEFAULT_TOPICS</li> </ul> |
| <i>reply</i> | input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.   |

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
```

SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK

## Reply Structure Functions by Message Type

SM\_QUERY\_DEFAULT\_REPLY  
SmGetEnrollId  
SmGetEnrollIds  
SmGetRc  
SmGetScripts  
SmGetTask  
SmGetTasks  
SmGetUserId  
SmGetUserIds  
SmGetDefaultTopics

## Task Related Functions and Callbacks

SmConnect  
SmOpen  
SmSetDefault  
SmQueryUserDefault  
SmSetUserDefault  
SmNqueryDefaultCallback  
SmNqueryUserDefaultCallback  
SmNsetDefaultCallback  
SmNsetUserDefaultCallback

## 3.42 SmQueryEnabledVocabs

### Purpose

SmQueryEnabledVocabs queries currently enabled vocabularies. This function obtains a list of all currently enabled vocabularies.

### Syntax

```
int SmQueryEnabledVocabs ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_NOT\_VALID\_REQUEST

### Reply Structure Functions by Message Type

SM\_QUERY\_ENABLED\_VOCABS\_REPLY  
SmGetRc  
SmGetVocabList

## Task Related Functions and Callbacks

SmAddToVocab  
SmDefineVocab  
SmDisableVocab  
SmEnableVocab  
SmQueryAddedWords  
SmQueryVocabs  
SmQueryWord  
SmRemoveFromVocab  
SmUndefineVocab  
SmNaddToVocabCallback  
SmNdefineVocabCallback  
SmNdisableVocabCallback  
SmNenableVocabCallback  
SmNqueryAddedWordsCallback  
SmNqueryEnabledVocabsCallback  
SmNqueryVocabsCallback  
SmNqueryWordsCallback  
SmNremoveFromVocabCallback  
SmNundefineVocabCallback

### 3.43 SmQueryEnrollIds

#### Purpose

SmQueryEnrollIds queries user enrollment IDs. This function returns a list of the enrollment IDs previously generated for a user by an enrollment procedure. Each time a user enrolls for the recognition system a new enrollment ID is generated for that user. When speech recognition is performed, one of the enrollment IDs from the user's enrollment ID list must be specified for the SmConnect function.

SmQueryEnrollIds returns the sample rate associated with each queried enrollid. The new access function SmGetSampleRates is provided to access the array of sample rates.

#### Syntax

```
int SmQueryEnrollIds ( char    *user_id,
                      char    *enroll_id,
                      char    *language,
                      SM_MSG *reply);
```

#### Parameters

*user\_id*      input - The name of the user whose enrollment ID list is to be returned.

*enroll\_id*    input - The enrollment ID queried, or NULL if all enrollments are queried.

*language*    input - The language for the enrollment ID, or NULL if all languages are queried.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
```



SM\_RC\_BAD\_ENROLLID  
SM\_RC\_BAD\_USERID  
SM\_RC\_SERVER\_MALLOC\_ERROR

## Reply Structure Functions by Message Type

SM\_QUERY\_ENROLLIDS\_REPLY  
SmGetAlphabets  
SmGetDescriptions  
SmGetEnrollIds  
SmGetLanguages  
SmGetPercentages  
SmGetRc  
SmGetSampleRates  
SmGetScripts  
SmGetStates

## Task Related Functions and Callbacks

SmNrequestNewEnrollIdCallback

## 3.44 SmQueryLanguages

### Purpose

SmQueryLanguages queries the available languages. This function returns a list of available languages that can be used to initialize the speech recognition engine.

### Syntax

```
int SmQueryLanguages ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_SERVER\_MALLOC\_ERROR

### Reply Structure Functions by Message Type

SM\_QUERY\_LANGUAGES\_REPLY  
SmGetDescriptions  
SmGetLanguages  
SmGetRc

### Task Related Functions and Callbacks

SmNqueryLanguagesCallback

### 3.45 SmQueryPhraseAlternatives

#### Purpose

SmQueryPhraseAlternatives returns the next recognition phrase alternative for a grammar vocabulary. This function is called repeatedly to get recognition phrase alternatives. When the flag SM\_PHRASE\_NO\_ALTERNATIVES\_AVAILABLE is set in the flags field of the reply, it indicates no more alternatives are available. The flags field can be retrieved by calling the access function SmGetFlags against the SM\_QUERY\_PHRASE\_ALTERNATIVES\_REPLY message. The maximum number of alternative calculated by the engine can be set/queried using SmSet/SmQuery.

#### Syntax

```
int SmQueryPharaseAlternatives ( unsigned long  flags,
                                SM_MSG         *reply );
```

#### Parameters

*flags*        Reserved. Should be set to 0.

*reply*        The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
```

## Reply Structure Functions by Message Type

SM\_QUERY\_PHRASE\_ALTERNATIVES\_REPLY

SmGetVocabName

SmGetFirmWords

SmGetAnnotations

SmGetWordTimes

SmGetFlags

## Task Related Functions and Callbacks

SmNqueryPhraseAlternativesReply

SmSet

SmQuery

## 3.46 SmQueryPronunciation

### Purpose

SmQueryPronunciation queries the existence of a pronunciation. This function provides a method for a requesting application to determine whether an associated pronunciation exists for a given spelling. This query is applied to the pronunciations that have been added to the user's personal pronunciation pool and the temporary pool. The reply indicates whether or not the pronunciation exists, but not where it was found. If a specified word does not exist in the current pool, the backup dictionary will be searched. If the word is found in the backup dictionary, it will be added to the temporary pool, except that when SmAddToVocab specifies a dictation vocabulary such as "text," the word will be added to the personal pool.

### Syntax

```
int SmQueryPronunciation ( char    *spelling,
                          SM_MSG *reply );
```

### Parameters

*spelling*     input - A spelling for which a pronunciation is sought.

*reply*        input/output - A pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_INVOCAB
SM_RC_NOT_VALID_REQUEST
```

## Reply Structure Functions by Message Type

SM\_QUERY\_PRONUNCIATION\_REPLY

SmGetRc

SmGetSpelling

SmGetSpellings

## Task Related Functions and Callbacks

SmAddPronunciation

SmQueryPronunciations

SmRemovePronunciation

SmNaddPronunciationCallback

SmNqueryPronunciationCallback

SmNqueryPronunciationsCallback

SmNremovePronunciationCallback

### 3.47 SmQueryPronunciationEx

#### Purpose

SmQueryPronunciationEx queries the existence of a pronunciation in a specified vocabulary. This function is an extension of the function SmQueryPronunciation. This function provides a method for a requesting application to determine whether an associated pronunciation exists for a given spelling in a given vocabulary. If a vocabulary is not specified, this function defaults to the behavior of SmQueryPronunciation. If a vocabulary is specified, this function returns pronunciations for the given spelling in that vocabulary. If the vocabulary specified is "text", the search for pronunciations will include personal words added by the application and any enabled topics.

#### Syntax

```
int SmQueryPronunciationEx ( char          *spelling,
                             char          *vocab,
                             unsigned long flags,
                             SM_MSG       *reply );
```

#### Parameters

*spelling*     input - A spelling for which a pronunciation is sought.

*vocab*        input - A vocab in which a pronunciation is sought. This parameter is optional. If null all vocabs in the current pool will be searched.

*flags*        input - Reserved. must be set to 0.

*reply*        input/output - A pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
```

SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_INVOCAB  
SM\_RC\_NOT\_VALID\_REQUEST

## Reply Structure Functions by Message Type

SM\_QUERY\_PRONUNCIATION\_REPLY  
SmGetPhoneticPronunciations  
SmGetPronunciations  
SmGetRc  
SmGetSpelling  
SmGetSpellings  
SmGetVocabName

## Task Related Functions and Callbacks

SmAddPronunciation  
SmQueryPronunciation  
SmQueryPronunciations  
SmRemovePronunciation  
SmNaddPronunciationCallback  
SmNqueryPronunciationCallback  
SmNqueryPronunciationsCallback  
SmNremovePronunciationCallback



## 3.48 SmQueryPronunciations

### Purpose

SmQueryPronunciations queries a listing of added pronunciations. This function provides a method for an application to request a list of word pronunciations that have been added to the user's pronunciation pool within the buffer limits.

### Syntax

```
int SmQueryPronunciations ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_ERROR  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR

### Reply Structure Functions by Message Type

SM\_QUERY\_PRONUNCIATIONS\_REPLY  
SmGetPronunciations  
SmGetRc  
SmGetSpellings

## Task Related Functions and Callbacks

SmAddPronunciation

SmQueryPronunciation

SmRemovePronunciation

SmNaddPronunciationCallback

SmNqueryPronunciationCallback

SmNqueryPronunciationsCallback

SmNremovePronunciationCallback

### 3.49 SmQueryPronunciationsEx

#### Purpose

SmQueryPronunciationsEx is an extension of SmQueryPronunciations. This function accepts an argument which specifies a starting point. It will return words beginning from this starting point up to the limit SM\_MAX\_WORDS (500). If the specified starting point is greater than the number of added words, no words will be returned.

Calling SmQueryPronuciations is equivalent to calling SmQueryPronunciationsEx with a start argument of 0. If there are more added pronunciations than the limit SM\_MAX\_WORDS, this function may be called repeatedly, specifying a start argument of 0 the first time and incrementing the start argument by the number of pronunciations returned. When 0 pronunciations are returned, all pronunciations have been retrieved.

#### Syntax

```
int SmQueryPronunciationsEx ( unsigned long  flags,
                             unsigned long  start,
                             SM_MSG        *reply );
```

#### Parameters

*flags*        Reserved. Should be set to 0.

*start*       Specifies 0-origin starting place.

*reply*       input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
 SM\_RC\_EALLOCS  
 SM\_RC\_EBADHANDLE  
 SM\_RC\_ENOCONN  
 SM\_RC\_ENOMEM  
 SM\_RC\_ENOMSG  
 SM\_RC\_ENOSERVER  
 SM\_RC\_EUNEXP  
 SM\_RC\_SM\_NOT\_OPEN  
 SM\_RC\_OK  
 SM\_RC\_INVALID\_PARM\_MAX\_LEN  
 SM\_RC\_NOT\_VALID\_REQUEST  
 SM\_RC\_SERVER\_MALLOC\_ERROR

## Reply Structure Functions by Message Type

SM\_QUERY\_PRONUNCIATIONS\_REPLY

SmGetSpellings

SmGetPronunciations

## Task Related Functions and Callbacks

SmAddPronunciation

SmQueryPronunciation

SmRemovePronunciation

SmNaddPronunciationCallback

SmNqueryPronunciationCallback

SmNqueryPronunciationsCallback

SmNqueryPronunciationsExCallback

SmNremovePronunciationCallback

## 3.50 SmQueryScripts

### Purpose

SmQueryScripts returns information about available scripts. The scripts about which information is returned can be filtered by userid/enrollid, or task.

### Syntax

```
int SmQueryScripts ( char    *user_id,
                    char    *enrollid,
                    char    *task,
                    SM_MSG *reply );
```

### Parameters

- user\_id*     input - Required if enrollid specified. Specifies user identifier.
- enrollid*    input - Required if user\_id specified. Specifies enrollment identifier.
- task*        input - Optional: if not NULL specifies task for which data on associated scripts will be returned.
- reply*       input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
SM_RC_BAD_LANGUAGE
```

## Reply Structure Functions by Message Type

SM\_QUERY\_SCRIPTS\_REPLY

SmGetDescriptions

SmGetIncrements

SmGetScriptFlags

SmGetScripts

SmGetSizes

SmGetStates

SmGetTasks

SmGetTrained

## Task Related Functions and Callbacks

SmSelectScript

SmRequestScriptText

SmNqueryScriptsCallback

## 3.51 SmQuerySessions

### Purpose

SmQuerySessions queries sessions and returns a list of connected recognition sessions.

### Syntax

```
int SmQuerySessions ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK

### Reply Structure Functions by Message Type

SM\_QUERY\_SESSIONS\_REPLY  
SmGetApplications  
SmGetEnrollIds  
SmGetRc  
SmGetTasks  
SmGetUserIds

### Task Related Functions and Callbacks

SmDetachSessions  
SmNdetachSessionsCallback  
SmNquerySessionsCallback

## 3.52 SmQuerySpeechData

### Purpose

SmQuerySpeechData queries the estimated size of speech data for the current session.

### Syntax

```
int SmQuerySpeechData ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK

### Reply Structure Functions by Message Type

SM\_QUERY\_SPEECH\_DATA\_REPLY  
SmGetSpeechDataSize

### Task Related Functions and Callbacks

SmQuerySpeechDataEx  
SmSaveSpeechData  
SmSaveSpeechDataEx  
SmRestoreSpeechData  
SmRestoreSpeechDataEx  
SmDiscardSpeechData  
SmNquerySpeechDataCallback



### 3.53 SmQuerySpeechDataEx

#### Purpose

SmQuerySpeechDataEx allows an application to query the estimated size of session data associated with specific tags.

#### Syntax

```
int SmQuerySpeechDataEx ( unsigned long  flags,
                          long          ntags,
                          unsigned long *tags,
                          SM_MSG       *reply );
```

#### Parameters

*flags*            input - Identifies options flags. Valid values:  
                   *SM\_SAVE\_ALL\_TAGS*  
                   Queries size for all tags in session.

*ntags*           input - The number of tags.

*tags*            input - The array of tags.

*reply*           input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
 SM\_RC\_EALLOCS  
 SM\_RC\_EBADHANDLE  
 SM\_RC\_ENOCONN  
 SM\_RC\_ENOMEM  
 SM\_RC\_ENOMSG  
 SM\_RC\_ENOSERVER  
 SM\_RC\_EUNEXP  
 SM\_RC\_SM\_NOT\_OPEN  
 SM\_RC\_OK  
 SM\_RC\_INVALID\_PARM\_MAX\_LEN  
 SM\_RC\_SERVER\_MALLOC\_ERROR  
 SM\_RC\_NOT\_VALID\_REQUEST

## Reply Structure Functions by Message Type

SM\_QUERY\_SPEECH\_DATA\_REPLY

SmGetSpeechDataSize

SmGetTags

SmGetFlags

## Task Related Functions and Callbacks

SmSaveSpeechData

SmRestoreSpeechData

SmQuerySpeechData

SmSaveSpeechDataEx

SmRestoreSpeechDataEx

SmNquerySpeechDataCallback

### 3.54 SmQuerySpeechUserSize

#### Purpose

SmQuerySpeechUserSize allows an application to query the size of speaker data associated with a userid.

#### Syntax

```
int SmQuerySpeechUserSize ( char      *userid,
                           char      *enrollid,
                           char      *language,
                           char      *script,
                           unsigned long flags,
                           SM_MSG    *reply );
```

#### Parameters

*userid*        input - UserID for which speaker data is to be archived.

*enrollid*     input - Optional EnrollID for which speaker data is to be archived.

*language*    input - Optional language for which speaker data is to be archived.

*script*       input - Optional script for which speaker data is to be archived.

*flags*        input - Reserved options flags.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
```

## Reply Structure Functions by Message Type

SM\_QUERY\_SPEECH\_USER\_SIZE\_REPLY

SmGetUserid

SmGetEnrollid

SmGetLanguages

SmGetScript

SmGetFlags

SmGetSpeechDataSize

## Task Related Functions and Callbacks

SmRestoreSpeechUser

SmSaveSpeechUser

SmNquerySpeechUserSizeCallback

### 3.55 SmQueryTasks

#### Purpose

SmQueryTasks queries a speech recognition engine domain list. This function returns a list of the domains that can be used for dictation and enrollment. A domain consists of a set of vocabularies, word-usage models, and other associated parameters used during a recognition session. A domain name must be specified when the speech recognition engine is initialized for a user. This function will also return the maximum sample rate supported by a domain.

#### Syntax

```
int SmQueryTasks ( char    *language,
                  SM_MSG *reply );
```

#### Parameters

*language*    input - The language used for the domain. If specified as NULL, all domains are returned, regardless of language.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_SERVER_MALLOC_ERROR
```

## Reply Structure Functions by Message Type

SM\_QUERY\_TASKS\_REPLY

SmGetAlphabets

SmGetDescriptions

SmGetLanguages

SmGetRc

SmGetTasks

## Task Related Functions and Callbacks

SmNqueryTasksCallback

### 3.56 SmQueryTopics

#### Purpose

SmQueryTopics will be called by a client application to request information about the current topics or a specified topic. It accepts a taskid argument and if specified will return information about only those topics associated with that task.

The topics will be returned in sorted order such that the topics which are designed to be used the this task (for example, the preferred topics) will appear first. The npreferred field refers to these topics. The entire array of topics is accessed using SmGetTopics. The npreferred field is accessed using SmGetPreferred.

#### Syntax

```
int SmQueryTopics ( unsigned long  flags,
                    char           *taskid,
                    char           *topic,
                    SM_MSG         *reply);
```

#### Parameters

- flags*        input - Identifies options flags. Valid values:
- SM\_INSTALLED\_TOPICS*  
Query all installed topics on the system
  - SM\_ACTIVE\_TOPICS*  
Query only topics currently in use
- taskid*      input - Name of task for which associated topics will be returned.
- topic*        input Optional topic name, if specified only information about this topic will be returned.
- reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
```

SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_SERVER\_MALLOC\_ERROR  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR

## Reply Structure Functions by Message Type

SM\_QUERY\_TOPICS\_REPLY  
SmGetDescriptions  
SmGetSpellings  
SmGetFlags  
SmGetPreferred  
SmGetTopics

## Task Related Functions and Callbacks

SmQueryUserDefault  
SmNqueryTopicsCallback  
SmNqueryUserDefaultCallback  
SmNsetUserDefaultCallback



### 3.57 SmQueryUserDefault

#### Purpose

The default enrollid, taskid, and topics are stored by the engine on a per-user basis. The function SmQueryUserDefault is used to query one of these defaults for a particular user.

If the item SM.DEFAULT.TOPICS is specified, the item\_value returned in the reply structure will be a string of the default topics for the specified userid concatenated with blank delimiters (for example, "topic1 topic2 topic3").

#### Syntax

```
int SmQueryUserDefault ( char    *user_id,
                        long     item,
                        SM_MSG *reply);
```

#### Parameters

*user\_id*      input - Specifies which default to query. Valid values:

*SM\_DEFAULT\_ENROLLID*

Specifies request for default enrollid.

*SM\_DEFAULT\_TASK*

Specifies a request for default task.

*SM\_DEFAULT\_TOPICS*

Specifies request for default topics.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM

SM\_RC\_EALLOCS

SM\_RC\_EBADHANDLE

SM\_RC\_ENOCONN

SM\_RC\_ENOMEM

SM\_RC\_ENOMSG

SM\_RC\_ENOSERVER

SM\_RC\_EUNEXP

SM\_RC\_SM\_NOT\_OPEN

SM\_RC\_OK

SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_VALID\_REQUEST

## **Reply Structure Functions by Message Type**

SM\_QUERY\_USER\_DEFAULT\_REPLY  
SmGetTask  
SmGetTasks  
SmGetUserId  
SmGetUserIds  
SmGetEnrollId  
SmGetEnrollIds  
SmGetDefaultTopics

## **Task Related Functions and Callbacks**

SmConnect  
SmOpen  
SmSetUserDefault  
SmNconnectCallback  
SmNqueryUserDefaultCallback  
SmNconnectCallback  
SmNsetUserDefaultCallback

### 3.58 SmQueryUserInfo

#### Purpose

SmQueryUserInfo queries user ID or enrollment ID information. This function retrieves information associated with a user ID or an enrollment ID. Only one value can be queried at a time.

#### Syntax

```
int SmQueryUserInfo ( char    *user_id,
                      char    *enroll_id,
                      char    *itemname,
                      SM_MSG *reply );
```

#### Parameters

*user\_id*      input - The name of the user whose information is retrieved.

*enroll\_id*    input - The enrollment ID of the user whose information is retrieved. This parameter is NULL if information on the user ID is retrieved.

*itemname*    input - The queried item name. The special predefined itemname parameter, SM\_USER\_DIRECTORY, returns the full path of the location where user files are stored. An application can store its files in the same location.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_BAD_ENROLLID
SM_RC_BAD_ITEM
SM_RC_BAD_USERID
```

## Reply Structure Functions by Message Type

SM\_QUERY\_USER\_INFO\_REPLY

SmGetEnrollId

SmGetEnrollIds

SmGetNameValue

SmGetRc

SmGetUserId

SmGetUserIds

## Task Related Functions and Callbacks

SmQueryUsers

SmSetUserInfo

SmNqueryUsersCallback

SmNqueryUserInfoCallback

SmNsetUserInfoCallback

## 3.59 SmQueryUsers

### Purpose

SmQueryUsers queries a user list. This function returns a list of users allowed to use the speech recognition engine. To establish a speech session, a user from this list must be supplied in the SmNuserId speech attribute in the argument list passed to SmConnect.

### Syntax

```
int SmQueryUsers ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_SERVER\_MALLOC\_ERROR

### Reply Structure Functions by Message Type

SM\_QUERY\_USERS\_REPLY  
SmGetAlphabets  
SmGetDescriptions  
SmGetRc  
SmGetUserIds  
SmGetUsers

## Task Related Functions and Callbacks

SmQueryUserInfo

SmSetUserInfo

SmNqueryUsersCallback

SmNqueryUserInfoCallback

SmNsetUserInfoCallback

## 3.60 SmQueryUtterances

### Purpose

SmQueryUtterances returns information about the sentences which have been recorded with the current script. This function can only be called when connected to the speech recognition engine for an enrollment session.

### Syntax

```
int SmQueryUtterances ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_BAD\_SCRIPT  
SM\_RC\_SERVER\_MALLOC\_ERROR

### Reply Structure Functions by Message Type

SM\_QUERY\_UTTERANCES\_REPLY  
SmGetNumberProcessed  
SmGetNumberRequired  
SmGetNumberUtterances  
SmGetUtteranceList

## **Task Related Functions and Callbacks**

SmDiscardUtterance

SmSetUtteranceNumber

SmNqueryUtterancesCallback



## 3.61 SmQueryVocabs

### Purpose

SmQueryVocabs queries currently defined vocabularies. This function obtains the list of all defined speech recognition engine vocabularies that belong to the current session. This includes predefined vocabularies and those defined after initialization with the SmDefineVocab or SmDefineVocabEx function.

### Syntax

```
int SmQueryVocabs ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_NOT\_VALID\_REQUEST

### Reply Structure Functions by Message Type

SM\_QUERY\_VOCABS\_REPLY  
SmGetRc  
SmGetVocabList

## Task Related Functions and Callbacks

SmAddToVocab  
SmDefineVocab  
SmDisableVocab  
SmEnableVocab  
SmQueryAddedWords  
SmQueryEnabledVocabs  
SmQueryWord  
SmRemoveFromVocab  
SmUndefineVocab  
SmNaddToVocabCallback  
SmNdefineVocabCallback  
SmNdisableVocabCallback  
SmNenableVocabCallback  
SmNqueryAddedWordsCallback  
SmNqueryEnabledVocabsCallback  
SmNqueryVocabsCallback  
SmNqueryWordsCallback  
SmNremoveFromVocabCallback  
SmNundefineVocabCallback

## 3.62 SmQueryWord

### Purpose

SmQueryWord checks for a specified word in all active vocabularies.

### Syntax

```
int SmQueryWord ( SM_WORD *word,
                  SM_MSG  *reply );
```

### Parameters

*word*            input - The queried word.

*reply*           input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_INVOCAB

### Reply Structure Functions by Message Type

SM\_QUERY\_WORD\_REPLY  
SmGetRc  
SmGetVocabList  
SmGetWords

## Task Related Functions and Callbacks

SmAddToVocab  
SmDefineVocab  
SmDisableVocab  
SmEnableVocab  
SmPlayWords  
SmQueryAddedWords  
SmQueryEnabledVocabs  
SmQueryVocabs  
SmRemoveFromVocab  
SmUndefineVocab  
SmWordCorrection  
SmNaddToVocabCallback  
SmNdefineVocabCallback  
SmNdisableVocabCallback  
SmNenableVocabCallback  
SmNplayWordsCallback  
SmNplayWordsStatusCallback  
SmNqueryAddedWordsCallback  
SmNqueryEnabledVocabsCallback  
SmNqueryVocabsCallback  
SmNqueryWordsCallback  
SmNremoveFromVocabCallback  
SmNundefineVocabCallback  
SmNwordCorrectionCallback

### 3.63 SmReceiveMsg

#### Purpose

SmReceiveMsg receives a message from the speech recognition engine. This function provides the method through which speech-aware applications receive asynchronous messages, including unsolicited asynchronous messages such as SM\_RECOGNIZED.TEXT, from the speech recognition engine. This function receives one complete message from the speech recognition engine. For details, see "Session Sharing" in the SMAPI Developer's Guide.

#### Syntax

```
int SmReceiveMsg ( unsigned long  ap_val,
                   SM_MSG        *msg);
```

#### Parameters

*ap\_val*        input - For Windows applications, pass lParam here. Non-Windows applications should set this to 0.

*msg*            output - The pointer to the reply structure.

#### Return Values

SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL  
 SM\_RC\_SM\_NOT\_OPEN

#### Reply Structure Functions by Message Type

SmGetRc  
 SmGetRcName

### 3.64 SmRecognizeNextWord

#### Purpose

SmRecognizeNextWord enables the recognition of the next word. This function searches for the next word to decode. When running, the engine searches the currently enabled vocabularies to find a word that matches the incoming speech. The vocabulary that contains the best match determines what happens next:

- If the matching word comes from a dictation vocabulary, the engine sends the firm words to the application in an SM\_RECOGNIZED\_TEXT message and continues decoding.
- If the matching word comes from a command vocabulary, the engine sends the word and some alternative choices to the application in an SM\_RECOGNIZED\_WORD message. The engine then halts and waits for the application to request another word.

If a recognized word occurs in two or more vocabularies enabled at the same time, the engine selects the word from the more recently enabled command vocabulary. Command vocabularies always override dictation vocabularies; when a recognized word occurs in both a command and dictation vocabulary enabled at the same time, the engine selects the command-vocabulary word.

#### Syntax

```
int SmRecognizeNextWord ( SM_MSG *reply );
```

#### Parameters

*reply*            input/output - The pointer to the reply structure or to the SmAsynchronous value indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_NOT_VALID_REQUEST
```

## Reply Structure Functions by Message Type

SM\_RECOGNIZED\_WORD

SmGetAlternates

SmGetFirmWords

SmGetRc

SmGetTimes

SM\_RECOGNIZE\_NEXT\_WORD\_REPLY

SmGetRc For command vocabularies with microphone off and audio input stream processed:

SM\_UTTERANCE\_COMPLETED

SmGetRc

SmGetUtteranceNumber

## Task Related Functions and Callbacks

SmHaltRecognizer

SmMicOff

SmMicOn

SmQuery

SmSet

SmNhaltRecognizerCallback

SmNmicOffCallback

SmNmicOnCallback

SmNqueryCallback

SmNrecognizeNextWordCallback

SmNrecognizedTextCallback

SmNrecognizedWordCallback

SmNsetCallback

SmNutteranceCompletedCallback

## 3.65 SmReleaseFocus

### Purpose

SmReleaseFocus releases speech focus. This function releases speech focus for the application calling this API. If the application does not have the speech focus, this function returns SM\_RC\_NOT\_VALID\_REQUEST; otherwise it returns SM\_RC\_OK.

If the call is made asynchronously the reply can be dispatched through either of the following:

- SM\_RELEASE\_FOCUS
- SmNreleaseFocusCallback

### Syntax

```
int SmReleaseFocus ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_OK

### Reply Structure Functions by Message Type

SM\_RELEASE\_FOCUS  
SmGetRc  
SM\_FOCUS\_LOST

### Task Related Functions and Callbacks

SmRequestFocus  
SmNfocusGrantedCallback  
SmNfocusRequestedCallback  
SmNreleaseFocusCallback  
SmNrequestFocusCallback



## 3.66 SmRemoveCallback

### Purpose

SmRemoveCallback removes a single callback routine for a specific callback.

### Syntax

```
int SmRemoveCallback ( char      *reply_name,  
                       SmHandler *handler,  
                       void      *client_data);
```

### Parameters

*reply\_name*

input - The name of the type of message.

*handler*

input - The function name of the routine that handles the message.

*client\_data*

input - The data passed back to the handler when it is called.

### Return Values

SM\_RC\_ENOMEM  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_OK  
SM\_RC\_SM\_NOT\_OPEN

### Task Related Functions and Callbacks

SmAddCallback  
SmDispatch

## 3.67 SmRemoveEnrollid

### Purpose

SmRemoveEnrollid removes an enrollid from a userid.

### Syntax

```
int SmRemoveEnrollid ( char    *user_id,
                      char    *enroll_id,
                      SM_MSG *reply );
```

### Parameters

*user\_id*      input - The userid from which the enrollid is removed.

*enroll\_id*    input - The enrollid to remove.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_USERID  
SM\_RC\_BAD\_ENROLLID  
SM\_RC\_SERVER\_MALLOC\_ERROR  
SM\_RC\_SERVER\_FILE\_CLOSE\_ERROR

### Reply Structure Functions by Message Type

SM\_REMOVE\_ENROLLID\_REPLY

## **Task Related Functions and Callbacks**

SmAddUser

SmAddEnrollid

SmRemoveUser

SmNremoveEnrollidCallback

### 3.68 SmRemoveFromVocab

#### Purpose

SmRemoveFromVocab removes words from a dynamic vocabulary. This function removes words previously added, using SmAddToVocab, to either dynamic vocabularies created by SmDefineVocab, SmDefinVocabEx or added to predefined vocabularies. This function can be used to dynamically change command vocabularies in an application. Pronunciations for the specified words are not removed by this call and can be reused later by words added to this or any other vocabulary. This call is valid only when the speech recognition engine is not accepting dictation.

#### Syntax

```
int SmRemoveFromVocab ( char      *vocab,
                        short      nvocwords,
                        SM_VOCWORD *vocwords[],
                        SM_MSG     *reply );
```

#### Parameters

*vocab*        input - The name of the vocabulary from which words are removed.

*nvocwords*   input - The number of words removed from the vocabulary. Limited to SM\_MAX\_VOCWORDS.

*vocwords*    input - The spellings of the words removed from the vocabulary.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
```

SM\_RC\_BAD\_VOCAB  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR  
SM\_RC\_SERVER\_FILE\_WRITE\_ERROR

## Reply Structure Functions by Message Type

SM\_REMOVE\_FROM\_VOCAB\_REPLY  
SmGetRc  
SmGetVocabName

## Task Related Functions and Callbacks

SmAddToVocab  
SmDefineVocab  
SmDefineVocabEx  
SmDisableVocab  
SmEnableVocab  
SmQueryAddedWords  
SmQueryEnabledVocabs  
SmQueryVocabs  
SmQueryWord  
SmUndefineVocab  
SmNaddToVocabCallback  
SmNdefineVocabCallback  
SmNdisableVocabCallback  
SmNenableVocabCallback  
SmNqueryAddedWordsCallback  
SmNqueryEnabledVocabsCallback  
SmNqueryVocabsCallback  
SmNqueryWordsCallback  
SmNremoveFromVocabCallback  
SmNundefineVocabCallback

## 3.69 SmRemovePronunciation

### Purpose

SmRemovePronunciation removes a pronunciation from the user's personal pronunciation pool. You cannot remove pronunciations from the permanent pool.

### Syntax

```
int SmRemovePronunciation ( char    *spelling,
                           char    *pronunciation,
                           SM_MSG *reply );
```

### Parameters

*spelling*      input - The word spelling which references the pronunciation.

*pronunciation*  
                input - The sounds-like spelling of the pronunciation to be removed.

*reply*          input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
SM_RC_NOT_YET
SM_RC_SERVER_ERROR
```

## Reply Structure Functions by Message Type

SM\_REMOVE\_PRONUNCIATION\_REPLY

SmGetRc

SmGetSpelling

SmGetSpellings

## Task Related Functions and Callbacks

SmAddPronunciation

SmCorrectText

SmCorrectTextCancel

SmNewContext

SmQueryPronunciation

SmQueryPronunciations

SmWordCorrection

SmNaddPronunciationCallback

SmNcorrectTextCallback

SmNcorrectTextCancelCallback

SmNnewContextCallback

SmNqueryPronunciationCallback

SmNqueryPronunciationsCallback

SmNremovePronunciationCallback

SmNwordCorrectionCallback

## 3.70 SmRemoveUser

### Purpose

SmRemoveUser removes a speech user from the system.

### Syntax

```
int SmRemoveUser ( char    *user_id,
                   SM_MSG *reply );
```

### Parameters

*user\_id* input - Identifies the speech user to be removed.

*reply* input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_USERID  
SM\_RC\_SERVER\_MALLOC\_ERROR  
SM\_RC\_SERVER\_FILE\_CLOSE\_ERROR

### Reply Structure Functions by Message Type

SM\_REMOVE\_USER\_REPLY



## **Task Related Functions and Callbacks**

SmAddUser

SmAddEnrollid

SmRemoveEnrollid

SmNremoveUserCallback

## 3.71 SmRequestFocus

### Purpose

SmRequestFocus requests speech focus. An application requests speech focus with a SmRequestFocus function call. If the call is made asynchronously, the reply can be dispatched through either of the following:

- SM\_REQUEST\_FOCUS
- SmNrequestFocusCallback

If the request is accepted, an asynchronous message is sent after the engine grants focus. This event can be dispatched through either of the following:

- SM\_FOCUS\_GRANTED
- SmNfocusGrantedCallback

**Please note:** SM\_RC\_FOCUS\_GRANTED means the requesting application had previously been granted speech focus and no asynchronous notification message is generated by the engine.

### Syntax

```
int SmRequestFocus ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_FOCUS\_GRANTED  
SM\_RC\_OK

### Reply Structure Functions by Message Type

SM\_REQUEST\_FOCUS  
SmGetRc  
SM\_FOCUS\_GRANTED

## Task Related Functions and Callbacks

SmReleaseFocus

SmNfocusGrantedCallback

SmNfocusRequestedCallback

SmNreleaseFocusCallback

SmNrequestFocusCallback

## 3.72 SmRequestMicOff

### Purpose

SmRequestMicOff requests that the microphone be turned off. This function requests the speech recognition engine to request the application with speech focus to turn off the microphone. If no application has speech focus, then SM\_RC\_NO\_FOCUS\_APP is returned.

### Syntax

```
int SmRequestMicOff ( SM_MSG *reply );
```

### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_NO\_FOCUS\_APP  
SM\_RC\_OK

### Reply Structure Functions by Message Type

SM\_REQUEST\_MIC\_OFF\_REPLY  
SmGetRc  
SM\_REQUEST\_MIC\_OFF

### Task Related Functions and Callbacks

SmRequestMicOn  
SmNrequestMicOffCallback  
SmNrequestMicOnCallback

### 3.73 SmRequestMicOn

#### Purpose

SmRequestMicOn requests that the microphone be turned on. This function requests the speech recognition engine to request the application with speech focus to turn on the microphone. If no application has speech focus, then SM\_RC\_NO\_FOCUS\_APP is returned.

#### Syntax

```
int SmRequestMicOn ( SM_MSG *reply );
```

#### Parameters

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

SM\_RC\_NO\_FOCUS\_APP  
SM\_RC\_OK

#### Reply Structure Functions by Message Type

SM\_REQUEST\_MIC\_ON\_REPLY  
SmGetRc  
SM\_REQUEST\_MIC\_ON

#### Task Related Functions and Callbacks

SmRequestMicOff  
SmNrequestMicOffCallback  
SmNrequestMicOnCallback

### 3.74 SmRequestNewEnrollid

#### Purpose

SmRequestNewEnrollid returns a valid enrollid name which can be used for adding a new enrollid to a userid.

#### Syntax

```
int SmRequestNewEnrollid ( char    *user_id,
                           char    *user_name,
                           SM_MSG *reply );
```

#### Parameters

- user\_id*      input - Short name to identify the user. Must be no more than SM\_MAX\_USERID\_LEN in length. The following characters must not appear in a userid: :\\n\r\t"!#\$%&()\*\*,.;<=>?|
- user\_name*    input - Full name of the user.
- reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
```

## Reply Structure Functions by Message Type

SM\_REQUEST\_NEW\_ENROLLID\_REPLY

SmGetUserid

SmGetUserids

SmGetEnrollid

SmGetEnrollids

## Task Related Functions and Callbacks

SmAddUser

SmAddEnrollid

SmRemoveEnrollid

SmRemoveUser

SmNrequestNewEnrollidCallback

## 3.75 SmRequestScriptText

### Purpose

SmRequestScriptText returns script text for a particular utterance. This function can only be called when connected to the speech recognition engine for an enrollment session.

### Syntax

```
int SmRequestScriptText ( long    uttno,
                        SM_MSG *reply );
```

### Parameters

*uttno*        input - The utterance number for which script text is requested.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
 SM\_RC\_EALLOCS  
 SM\_RC\_EBADHANDLE  
 SM\_RC\_ENOCONN  
 SM\_RC\_ENOMEM  
 SM\_RC\_ENOMSG  
 SM\_RC\_ENOSERVER  
 SM\_RC\_EUNEXP  
 SM\_RC\_SM\_NOT\_OPEN  
 SM\_RC\_OK  
 SM\_RC\_NOT\_VALID\_REQUEST  
 SM\_RC\_BAD\_UTTNO  
 SM\_RC\_BAD\_SCRIPT  
 SM\_RC\_SERVER\_MALLOC\_ERROR

### Reply Structure Functions by Message Type

SM\_REQUEST\_SCRIPT\_TEXT\_REPLY  
 SmGetWords  
 SmGetComment



## **Task Related Functions and Callbacks**

SmQueryScripts

SmSelectScript

SmNrequestScriptTextCallback

## 3.76 SmRestoreSpeechData

### Purpose

SmRestoreSpeechData overwrites the current session files and resets the running utterance number.

**Please note:** For this call to be valid, the application must first enable audio saving through SmSet(SM\_SAVE\_AUDIO, TRUE). For more information on saving and restoring speech sessions, reference "Writing ViaVoice Applications to Save and Restore Audio" in the SMAPI Developer's Guide.

Since this function can take a long time, we suggest that applications call SmRestoreSpeechData asynchronously to avoid the SMAPI synchronous call timeout. If this function is called asynchronously all subsequent SMAPI function requests from the application will be returned with an rc of SM\_RC\_NOT\_VALID\_REQUEST until the SmRestoreSpeechData request completes.

### Syntax

```
int SmRestoreSpeechData ( char    *archive,
                          long     version,
                          SM_MSG *reply );
```

### Parameters

*archive*      input - File name specified on SmSaveSpeechData.

*version*      input - Value passed as argument to SmSaveSpeechData by the application when saving session audio data and checked by the engine when the session audio data is restored. reply

*input/output* - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
```

SM\_RC.SM\_NOT\_OPEN  
SM\_RC.OK  
SM\_RC.INVALID\_PARM\_MAX\_LEN  
SM\_RC.SERVER\_FILE\_OPEN\_ERROR  
SM\_RC.SERVER\_FILE\_READ\_ERROR  
SM\_RC.SERVER\_FILE\_WRITE\_ERROR

## **Reply Structure Functions by Message Type**

SM\_RESTORE\_SPEECH\_DATA\_REPLY  
SmGetSpeechDataArchive  
SmGetSpeechDataVersion

## **Task Related Functions and Callbacks**

SmDiscardSpeechData  
SmQuerySpeechData  
SmQuerySpeechDataEx  
SmRestoreSpeechDataEx  
SmSaveSpeechData  
SmSaveSpeechDataEx

### 3.77 SmRestoreSpeechDataEx

#### Purpose

SmRestoreSpeechDataEx restores audio data from an archive file into the current session and resets the running utterance number. This function is an extension of the function SmRestoreSpeechDataEx.

**Please note:** For this call to be valid, the application must first enable audio saving through SmSet(SM\_SAVE\_AUDIO, TRUE). For more information on saving and restoring speech sessions, reference "Writing ViaVoice Applications to Save and Restore Audio" in the SAPI Developer's Guide. Since this function can take a long time, we suggest that applications call SmRestoreSpeechDataEx asynchronously to avoid the SAPI synchronous call timeout. If this function is called asynchronously all subsequent SAPI function requests from the application will be returned with an rc of SM\_RC\_NOT\_VALID\_REQUEST until the SmRestoreSpeechDataEx request completes.

#### Syntax

```
int SmRestoreSpeechDataE ( char          *archive,
                           long          version,
                           unsigned long flags,
                           SM_MSG       *reply );
```

#### Parameters

- archive*      input - File name specified on SmSaveSpeechData.
- version*      input - Value passed as argument to SmSaveSpeechData by the application when saving session audio data and checked by the engine when the session audio data is restored.
- flags*        input - Options flags. Valid values:
- SM\_MERGE\_TAGS*  
Specifies that data restored from archive will be merged with current session data instead of replacing it. An offset is returned which, when added to the archived tags, will yield the new tag values. This offset is retrieved from the reply using the access function SmGetTagOffset
  - SM\_REPLACE\_TAGS*  
Specifies that data restored from the archive will replace current session data.
  - SM\_RESTORE\_VERSION\_NUMBER*  
Specifies to skip the version number check and restore data from the archive regardless of its version number. The version number

from the archive is returned and can be retrieved from the reply using the access function `SmGetSpeechDataVersion`.

*reply*      input/output - The pointer to a reply structure or to `SmAsynchronous` indicating that the call is made asynchronously.

## Return Values

`SM_RC_DEALLOCATING_SH_MEM`  
`SM_RC_EALLOCS`  
`SM_RC_EBADHANDLE`  
`SM_RC_ENOCONN`  
`SM_RC_ENOMEM`  
`SM_RC_ENOMSG`  
`SM_RC_ENOSERVER`  
`SM_RC_EUNEXP`  
`SM_RC_SM_NOT_OPEN`  
`SM_RC_OK`  
`SM_RC_INVALID_PARM_MAX_LEN`  
`SM_RC_SERVER_FILE_OPEN_ERROR`  
`SM_RC_SERVER_FILE_READ_ERROR`  
`SM_RC_SERVER_FILE_WRITE_ERROR`

## Reply Structure Functions by Message Type

`SM_RESTORE_SPEECH_DATA_REPLY`  
`SmGetFlags`  
`SmGetSpeechDataArchive`  
`SmGetSpeechDataVersion`  
`SmGetTagOffset`

## Task Related Functions and Callbacks

`SmDiscardSpeechData`  
`SmQuerySpeechData`  
`SmQuerySpeechDataEx`  
`SmRestoreSpeechData`  
`SmSaveSpeechData`  
`SmSaveSpeechDataEx`

## 3.78 SmRestoreSpeechUser

### Purpose

SmRestoreSpeechUser allows an application to restore speaker data associated with a userid.

### Syntax

```
int SmRestoreSpeechUser ( void      *archive,
                          char      *userid,
                          char      *enrollid,
                          char      *language,
                          char      *script,
                          unsigned long  flags,
                          SM_MSG      *reply );
```

### Parameters

<i>archive</i>	input - Archive file name where data is saved.
<i>userid</i>	input - UserID for which speaker data is to be restored.
<i>enrollid</i>	input - Optional EnrollID for which speaker data is to be restored.
<i>language</i>	input - Optional language for which speaker data is to be restored.
<i>script</i>	input - Optional script for which speaker data is to be restored.
<i>flags</i>	input - Reserved options flags.
<i>reply</i>	input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC.SM_NOT_OPEN
SM_RC_OK
SM_RC.INVALID_PARM_MAX_LEN
SM_RC.SERVER_PROCESS_ERROR
```

## Reply Structure Functions by Message Type

SM\_RESTORE\_SPEECH\_DATA\_REPLY S

SmGetSpeechDataArchive

SmGetUserid

SmGetEnrollid

SmGetLanguages

SmGetScript

SmGetFlags

## Task Related Functions and Callbacks

SmQuerySpeechUserSize

SmSaveSpeechUser

SmNsaveSpeechuserCallback

## 3.79 SmSaveSpeechData

### Purpose

SmSaveSpeechData saves current session speech data to a file.

**Please note:** For this call to be valid, the application must first enable audio saving through SmSet(SM\_SAVE\_AUDIO, TRUE). For more information on saving and restoring speech sessions, reference "Writing ViaVoice Applications to Save and Restore Audio" in the SMAPI Developer's Guide.

Since this function can take a long time, we suggest that applications call SmSaveSpeechData asynchronously to avoid the SMAPI synchronous call timeout. If this function is called asynchronously all subsequent SMAPI function requests from the application will be returned with an rc of SM\_RC\_NOT\_VALID\_REQUEST until the SmSaveSpeechData request completes.

### Syntax

```
int SmSaveSpeechData ( char    *archive,
                      long     version,
                      long     flags,
                      SM_MSG *reply );
```

### Parameters

*archive*      input - File name where data will be stored.

*version*      input - Identifies the archive file version.

*flags*          input - Identifies how the data should be saved, using one of the following values:

*SM\_NORMAL\_FILE*

The speech data is written to the file specified by the file name. This file will contain only engine data.

*SM\_COMPOUND\_FILE*

The speech data is written to the Windows compound file specified by the file name, using IStorage/IStream. The data is saved in a section named "VTDSessionArchive." This allows an application to save both engine state and application state data in the same file.

*reply*          input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.



## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR  
SM\_RC\_SERVER\_FILE\_READ\_ERROR  
SM\_RC\_SERVER\_FILE\_WRITE\_ERROR

## Reply Structure Functions by Message Type

SM\_SAVE\_SPEECH\_DATA\_REPLY  
SmGetSpeechDataArchive  
SmGetSpeechDataOptions  
SmGetSpeechDataSize  
SmGetSpeechDataVersion

## Task Related Functions and Callbacks

SmDiscardSpeechData  
SmQuerySpeechData  
SmQuerySpeechDataEx  
SmSaveSpeechDataEx  
SmRestoreSpeechData  
SmRestoreSpeechDataEx

### 3.80 SmSaveSpeechDataEx

#### Purpose

SmSaveSpeechDataEx saves current session speech data to a file. This function is an extension of the function SmSaveSpeechData. It allows an application to specify a list of tags for which the associated audio data will be saved. This function can also be used to save audio data for unsupervised training.

**Please note:** For this call to be valid, the application must first enable audio saving through SmSet(SM\_SAVE\_AUDIO, TRUE). For more information on saving and restoring speech sessions, reference "Writing ViaVoice Applications to Save and Restore Audio" in the SMAPI Developer's Guide.

Since this function can take a long time, we suggest that applications call SmSaveSpeechDataEx asynchronously to avoid the SMAPI synchronous call timeout. If this function is called asynchronously all subsequent SMAPI function requests from the application will be returned with an rc of SM\_RC\_NOT\_VALID\_REQUEST until the SmSaveSpeechDataEx request completes.

#### Syntax

```
int SmSaveSpeechDataEx ( char          *archive,
                        long           version,
                        unsigned long  flags,
                        long           ntags,
                        unsigned long  *tsg,
                        SM_MSG         *reply );
```

#### Parameters

*archive*      input - File name where data will be stored. May be null if only copying data for unsupervised adaptation.

*version*      input - Identifies the archive file version. If copying data for unsupervised adaptation this is an application defined document identifier. Data to be saved for unsupervised adaptation will be grouped by document identifier.

*flags*        input - Identifies how the data should be saved, using one of the following values:

*SM\_NORMAL\_FILE*

The speech data is written to the file specified by the file name. This file will contain only engine data.

*SM\_COMPOUND\_FILE*

Windows only - The speech data is written to the Windows compound file specified by the file name, using IStorage/IStream. The

data is saved in a section named "VTDSessionArchive." This allows an application to save both engine state and application state data in the same file.

### *SM\_SAVE\_ALL\_TAGS*

Specifies saving audio data associated with all tags in the session.

### *SM\_SAVE\_FOR\_ADAPTATION*

Specifies saving audio data for unsupervised adaptation. Data is copied to a location where it can be read by the training program. If the archive name is null data will not be written to the archive; only copied for unsupervised adaptation.

<i>ntags</i>	input - Specifies the number of tags in the tags array. Valid values: 1 - SM_MAX_API_TAGS.
<i>tags</i>	input - A list of tags for which associated audio data will be saved.
<i>reply</i>	input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_SERVER\_MALLOC\_ERROR  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR  
SM\_RC\_SERVER\_FILE\_READ\_ERROR  
SM\_RC\_SERVER\_FILE\_WRITE\_ERROR

## Reply Structure Functions by Message Type

SM\_SAVE\_SPEECH\_DATA\_REPLY  
SmGetFlags  
SmGetSpeechDataArchive

SmGetSpeechDataSize

SmGetSpeechDataVersion

SmGetTags

## **Task Related Functions and Callbacks**

SmDiscardSpeechData

SmQuerySpeechData

SmQuerySpeechDataEx

SmRestoreSpeechData

SmRestoreSpeechDataEx

SmSaveSpeechData

SmNsaveSpeechDataCallback

### 3.81 SmSaveSpeechUser

#### Purpose

SmSaveSpeechUser allows an application to save speaker data associated with a userid.

#### Syntax

```
int SmSaveSpeechUser ( void          *archive,
                      char          *userid,
                      char          *enrollid,
                      char          *language,
                      char          *script,
                      unsigned long  flags,
                      SM_MSG        *reply );
```

#### Parameters

*archive*     input - Archive file name where data is saved.

*userid*     input - UserID for which speaker data is to be archived.

*enrollid*   input - Optional EnrollID for which speaker data is to be archived.

*language*   input - Optional language for which speaker data is to be archived.

*script*     input - Optional script for which speaker data is to be archived.

*flags*       input - Reserved, must be set to 0.

*reply*       input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_SERVER_PROCESS_ERROR
```

## Reply Structure Functions by Message Type

SM\_SAVE\_SPEECH\_DATA\_REPLY

SmGetSpeechDataArchive

SmGetUserid

SmGetEnrollid

SmGetLanguages

SmGetScript

SmGetFlags

SmGetSpeechDataSize

## Task Related Functions and Callbacks

SmQuerySpeechUserSize

SmRestoreSpeechUser

SmNsaveSpeechUserCallback

## 3.82 SmSelectScript

### Purpose

SmSelectScript selects a particular script for enrollment. This function can be used to change the script which was specified when connecting for enrollment. This function can only be called when connected to the speech recognition engine for an enrollment session.

### Syntax

```
int SmSelectScript ( char    *scriptname,
                    char    *language,
                    SM_MSG *reply );
```

### Parameters

*scriptname*

input - The name of the script to select.

*language*

input - The language of the script.

*reply*

input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_BAD\_SCRIPT

### Reply Structure Functions by Message Type

SM\_SELECT\_SCRIPT\_REPLY  
SmGetNumberUtterances

## **Task Related Functions and Callbacks**

SmQueryScripts

SmRequestScriptText

SmNselectScriptCallback



## 3.83 SmSet

### Purpose

SmSet sets the value of a specified speech recognition engine parameter.

### Syntax

```
int SmSet ( short   item,
            long    value,
            SM_MSG *reply );
```

### Parameters

*item* input - The parameter set, which can be any one of the following:

*SM\_API\_DEBUG*

Values that can be set include the following: 0 is debugging off, 1 to 5 provide progressively more debug information. A value of 1 or 2 provides flow of control information.

*SM\_API\_DISPLAY*

Reserved.

*SM\_API\_LOG*

The values are:

- 0=Off
- 1=Message headers
- 2=Message headers + content
- 3=Message headers + content + internal engine messages
- Values above 3 are reserved.

*SM\_API\_TIMING*

Reserved.

*SM\_AUDIO\_DEVICE*

Selects the type of audio hardware.

*SM\_AUDIO\_CONFIGURATION*

(Not applicable to Windows. Instead use Audio Setup program.)  
Query the input source of the audio hardware where audio data is collected. The returned value is bit mapped and can be any combination of the following values:

- SM\_AUDIO\_INPUT\_MIC\_HI\_GAIN
- SM\_AUDIO\_INPUT\_MIC\_LO\_GAIN
- SM\_AUDIO\_INPUT\_LINE\_LEFT

- `SM_AUDIO_INPUT_LINE_RIGHT`
- `SM_AUDIO_INPUT_VARIABLE_GAIN`
- `SM_AUDIO_OUTPUT_LINE_LEFT`
- `SM_AUDIO_OUTPUT_LINE_RIGHT`
- `SM_AUDIO_OUTPUT_INTERNAL_SPEAKER`
- `SM_AUDIO_OUTPUT_VARIABLE_GAIN`

Refer to the `SMLIMITS.H` file for bounds on input and output values.

#### *SM\_AUDIO\_INPUT\_MODE*

(Not applicable to Windows. Instead use Audio Setup program.) Sets the input source of the audio hardware where audio data is collected. Only one value can be specified per call and values include the following:

- `SM_AUDIO_INPUT_LINE_LEFT`
- `SM_AUDIO_INPUT_LINE_RIGHT`
- `SM_AUDIO_INPUT_MIC_HI_GAIN`
- `SM_AUDIO_INPUT_MIC_LO_GAIN`

#### *SM\_AUDIO\_OUTPUT\_MODE*

(Not applicable to Windows. Instead use the Audio Setup program.) If supported by the underlying audio, this parameter indicates the current output destination. Only one value can be specified per call and values include the following:

- `SM_AUDIO_OUTPUT_LINE_LEFT`
- `SM_AUDIO_OUTPUT_LINE_RIGHT`
- `SM_AUDIO_OUTPUT_INTERNAL_SPEAKER`

#### *SM\_AUDIO\_INPUT\_GAIN*

(Not applicable to Windows. Instead use Audio Setup program.) If supported by underlying audio, this parameter allows you to set current gain setting. Refer to the `SMLIMITS.H` file for bounds on input and output values.

#### *SM\_AUDIO\_OUTPUT\_GAIN*

(Not applicable to Windows. Instead use Audio Setup program.) If supported by underlying audio, this parameter allows you to set current gain setting. Refer to the `SMLIMITS.H` file for bounds on input and output values.

#### *SM\_AVAILABLE\_AUDIO\_DEVICES*

For the default, `SPCH_DEFAULT_SYSTEM_AUDIO`. If using `MWAVE`, `SPCH_MWAVE_AUDIO`.

#### *SM\_COMMAND\_PHRASE\_TIMEOUT*

This timeout parameter is used to control the behavior of finite state grammars. In particular, they specify how much silence is

needed at the end of commands before the engine will make a recognition decision to accept or reject the command. The amount of silence required to accept or reject a phrase that is both partially complete and complete. Default value is 250 (csecs). It can be overridden by task dependent .par value. (Previously called `SM_PARTIAL_COMMAND_TIMEOUT`). For example, consider the grammar: `command = move up 3 move up five` Now consider the following acoustic scenarios: 1.`MOVE` 2.`MOVE UP` 3.`MOVE UP FIVE` 1.The engine will accept silences up to 0.5 seconds, waiting for the `UP`, before rejecting the phrase as incomplete. 2.The engine will accept silences up to 0.5 seconds, waiting for the optional `FIVE`, before accepting the phrase `MOVE UP`. 3.Then engine will wait only 0.1 seconds after the `FIVE` to accept the phrase. For the example given, it seems that you would like to increase the `SM_PARTIAL_COMMAND_TIMEOUT` in order to allow longer pauses within phrases. Note that increasing this parameter will also increase the decision time for "ambiguously complete" phrases.

#### *SM\_CONNECTION\_ID*

Sets the value of the `WPARAM` parameter for `WM_CONTROL` messages received from the engine. If `SM_PM_ENABLE` is `TRUE` all asynchronous API messages from the engine are `WM_CONTROL` messages. The `WPARAM` parameter is the value of the `SM_CONNECTION_ID`.

#### *SM\_DELAY\_EXIT*

Set the amount of time the engine waits before terminating after the last client disconnects. Valid values: -1 - `MAX_LONG`. If set to -1 the engine will not terminate after the last client disconnects. If set to 0 the engine will terminate immediately after the last client disconnects. For values of `x` where  $0 < x \leq \text{MAX\_LONG}$ , the engine will wait `x` seconds before terminating after the last client disconnects.

#### *SM\_ENABLE\_EXCLUSIVE\_VOCABS*

Sets the calling application's vocabularies. As part of the new global vocabulary mechanism, applications have been given control of these as well as the navigator vocabularies. The `SmSet` item `SM_ENABLE_EXCLUSIVE_VOCABS` now takes four values:

- `SM_EXCLUDE_NONE`
- `SM_EXCLUDE_ALL`
- `SM_EXCLUDE_GLOBALS`
- `SM_EXCLUDE_ALLBUTNAVGLOBAL`

The first two correspond to the old 0/1 values, so old clients will work as before. The third is not very useful but would let a client hide any of these new globals. The fourth provides the "super-global" function so that dictation clients can hide all vocabularies except those the navigator defines as global.

*SM\_ENGINE\_DEBUG*

Reserved. Values that can be set include the following: 0 is disabled, greater than 0 provides progressively more information.

*SM\_ENGINE\_DISPLAY*

Reserved. Values that can be set include the following: 0 is disabled, greater than 0 provides progressively more information.

*SM\_ENGINE\_LOG*

Reserved. Values that can be set include the following: 0 is disabled, greater than 0 provides progressively more information.

*SM\_ENGINE\_TIMING*

Reserved

*SM\_IMMEDIATE\_FIRMUP\_MODE*

This formerly unused item will now enable quick-firm-up mode, causing the engine to firm-up the next command or short phrase terminated by silence. It is reset when the first firm word is sent. Currently there are no new notifications, so the last 3 modes produce the same SM\_NOTIFY\_FOCUS\_APP\_EXCLUSIVE engine state notification:

- SM\_NOTIFY\_NAVIGATOR\_EXCLUSIVE
- SM\_NOTIFY\_FOCUS\_APP\_EXCLUSIVE
- SM\_NOTIFY\_NONE\_EXCLUSIVE

*SM\_NOTIFY\_AUDIO\_LEVEL*

Sets the returning of audio-level data during recognition or enrollment. The value is either TRUE (1) or FALSE (0). See [Section 4.6 \[SmGetAudioLevel\]](#), page 205.

*SM\_NOTIFY\_COMMAND\_WORD*

Sets whether the application is notified when a command word is recognized by having the engine send a SM\_COMMAND\_WORD reply message. The value is either TRUE (1) or FALSE (0).

*SM\_NOTIFY\_ENGINE\_STATE*

Sets whether the application is notified of a speech recognition engine state change. The value is either TRUE (1) or FALSE (0). See [Section 4.12 \[SmGetEngineState\]](#), page 211.

*SM\_NOTIFY\_FOCUS\_STATE*

Sets whether the application is notified of a speech focus state change. The value is either TRUE (1) or FALSE (0). See [Section 4.22 \[SmGetFocusState\]](#), page 222.

*SM\_NOTIFY\_MIC\_STATE*

Sets whether the application is notified of a microphone state change. The value is either TRUE (1) or FALSE (0). See [Section 4.28 \[SmGetMicState\]](#), page 228.

#### *SM\_OPTIMIZE\_PERFORMANCE*

Sets the speech recognition value for the engine depending on your requirements. Only one value can be specified per call. Valid values are:

- *SM\_OPTIMIZE\_SPEED* (fast)
- *SM\_OPTIMIZE\_DEFAULT* (balanced)
- *SM\_OPTIMIZE\_ACCURACY* (accurate)

#### *SM\_PHRASE\_ALTERNATIVES*

Sets the maximum number of alternatives which can be returned by the engine when querying phrase alternatives. Valid values: 1 - 100. See [Section 3.45 \[SmQueryPhraseAlternatives\]](#), page 105.

#### *SM\_REDUCED\_CPU\_MODE*

Sets the CPU mode of the speech recognition engine. The value is either TRUE (1) for reduced CPU mode, or FALSE (0) for normal CPU mode. See [Section 4.12 \[SmGetEngineState\]](#), page 211.

#### *SM\_REJECTION\_THRESHOLD*

The speech recognition engine rejects out-of-vocabulary words or background noise during monitored enrollment or command recognition. However, there is a trade-off between correctly rejecting these words/noises, and incorrectly rejecting properly spoken words.

*SM\_REJECTION\_THRESHOLD* allows an application to adjust this trade-off. The limits for the value of this parameter are *SM\_MIN\_REJECTION\_THRESHOLD* and *SM\_MAX\_REJECTION\_THRESHOLD* which are defined in *SMLIMITS.H*. At low settings of *SM\_REJECTION\_THRESHOLD* the engine allows more matches through with fewer rejections. This causes more background noises or incorrect word matches to be recognized as speech. Fewer correct words are rejected. At high rejection threshold values, the engine is biased toward requiring a closer match, resulting in more rejections, and potentially more rejections of correct words. This parameter also determines how close the match must be between spoken acoustics and pronunciations in the *SmAddPronunciation* and *SmWordCorrection* calls. See *SMLIMITS.H* for minimum and maximum limits.

#### *SM\_RELOAD\_ACOUSTICS*

This flag will be specified as an argument to the *SmSet* function. It will be used by a client application to request that the engine reload acoustic data after speaker clustering or microphone adaptation.

#### *SM\_SAVE\_AUDIO*

Enables the saving of speech data during recognition based on the following flags:

##### *SM\_SAVE\_AUDIO\_ADAPTATION*

Saves audio data required for training.

*SM\_SAVE\_AUDIO\_ALTERNATES*

Saves audio data required for alternates (SmQueryAlternates).

*SM\_SAVE\_AUDIO\_DEFAULT*

Saves all audio data for playback, add pronunciation and alternates. DEFAULT is mapped to PLAYBACK|ALTERNATES|TRAINWORD to preserve compatibility with older applications which asserted TRUE.

*SM\_SAVE\_AUDIO\_PLAYBACK*

Saves audio data required for playback (SmPlayWords)

*SM\_SAVE\_AUDIO\_PLAYBACKHIQ*

Saves audio data required for high quality playback.

*SM\_SAVE\_AUDIO\_TRAINWORD*

Saves audio data required for trainword (SmAddPronunciation).

A value of 0 will turn off all audio saving.

*SM\_TEXT\_PHRASE\_TIMEOUT*

This timeout parameter is used to specify the length of silence at the end of commands during dictation.

*SM\_UNAMBIGUOUS\_COMMAND\_PHRASE\_TIMEOUT*

This timeout parameter is used to control the behavior of finite state grammars. In particular, they specify how much silence is needed at the end of commands before the engine will make a recognition decision to accept or reject the command. It is the amount of silence required to accept or reject a phrase that is unambiguously complete. Default value is 25 (csecs). Can be overridden by task dependent .par value. (Previously called SM\_COMPLETE\_COMMAND\_TIMEOUT).

*SM\_SILENCE\_DETECTION*

Enables or disables silence detection. Valid values: 1, 0. If set to 1, silence detection enabled, if set to 0, silence detection disabled.

*value* input - The value to set the parameter to.

*reply* input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM

SM\_RC\_EALLOCS

SM\_RC\_EBADHANDLE

SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_BAD\_MODE  
SM\_RC\_BAD\_VALUE  
SM\_RC\_NOT\_YET  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR  
SM\_RC\_SERVER\_ERROR  
SM\_RC\_EXISTS\_IN\_NOTIFY  
SM\_RC\_NOT\_IN\_NOTIFY  
SM\_RC\_NOT\_VALID\_REQUEST

## Reply Structure Functions by Message Type

SM\_SET\_REPLY  
SmGetRcS  
mGetItemValue

## Task Related Functions and Callbacks

SmHaltRecognizer  
SmMicOff  
SmMicOn  
SmQuery  
SmRecognizeNextWord  
SmNhaltRecognizerCallback  
SmNmicOffCallback  
SmNmicOnCallback  
SmNqueryCallback  
SmNrecognizeNextWordCallback  
SmNrecognizedTextCallback  
SmNrecognizedWordCallback  
SmNsetCallback  
SmNutteranceCompletedCallback

## 3.84 SmSetArg

### Purpose

SmSetArg is a macro that fills an SmArg structure when given its components. This function sets the components of the arg parameter. The pointer to arg or to a list of similarly created arguments can then be passed to a number of functions, such as SmOpen and SmConnect. For further information on attributes, see [Chapter 7 \[Attributes\]](#), [page 313](#). Also, see "Establishing a Speech Session" in the SMAPI Developer's Guide.

### Syntax

```
void SmSetArg ( SmArg arg,  
                char *name,  
                long  value );
```

### Parameters

*arg*            input - The argument.  
*name*          input - The name of the attribute.  
*value*        input - The value of the attribute.

### Task Related Functions and Callbacks

SmConnect  
SmOpen  
SmNconnectCallback



### 3.85 SmSetBinary

#### Purpose

SmSetBinary sets the value of a specified speech recognition engine parameter. This function is used instead of SmSet to set values of arbitrary length and data type.

#### Syntax

```
int SmSetBinary ( short    item,
                  short    length,
                  void     *value,
                  SM_MSG *reply );
```

#### Parameters

- item*           input - The parameter to be set, which can be any one of the following:
- SM\_AUDIO\_SOURCE*  
Specifies setting value in audio library. Value depends on audio library implementation.
  - SM\_MNR\_VALUE*  
Specifies setting mn timer value. Value is hexadecimal representation of mn timer.
  - SM\_SIGNAL\_NOISE*  
Specifies setting signal/noise ratio values. Value is character string holding the following blank delimited values: signal to noise ratio in decibels, signal level in decibels, fraction of samples clipped.
- length*       input - The length in bytes of the data pointed to by the argument value.
- value*         input - Points to data to which item is to be set.
- reply*         input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
```

SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK

## Reply Structure Functions by Message Type

SM\_SET\_BINARY\_REPLY  
SmGetRc  
SmGetBinaryItemValue

## Task Related Functions and Callbacks

SmQueryBinary  
SmNsetBinaryCallback

### 3.86 SmSetDefault

#### Purpose

SmSetDefault sets a default value for a user, enrollment, task ID speech attribute, or the default topics. The default user, enrollment, or task ID is used by the speech recognition engine when SM\_USE\_DEFAULT is specified for the speech attribute as arguments when calling SmOpen and SmConnect. The default topics are always used by the speech recognition engine.

The default enrollid, taskid, and topics are stored on a per-user basis. SmSetDefault called with an item value of SM\_DEFAULT\_USERID will set the value of the default userid and will change the current topics for all sessions. SmSetDefault called with other item values will set the default enrollid, taskid or topics for the default userid. To set the default enrollid, task or topics for a user other than the default user, the function SmSetUserDefault is provided.

#### Syntax

```
int SmSetDefault ( long    item,
                  char    *item_value,
                  SM_MSG *reply );
```

#### Parameters

- item*           input - Type of default ID speech attribute. Valid values include the following:
- SM\_DEFAULT\_USERID
  - SM\_DEFAULT\_ENROLLID
  - SM\_DEFAULT\_TASK
  - SM\_DEFAULT\_TOPICS
- item\_value*   input - Value for default ID speech attribute.
- reply*           input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
```

SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK

## Reply Structure Functions by Message Type

SM\_SET\_DEFAULT\_REPLY  
SmGetRc

## Task Related Functions and Callbacks

SmConnect  
SmOpen  
SmQueryDefault  
SmQueryUserDefault  
SmSetUserDefault  
SmNqueryDefaultCallback  
SmNqueryUserDefaultCallback  
SmNsetDefaultCallback  
SmNsetUserDefaultCallback

### 3.87 SmSetDirectory

#### Purpose

SmSetDirectory sets the name of files or directories used by the engine. In this release, SmSetDirectory defines the output.wav file name, used when SmPlayWords is subsequently called with the SM\_PLAY\_WORDS\_SAVE\_WAVFILE option flag asserted. The fully qualified file name is passed in the directory\_name string.

#### Syntax

```
int SmSetDirectory( short  *file_class,
                   char   *directory_name,
                   SM_MSG *reply );
```

#### Parameters

*file\_class*     input - Reserved, must be set to 0.

*directory\_name*  
                 input - String containing the file/directory name being set.

*reply*           input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_BAD_VALUE
```

#### Reply Structure Functions by Message Type

```
SM_SET_DEFAULT_REPLY
SmGetRc
```

## **Task Related Functions and Callbacks**

SmConnect

SmOpen

SmQueryDefault

### 3.88 SmSetUserDefault

#### Purpose

SmSetUserDefault sets a default value for the enrollid, task id, or topics for a particular user. If the item SM\_DEFAULT\_TOPICS is specified, the item\_value argument must be a string of the default topics for the specified userid concatenated with blank delimiters (for example, "topic1 topic2 topic3") and will change the current topics for all sessions.

#### Syntax

```
int SmSetUserDefault ( char    *user_id,
                       long     item,
                       char     *item_value,
                       SM_MSG *reply );
```

#### Parameters

- user\_id*      Userid for which default is set.
- item*          Specifies which default is set. Valid values:
- SM\_DEFAULT\_ENROLLID*  
                Specifies setting default enrollid
  - SM\_DEFAULT\_TASK*  
                Specifies setting default task
  - SM\_DEFAULT\_TOPICS*  
                Specifies setting default topics.
- itemvalue*    New default value to set.
- reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
```

SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_SERVER\_FILE\_OPEN\_ERROR

## **Reply Structure Functions by Message Type**

SM\_SET\_USER\_DEFAULT\_REPLY

## **Task Related Functions and Callbacks**

SmConnect  
SmOpen  
SmQueryUserDefault  
SmNconnectCallback  
SmNqueryUserDefaultCallback  
SmNconnectCallback  
SmNsetUserDefaultCallback



### 3.89 SmSetUserInfo

#### Purpose

SmSetUserInfo sets user information. This function stores any information that pertains to a user ID or an enrollment ID. It is the responsibility of the speech-aware application to select a proper item name (tag) that uniquely identifies stored information. This item is used later by the SmQueryUserInfo function for retrieving the stored information. This function allows an application to associate string information with a particular user ID or enrollment ID.

#### Syntax

```
int SmSetUserInfo ( char    *user_id,
                    char    *enroll_id,
                    char    *itemname,
                    char    *itemvalue,
                    SM_MSG *reply );
```

#### Parameters

*user\_id*      input - The name of the user whose information is stored.

*enroll\_id*    input - The enrollment ID of the user whose information is to be stored. This parameter is NULL if the information stored is related to the user ID.

*itemname*    input - The name of the information item to be stored.

*itemvalue*   input - The value of the information item to be stored.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
 SM\_RC\_EALLOCS  
 SM\_RC\_EBADHANDLE  
 SM\_RC\_ENOCONN  
 SM\_RC\_ENOMEM  
 SM\_RC\_ENOMSG  
 SM\_RC\_ENOSERVER  
 SM\_RC\_EUNEXP  
 SM\_RC\_SM\_NOT\_OPEN

SM\_RC\_OK  
SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_BAD\_ENROLLID  
SM\_RC\_BAD\_ITEM  
SM\_RC\_BAD\_USERID

## **Reply Structure Functions by Message Type**

SM\_SET\_USER\_INFO\_REPLY  
SmGetEnrollId  
SmGetEnrollIds  
SmGetRc  
SmGetUserId  
SmGetUserIds

## **Task Related Functions and Callbacks**

SmQueryUserInfo  
SmQueryUsers  
SmNqueryUsersCallback  
SmNqueryUserInfoCallback  
SmNsetUserInfoCallback

## 3.90 SmSetUtteranceNumber

### Purpose

SmSetUtteranceNumber sets the current utterance number. This function can only be called when connected to the speech recognition engine for an enrollment session.

### Syntax

```
int SmSetUtteranceNumber ( long uttno,  
                           SM_MSG *reply );
```

### Parameters

*uttno*        input - The utterance number to be set.

*reply*        input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

### Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM  
SM\_RC\_EALLOCS  
SM\_RC\_EBADHANDLE  
SM\_RC\_ENOCONN  
SM\_RC\_ENOMEM  
SM\_RC\_ENOMSG  
SM\_RC\_ENOSERVER  
SM\_RC\_EUNEXP  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_OK  
SM\_RC\_BAD\_SCRIPT  
SM\_RC\_BAD\_UTTNO

### Reply Structure Functions by Message Type

SM\_SET\_UTTERANCE\_NUMBER\_REPLY

### Task Related Functions and Callbacks

SmDiscardUtterance  
SmQueryUtterances  
SmNsetUtteranceNumberCallback

### 3.91 SmUndefineVocab

#### Purpose

SmUndefineVocab deletes a dynamic vocabulary. This function deletes a vocabulary defined by a call to SmDefineVocab or SmDefineVocabEx. After the vocabulary is deleted, no calls to SmEnableVocab, SmDisableVocab, SmAddToVocab, or SmRemoveFromVocab can be made for the specified vocabulary. The specified vocabulary must be disabled to be deleted. SmUndefineVocab releases the image of the private area of the FSG file. SmUndefineVocab is valid only when the speech recognition engine is not decoding speech to text. See "Setting Up Vocabularies" in the SMAPI Developer's Guide for examples of conditions when the engine is not decoding speech to text.

#### Syntax

```
int SmUndefineVocab ( char    *vocab,
                     SM_MSG *reply );
```

#### Parameters

*vocab*            input - The name of the vocabulary undefined.

*reply*            input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

#### Return Values

```
SM_RC_DEALLOCATING_SH_MEM
SM_RC_EALLOCS
SM_RC_EBADHANDLE
SM_RC_ENOCONN
SM_RC_ENOMEM
SM_RC_ENOMSG
SM_RC_ENOSERVER
SM_RC_EUNEXP
SM_RC_SM_NOT_OPEN
SM_RC_OK
SM_RC_BAD_VOCAB
SM_RC_INVALID_PARM_MAX_LEN
SM_RC_NOT_VALID_REQUEST
```

## Reply Structure Functions by Message Type

SM\_UNDEFINE\_VOCAB\_REPLY

SmGetRc

SmGetVocabName

## Task Related Functions and Callbacks

SmAddToVocab

SmDefineVocab

SmDisableVocab

SmEnableVocab

SmQueryAddedWords

SmQueryEnabledVocabs

SmQueryVocabs

SmQueryWord

SmRemoveFromVocab

SmNaddToVocabCallback

SmNdefineVocabCallback

SmNdisableVocabCallback

SmNenableVocabCallbackS

SmNqueryAddedWordsCallback

SmNqueryEnabledVocabsCallback

SmNqueryVocabsCallback

SmNqueryWordsCallback

SmNremoveFromVocabCallback

SmNundefineVocabCallback

## 3.92 SmWordCorrection

### Purpose

SmWordCorrection corrects a misrecognized word. This function notifies the speech recognition engine that an incorrectly recognized word or sequence of words was corrected by the user. The user corrects the word by providing the correct spelling. This call specifies whether the pronunciation for this word is to be added to the user's personal pronunciation pool and whether the corrected spelling is to be added to the user's personal text vocabulary extension. The speech recognition engine uses this information to assist future recognition.

If an SmWordCorrection call fails and the call is immediately repeated with exactly the same parameters, the engine will automatically apply more lenient threshold parameters the second time. ( For a detailed description of the SM\_REJECTION\_THRESHOLD parameter see [Section 3.83 \[SmSet\], page 175.](#))

This call is valid only when the speech recognition engine is not decoding speech to text. See "Changing the Engine Decoding State" in the SMAPI Developer's Guide for examples of conditions when the engine is not decoding speech to text. **Please note:**

- For this call to be valid, the application must first enable audio saving through SmSet(SM\_SAVE\_AUDIO, TRUE). For more information on saving and restoring speech sessions, reference "Writing ViaVoice Applications to Save and Restore Audio" in the SMAPI Developer's Guide.
- If you get an RC\_OK return value for a synchronous call, you need to check the status by performing an SmGetStatus function.

### Syntax

```
int SmWordCorrection ( short   ntags,
                      long    tags[],
                      short   nspells,
                      char    *spellings[],
                      char    *pronunciations[],
                      short   new_pronunciation,
                      SM_MSG *reply );
```

### Parameters

<i>ntags</i>	input - The number of tagged words corrected.
<i>tags</i>	input - An array of tags for the words corrected.
<i>nspells</i>	input - The number of new words to replace the words in error.
<i>spellings</i>	input - An array of new spellings for the words corrected.

*pronunciations*

input - An array of spoken-like spellings for the words that were corrected if their pronunciation is different than the spellings would suggest. Specify NULL if the spoken-like spelling is the same as the true spelling.

*new\_pronunciation*

input - A flag to indicate whether the corrected word is a new pronunciation for a word already in the recognition vocabulary. The following flag values are valid:

*SM\_CHECK\_SPELLING*

Check the spelling of the new word before adding it to the recognition vocabulary as a new word. SM\_CHECK\_SPELLING is valid only with SM\_DEFAULT\_ADDWORD. SM\_FORCE\_ADDWORD and SM\_REPLACE\_ADDWORD override the spelling check.

*SM\_FORCE\_ADDWORD*

The corrected word is a new pronunciation and needs to be added as a new word even if the word already exists in the vocabulary.

*SM\_NO\_ADDWORD*

The corrected word need not be added to the recognition vocabulary as a new word.

*SM\_DEFAULT\_ADDWORD*

The user needs to decide whether to add the corrected word to the recognition vocabulary. The speech recognition engine produces a pronunciation if one does not already exist.

*SM\_REPLACE\_ADDWORD*

The new pronunciation needs to replace any existing pronunciations of the corrected word.

*reply*

input/output - The pointer to a reply structure or to SmAsynchronous indicating that the call is made asynchronously.

## Return Values

SM\_RC\_DEALLOCATING\_SH\_MEM

SM\_RC\_EALLOCS

SM\_RC\_EBADHANDLE

SM\_RC\_ENOCONN

SM\_RC\_ENOMEM

SM\_RC\_ENOMSG

SM\_RC\_ENOSERVER

SM\_RC\_EUNEXP

SM\_RC\_SM\_NOT\_OPEN

SM\_RC\_OK

SM\_RC\_INVALID\_PARM\_MAX\_LEN  
SM\_RC\_RECORDING\_REQUIRED  
SM\_RC\_SM\_NOT\_OPEN  
SM\_RC\_MULTIPLE\_SPELLINGS  
SM\_RC\_NOT\_VALID\_REQUEST  
SM\_RC\_RECORDING\_REQUIRED

## Reply Structure Functions by Message Type

SM\_WORD\_CORRECTION\_REPLY  
SmGetPronunciations  
SmGetRc  
SmGetSpellings  
SmGetStatus  
SmGetTags

## Task Related Functions and Callbacks

SmCorrectText  
SmCorrectTextCancel  
SmNewContext  
SmNcorrectTextCallback  
SmNcorrectTextCancelCallback  
SmNnewContextCallback  
SmNplayWordsCallback  
SmNplayWordsStatusCallback  
SmNqueryAddedWordsCallback  
SmNqueryWordsCallback  
SmNremovePronunciationCallback  
SmNwordCorrectionCallback  
SmPlayWords  
SmQueryAddedWords  
SmQueryWord  
SmRemovePronunciation



## 4 Data Access Functions

This chapter lists and describes the function calls that retrieve data from reply messages. These function calls do not interact with the engine; they provide local access to the logical contents of a message that has already been received.

### 4.1 SmGetAlphabets

#### Purpose

SmGetAlphabets retrieves a list of alphabets related to specific enroll IDs or tasks.

SmGetAlphabets retrieves a list of alphabet strings. This function can extract data from the following reply message structures:

- SM\_QUERY\_ENROLLIDS\_REPLY
- SM\_QUERY\_TASKS\_REPLY

The alphabet identifies the base technology used in the engine. Enrollment data must match engine technology. The current engine is using a ranks based technology (alphabet="R"). The previous engine used a Z-label based technology (alphabet="Z").

#### Syntax

```
int SmGetAlphabets ( SM_MSG      reply,
                    unsigned long *nalphabets,
                    char          ***alphabets );
```

#### Parameters

*reply* input - The reply structure from a SMAPI function.

*nalphabets* output - The pointer to the number of alphabets.

*alphabets* output - The pointer to a list of alphabets.

#### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

#### Task Related Functions and Callbacks

```
SmQueryEnrollIds
SmQueryTasks
```

## 4.2 SmGetAlternates

### Purpose

SmGetAlternates retrieves alternative words.

SmGetAlternates retrieves the list of alternative words from the reply message. This function can extract data from the following reply message structures:

- SM\_QUERY\_ALTERNATES\_REPLY
- SM\_RECOGNIZED\_WORD

### Syntax

```
int SmGetAlternates ( SM_MSG      reply,
                     unsigned long *nwords,
                     SM_WORD      **words );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nwords*      output - The pointer to the number of alternative words.

*words*        output - The pointer to a list of alternative words.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

### Reply Structure Functions by Message Type

SmQueryAlternates

## 4.3 SmGetAnnotations

### Purpose

SmGetAnnotations retrieves annotations.

SmGetAnnotations extracts the base vocabulary name from the reply message, uses this name to access the image of the private area of the FSG file, and uses the offsets stored in SM\_WORD to access the annotation data. This function can extract data from the following reply message structures:

- SM\_RECOGNIZED\_PHRASE
- SM\_QUERY\_PHRASE\_ALTERNATIVES\_REPLY

Annotations are stored in an SM\_ANNOTATION structure. SmGetAnnotations returns a pointer to an array of SM\_ANNOTATION structures. SM\_ANNOTATION is defined as:

```
struct _SM_ANNOTATION
{
    long    type;           // Type of annotation
    union
    {
        long    numeric;    // Return numeric annotations
        char    *string;    // Return string annotations
        void    *other;     // Reserved
        annodata;
    };
};
```

Types of annotations are defined as:

```
SM_ANNOTATION_NONE
SM_ANNOTATION_NUMERIC
SM_ANNOTATION_STRING
SM_ANNOTATION_OTHER (Reserved)
```

For every element of the array of SM\_WORD structures returned in a SM\_RECOGNIZED\_PHRASE or SM\_QUERY\_PHRASE\_ALTERNATIVES message there is a corresponding element in the SM\_ANNOTATION array. If there is no annotation associated with a particular word this is indicated by an annotation of type SM\_ANNOTATION\_NONE.

### Syntax

```
int SmGetAnnotations ( SM_MSG      reply,
                      unsigned long *nwords,
                      SM_ANNOTATION **annotations );
```

## Parameters

*reply*           input - The reply structure from a SMAPI function.  
*nwords*        output - The pointer to the number of annotations.  
*annotations*   output - The pointer to an annotation structure.

## Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_SERVER\_FILE\_READ\_ERROR

## Reply Structure Functions by Message Type

SmDefineGrammar  
SmUndefineVocab

## 4.4 SmGetApplication

### Purpose

SmGetApplication retrieves the name of the application.

SmGetApplication extracts the application name from the following reply messages:

- SM\_COMMAND\_WORD
- SM\_FOCUS\_STATE

### Syntax

```
int SmGetApplication ( SM_MSG    reply,
                      char      **application );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*application*       output - The pointer to the name of the application.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure Functions by Message Type

SmNcommandWordCallback  
SmNfocusStateCallback

## 4.5 SmGetApplications

### Purpose

SmGetApplications retrieves the names of the applications that have established a session.

SmGetApplications extracts the application names from the SM\_QUERY\_SESSIONS\_REPLY reply message sent by the speech recognition engine to the application.

### Syntax

```
int SmGetApplications ( SM_MSG      reply,
                        unsigned long *nsessions,
                        char          ***reco_sessions );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nsessions*      output - A pointer to the number of sessions that have been established with the speech recognition engine.

*reco\_sessions*        output - A pointer to a list of applications that have established sessions with the speech recognition engine.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure Functions by Message Type

SmNquerySessionsCallback

## 4.6 SmGetAudioLevel

### Purpose

SmGetAudioLevel retrieves the level of the audio signal during recognition. SmGetAudioLevel provides information about the level of audio received by the speech recognition engine when the microphone is on. If previously requested by the SM\_AUDIO\_INPUT\_LEVEL parameter supplied to the SmSet function, packets of audio level values are periodically sent to the speech recognition engine, which in turn calls SmNaudioLevelCallback and/or sends a message to your window procedure. Audio-level information can then be obtained using this function. Audio-level values range from 0(SM\_MIN\_AUDIO\_LEVEL) to 10(SM\_MAX\_AUDIO\_LEVEL) with acceptable volume ranges from 2 to 7. The average rate sent by the engine is 5 audio-level values per second. This function can extract data from the following reply message structures:

- SM\_AUDIO\_LEVEL

### Syntax

```
int SmGetAudioLevel ( SM_MSG  reply,
                     short  *volume );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*volume*      output - The pointer to a short. Volume can range from SM\_MIN\_AUDIO\_LEVEL to SM\_MAX\_AUDIO\_LEVEL inclusive.

### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

### Reply Structure - Related Functions and Callbacks

```
SmNaudioLevelCallback
```

## 4.7 SmGetBinaryItemValue

### Purpose

SmGetBinaryItemValue retrieves the value of item of arbitrary length and data type. SmGetBinaryItemValue can extract data from the following reply message structures:

- SM\_SET\_BINARY\_REPLY
- SM\_QUERY\_BINARY\_REPLY

### Syntax

```
int SmGetBinaryItemValue ( SM_MSG          sm_reply,
                          unsigned long    *item,
                          unsigned long    *length,
                          void             ***value );
```

### Parameters

*sm\_reply*    input - The reply structure from a SMAPI function.

*item*        output - A pointer to the name of the item.

*length*      output - A pointer to the length of the data pointed by the value argument.

*value*       output - A pointer to the value of the item.

### Return Values

SM\_RC\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

### Reply Structure Functions by Message Type

SmSetBinary  
 SmQueryBinary



## 4.8 SmGetCodePage

### Purpose

SmGetCodepage returns the Windows codepage which is to be used by the client application for sending/receiving character strings in/out of the engine. SmGetCodepage can extract data from the following reply message structures:

- SM\_CONNECT\_REPLY (Recognition and Enrollment Mode Only)

### Syntax

```
int SmGetCodepage ( SM_MSG      reply,  
                   unsigned long *codepage );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*codepage*        output - Returns the Windows codepage used for character strings in/out of the engine.

### Return Values

SM\_RC\_OK  
SM\_RC\_REPLY\_NULL  
SM\_RC\_SM\_EINVAL\_MSG\_TYPE

## 4.9 SmGetComment

### Purpose

SmGetComment retrieves a comment. It can extract data from the following reply message structure:

- SM\_REQUEST\_SCRIPT\_TEXT\_REPLY

### Syntax

```
int SmGetComment ( SM_MSG    reply,
                   char      **comment );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*comment*    output - The pointer to a comment from the script text.

### Return Values

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

## 4.10 SmGetConfidenceScores

### Purpose

SmGetConfidenceScores retrieves an array of confidence scores. It can extract data from the following reply message structures:

- SM\_RECOGNIZED\_PHRASE
- SM\_RECOGNIZED\_TEXT
- SM\_RECOGNIZED\_WORD

### Syntax

```
int SmGetConfidenceScores ( SM_MSG   reply,
                           unsigned long nscores,
                           short **scores );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nscores*     output - The number of elements in the array of confidence scores.

*scores*       output - The array of confidence scores.

### Return Values

SM\_RC\_OK  
SM\_RC\_REPLY\_NULL  
SM\_RC\_SM\_EINVAL\_MSG\_TYPE

## 4.11 SmGetDescriptions

### Purpose

SmGetDescriptions retrieves a list of enrollment enroll IDs, scripts, tasks, or users.

SmGetDescriptions can extract data from the following reply message structures:

- SM\_QUERY\_ENROLLIDS\_REPLY
- SM\_QUERY\_SCRIPTS\_REPLY SM\_QUERY\_TASKS\_REPLY
- SM\_QUERY\_USERS\_REPLY
- SM\_QUERY\_LANGUAGES\_REPLY
- SM\_QUERY\_TOPICS\_REPLY

### Syntax

```
int SmGetDescriptions ( SM_MSG      reply,
                       unsigned long *ndescriptions,
                       char          ***descriptions );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*ndescriptions*  
                 output - The pointer to the number of descriptions.

*descriptions*  
                 output - The pointer to a list of descriptions.

### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

### Reply Structure Functions by Message Type

```
SmQueryEnrollIds
SmQueryLanguages
SmQueryTasks
SmQueryUsers
```

## 4.12 SmGetEngineState

### Purpose

SmGetEngineState retrieves the engine state.

SmGetEngineState extracts the engine state from an SM\_ENGINE\_STATE reply message sent by the speech recognition engine to an application. SM\_ENGINE\_STATE reply messages are controlled by the SM\_NOTIFY\_ENGINE\_STATE parameter of the SmSet function.

### Syntax

```
int SmGetEngineState ( SM_MSG      reply,
                      unsigned long *engine_state );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*engine\_state*

output - The pointer to an unsigned long. Values include the following:

- SM\_NOTIFY\_NORMAL\_CPU
- SM\_NOTIFY\_REDUCED\_CPU
- SM\_NOTIFY\_ENGINE\_BUSY
- SM\_NOTIFY\_ENGINE\_IDLE
- SM\_NOTIFY\_NAVIGATOR\_EXCLUSIVE
- SM\_NOTIFY\_FOCUS\_APP\_EXCLUSIVE
- SM\_NOTIFY\_NONE\_EXCLUSIVE
- SM\_NOTIFY\_RECOGNIZED\_SPEECH
- SM\_NOTIFY\_SPEECH\_START
- SM\_NOTIFY\_SPEECH\_STOP
- SM\_NOTIFY\_SPEECH\_TOO\_HIGH
- SM\_NOTIFY\_SPEECH\_TOO\_LOW
- SM\_NOTIFY\_PRONUNCIATIONS\_ADDED
- SM\_NOTIFY\_PRONUNCIATIONS\_DELETED
- SM\_NOTIFY\_ENGINE\_SETTINGS\_CHANGED for the following parameters:
  - SM\_REJECTION\_THRESHOLD
  - SM\_AUDIO\_CONFIGURATION
  - SM\_AUDIO\_DEVICE

- SM\_AUDIO\_INPUT\_GAIN
- SM\_AUDIO\_INPUT\_MODE
- SM\_AUDIO\_OUTPUT\_GAIN
- SM\_AUDIO\_OUTPUT\_MODE
- SM\_NOTIFY\_APPLICATION\_CONNECTED
- SM\_NOTIFY\_APPLICATION\_DISCONNECTED

## Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

## Reply Structure Functions by Message Type

SmNengineStateCallback

## 4.13 SmGetEnrollId

### Purpose

SmGetEnrollId retrieves an enrollment ID.

SmGetEnrollId is valid only for reply message structures that contain one enrollment ID. Use SmGetEnrollIds for reply message structures with multiple enrollment IDs. SmGetEnrollId can extract data from the following reply message structures:

- SM\_CONNECT\_REPLY
- SM\_QUERY\_DEFAULT\_REPLY
- SM\_QUERY\_USER\_DEFAULT\_REPLY
- SM\_QUERY\_USER\_INFO\_REPLY
- SM\_SET\_USER\_INFO\_REPLY

### Syntax

```
int SmGetEnrollId ( SM_MSG   reply,
                   char    **enroll_id );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*enroll\_id*       output - The pointer to an enrollment ID.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure Functions by Message Type

SmQueryDefault

SmQueryEnrollIds

SmQueryUserInfo

SmSetUserInfo

## 4.14 SmGetEnrollIds

### Purpose

SmGetEnrollIds retrieves a list of enrollment IDs.

SmGetEnrollIds extracts data from the following reply message structures:

- SM\_QUERY\_DEFAULT\_REPLY
- SM\_QUERY\_SESSIONS\_REPLY
- SM\_CONNECT\_REPLY
- SM\_QUERY\_ENROLLIDS\_REPLY
- SM\_QUERY\_USER\_DEFAULT\_REPLY
- SM\_QUERY\_USER\_INFO\_REPLY
- SM\_SET\_USER\_INFO\_REPLY

### Syntax

```
int SmGetEnrollIds ( SM_MSG      reply,
                    unsigned long *nenroll_ids,
                    char          ***enroll_ids );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nenroll\_ids*       output - The pointer to the number of enrollment IDs.

*enroll\_ids*      output - The pointer to a list of enrollment IDs.

### Return Values

SM\_RC.SM\_EINVAL.MSG.TYPE  
 SM\_RC.OK  
 SM\_RC.REPLY\_NULL

### Reply Structure Functions by Message Type

SmQueryEnrollIds  
 SmQueryUserInfo  
 SmSetUserInfo



## 4.15 SmGetEventId

### Purpose

SmGetEventId retrieves the event ID.

SmGetEventId can extract data from the following reply message structures:

- SM\_EVENT\_NOTIFY\_REPLY
- SM\_EVENT\_SYNCH

### Syntax

```
int SmGetEventId ( SM_MSG      reply,  
                  unsigned long *event_id );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*event\_id*    output - The pointer to the event ID.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure Functions by Message Type

SmEventNotify  
SmNeventSynchCallback

## 4.16 SmGetEventOptions

### Purpose

SmGetEventOptions retrieves options for an event.

SmGetEventOptions can extract data from the following reply message structure:

- SM\_EVENT\_SYNCH

### Syntax

```
int SmGetEventOptions ( SM_MSG      reply,
                        unsigned long *options );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*options*     output - The pointer to options.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure Functions by Message Type

SmNeventSynchCallback

## 4.17 SmGetExpectedRecordingSpace

### Purpose

SmGetExpectedRecordingSpace retrieves the approximate number of bytes of disk space that will be used by the speech recognition engine for recording during enrollment. This function is limited by what can be returned in an unsigned long. Actual recording space may be greater. This function can extract data from the following reply message structure:

- SM\_CONNECT\_REPLY ( Enrollment mode only )

### Syntax

```
int SmGetExpectedRecordingSpace ( SM_MSG      reply,
                                unsigned long *expected_space );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*expected\_space*  
                 output - The expected number of bytes required for recording.

### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

### Reply Structure Functions by Message Type

```
SmConnect
```

## 4.18 SmGetExpectedTrainingSpace

### Purpose

SmGetExpectedTrainingSpace retrieves the approximate number of bytes of disk space that may be used by the training program. This function is limited by what can be returned in an unsigned long. Actual training space may be greater. This function can extract data from the following reply message structure:

- SM\_CONNECT\_REPLY ( Enrollment mode only )

### Syntax

```
int SmGetExpectedTrainingSpace ( SM_MSG      reply,
                                unsigned long *expected_space );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*expected\_space*  
                 output - The expected number of bytes required for training.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure Functions by Message Type

SmConnect

## 4.19 SmGetFirmWords

### Purpose

SmGetFirmWords retrieves firm words.

SmGetFirmWords extracts the SM\_WORD from the following reply message structures:

- SM\_RECOGNIZED\_PHRASE
- SM\_RECOGNIZED\_TEXT
- SM\_RECOGNIZED\_WORD
- SM\_COMMAND\_WORD
- SM\_QUERY\_PHRASE\_ALTERNATIVES\_REPLY

### Syntax

```
int SmGetFirmWords ( SM_MSG      reply,
                    unsigned long *nwords,
                    SM_WORD      **words );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nwords*      output - The pointer to the number of firm words.

*words*        output - The pointer to a list of firm words.

### Return Values

SM\_RC.SM\_EINVAL.MSG\_TYPE  
SM\_RC.OK  
SM\_RC.REPLY\_NULL

### Reply Structure Functions by Message Type

SmNrecognizedTextCallback  
SmNrecognizedWordCallback  
SmNcommandWordCallback

## 4.20 SmGetFlags

### Purpose

SmGetFlags is used to retrieve a single flags field from a reply structure. It is valid only for the following message types:

- SM\_AUTO\_COMPLETE\_REPLY
- SM\_CONNECT\_REPLY
- SM\_DISCARD\_SPEECH\_DATA\_REPLY
- SM\_QUERY\_PHRASE\_ALTERNATIVES\_REPLY
- SM\_QUERY\_PRONUNCIATIONS\_REPLY
- SM\_QUERY\_TOPICS
- SM\_QUERY\_TASKS\_REPLY

### Syntax

```
int SmGetFlags ( SM_MSG      reply,  
                 unsigned long *flags );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.  
*flags*        output - Receives flags field.

### Return Values

SM\_RC\_OK  
SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryPhraseAlternatives  
SmConnect  
SmAutoComplete  
SmDiscardSpeechData  
SmQuerAddedWordsEx  
SmQuerPhraseAlternatives  
SmQueryPronunciationsEx  
SmQueryTopics

## 4.21 SmGetFocusChangeReason

### Purpose

SmGetFocusChangeReason returns a reason code on focus changes.

SmGetFocusChangeReason extracts the reason from the SM\_FOCUS\_STATE reply structure. The change reason is only available when the focus state notification is for SM\_NOTIFY\_FOCUS\_RELEASE. The "change reason" flags can be:

#### **SM\_NOTIFY\_FOCUS\_CHANGE\_ON\_RELEASE**

The application released the focus by itself.

#### **SM\_NOTIFY\_FOCUS\_CHANGE\_ON\_REQUEST**

The application released focus due to request.

### Syntax

```
int SmGetFocusChangeReason ( SM_MSG          reply,  
                             unsigned long *reason );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*reason*      output - The pointer to the reason change.

### Return Values

SM\_RC\_OK

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmGetFocusState

SmNFocusStateCallback

## 4.22 SmGetFocusState

### Purpose

SmGetFocusState retrieves the focus state.

SmGetFocusState extracts the focus state from the SM\_FOCUS\_STATE reply message sent by the speech recognition engine to an application. SM\_FOCUS\_STATE reply messages are controlled by the SmSet SM\_NOTIFY\_FOCUS\_STATE parameter. The name of the application causing the focus change can be extracted from SM\_FOCUS\_STATE with SmGetApplication.

### Syntax

```
int SmGetFocusState ( SM_MSG      reply,
                     unsigned long *focus_state );
```

### Parameters

*reply*            input - The reply structure from an API function.

*focus\_state*    output - The pointer to an unsigned long. Values include the following:

- SM\_NOTIFY\_FOCUS\_REQUESTED
- SM\_NOTIFY\_FOCUS\_GRANTED
- SM\_NOTIFY\_FOCUS\_DENIED
- SM\_NOTIFY\_FOCUS\_RELEASED

### Return Values

SM\_RC\_OK  
 SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmGetFocusChangeReason  
 SmNfocusStateCallback



## 4.23 SmGetFreeSpace

### Purpose

SmGetFreeSpace retrieves the approximate number of bytes of free disk space on the disk used by the speech recognition engine during enrollment and by the training program. This function is limited by what can be returned in an unsigned long. Actual free space may be greater. This function extracts data from the following reply message structures:

- SM\_CONNECT\_REPLY ( Enrollment mode only )
- SM\_DISCONNECT\_REPLY ( Enrollment mode only )

### Syntax

```
int SmGetFreeSpace ( SM_MSG      reply,
                    unsigned long *free_space );
```

### Parameters

*reply*        input - The reply structure from an API function.

*free\_space*   output - The approximate number of bytes of free disk space.

### Return Values

SM\_RC\_OK  
 SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmConnect  
 SmDisconnect

## 4.24 SmGetGrammarPath

### Purpose

SmGetGrammarPath retrieves the full path name of an FSG file.

SmGetGrammarPath extracts data from the following reply message structure:

SM\_DEFINE\_GRAMMAR\_REPLY

### Syntax

```
int SmGetGrammarPath ( SM_MSG    reply,
                      char      **grammar_path );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*grammar\_path*  
                 output - The pointer to the full path name.

### Return Values

SM\_RC\_OK  
SM\_RC\_REPLY\_NULL  
SM\_RC\_SM\_EINVAL\_MSG\_TYPE

## Reply Structure - Related Functions and Callbacks

SmDefineGrammar

## 4.25 SmGetIncrements

### Purpose

SmGetIncrements is used to retrieve an array of increment values from a reply structure. It is valid only for the following message types:

- SM\_QUERY\_SCRIPTS\_REPLY

The increments values are the minimum number of sentences which must be recorded for that script before the training program can be run.

### Syntax

```
int SmGetIncrements ( SM_MSG      reply,
                      unsigned long *nincrements,
                      short          **increments );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nincrements*  
                 output - Receives number of elements in array of increments returned.

*increments*  
                 output - Receives array of increments.

### Return Values

SM\_RC\_OK  
SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_REPLY\_NULL

## Reply Structure Related Functions and Callbacks

SmQueryScripts

## 4.26 SmGetItemValue

### Purpose

SmGetItemValue retrieves the value for an item.

SmGetItemValue can extract data from the following reply message structures:

- SM\_QUERY\_REPLY
- SM\_SET\_REPLY

### Syntax

```
int SmGetItemValue ( SM_MSG      reply,
                    unsigned long *item,
                    unsigned long *value );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*item*        output - The pointer to the name of the item.

*value*       output - The pointer to the value of the item.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQuery

SmSet

## 4.27 SmGetLanguages

### Purpose

SmGetLanguages retrieves a list of languages.

SmGetLanguages can extract data from the following reply message structures:

- SM.CONNECT\_REPLY
- SM.QUERY\_ENROLLIDS\_REPLY
- SM.QUERY\_LANGUAGES\_REPLY
- SM.QUERY\_TASKS\_REPLY

### Syntax

```
int SmGetLanguages ( SM_MSG reply,  
                    unsigned long *nlanguages,  
                    char ***languages );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nlanguages*       output - The pointer to the number of languages.

*languages*      output - The pointer to a list of languages.

### Return Values

SM\_RC.SM\_EINVAL\_MSG\_TYPE  
SM\_RC.OK  
SM\_RC.REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryEnrollIds  
SmQueryLanguages  
SmQueryTasks  
SmConnect

## 4.28 SmGetMicState

### Purpose

SmGetMicState gets the microphone state.

SmGetMicState extracts the microphone state from one of the following messages sent by the speech recognition engine to the application:

- SM\_REPORT\_ENGINE\_ERROR
- SM\_FOCUS\_GRANTED
- SM\_MIC\_STATE

SM\_REPORT\_ENGINE\_ERROR is unsolicited, and SM\_FOCUS\_GRANTED and SM\_MIC\_STATE are sent if the application issues SmSet for SM\_NOTIFY\_FOCUS\_STATE and SM\_NOTIFY\_MIC\_STATE, respectively.

### Syntax

```
int SmGetMicState ( SM_MSG      reply,
                   unsigned long *mic_state );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*mic\_state*   output - Pointer to an unsigned long, which can be any of the following:

- SM\_NOTIFY\_MIC\_ON
- SM\_ENGINE\_MIC\_OFF

### Return Values

None

### Reply Structure - Related Functions and Callbacks

SmNmicStateCallback

## 4.29 SmGetMsgName

### Purpose

SmGetMsgName retrieves the name of the audio message.

SmGetMsgName can extract data from the following reply message structures:

- SM\_PLAY\_MESSAGE\_STATUS

### Syntax

```
int SmGetMsgName ( SM_MSG    reply,
                   char      **message_name );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*message\_name*        output - The pointer to the name of the audio message.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmNplayMessageStatusCallback

## 4.30 SmGetMsgType

### Purpose

SmGetMsgType retrieves the type of message from the reply structure associated with the input.

The values for SMAPI message types are defined in the SMCOMM.H file and in the section [Section A.2 \[SMAPI Message Types\]](#), page 333.

SmGetMsgType can extract data from all defined reply message structures.

### Syntax

```
int SmGetMsgType ( SM_MSG  reply,
                  int      *message_type );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*message\_type*  
                 output - The message type.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL



## 4.31 SmGetNameValue

### Purpose

SmGetNameValue retrieves a name and its value.

SmGetNameValue retrieves a name (tag) of user information and its corresponding value. This function can extract data from the following reply message structures:

- SM\_QUERY\_USER\_INFO\_REPLY

### Syntax

```
int SmGetNameValue ( SM_MSG    reply,
                    char      **name,
                    char      **value );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*name*            output - The pointer to the name.

*value*           output - The pointer to the value of the name.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmQueryUserInfo

## 4.32 SmGetNextAlternate

### Purpose

SmGetNextAlternate retrieves the next alternate from a reply structure. This function can extract data from the following reply message structures:

- SM\_QUERY\_USER\_INFO\_REPLY

This access function differs from others in that it is called multiple times against one reply structure. The first time it is called it will return information about the first alternate. Each successive time it is called it will return information about the next alternate. When it has returned information about all alternates in the reply structure it will set the *nwords* parameter to 0.

### Syntax

```
int SmGetNextAlternate ( SM_MSG      reply,
                        unsigned long *nwords,
                        SM_WORD      **words );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.  
*nwords*      output - The number of words in the words array.  
*words*       output - The list of words which make up the next alternate.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmQueryAlternates

## 4.33 SmGetNumberProcessed

### Purpose

SmGetNumberProcessed retrieves the number of sentences from the current script which have been trained. This function can extract data from the following reply message structures:

- SM\_QUERY\_UTTERANCES\_REPLY

### Syntax

```
int SmGetNumberProcessed ( SM_MSG      reply,
                           unsigned long *nprocessed );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nprocessed*    output - The number of sentences which have been trained.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryUtterances

## 4.34 SmGetNumberRecorded

### Purpose

SmGetNumberRecorded retrieves a list of the number of recorded sentences. This function can extract data from the following reply message structures:

- SM\_QUERY\_SCRIPTS\_REPLY

**Please note:** The information returned by this function may be invalid if any recording for supervised or unsupervised training has been done and a backup/restore user operation is performed before running the training program for that recorded data.

### Syntax

```
int SmGetNumberRecorded ( SM_MSG      reply,
                          unsigned long *nrecorded,
                          short         **recorded );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nrecorded*   output - A pointer size of the recorded list.

*recorded*    output - A pointer to the list of the number of recorded sentences. Each element in the list holds the number of recorded sentences for a particular script.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryScripts

## 4.35 SmGetNumberRequired

### Purpose

SmGetNumberRequired retrieves the minimum number of sentences which must be recorded for the current script before running the training program. This function can extract data from the following reply message structures:

- SM\_QUERY\_UTTERANCES\_REPLY

### Syntax

```
int SmGetNumberRequired ( SM_MSG      reply,
                          unsigned long *nrequired );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nrequired*      output - The number of sentences which must be recorded.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryUtterances

## 4.36 SmGetNumberUtterances

### Purpose

SmGetNumberUtterances retrieves the number of recorded utterances. This function can extract data from the following reply message structures:

- SM\_CONNECT\_REPLY (Enrollment mode only)
- SM\_QUERY\_UTTERANCES\_REPLY
- SM\_SELECT\_SCRIPT\_REPLY

### Syntax

```
int SmGetNumberUtterances ( SM_MSG      reply,  
                           unsigned long *nutterances );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nutterances*    output - The number of recorded utterances.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmConnect  
SmQueryUtterances  
SmSelectScript

## 4.37 SmGetNumberWordMsgs

### Purpose

SmGetNumberWordMsgs gets the number of words from a message.

SmGetNumberWordMsgs retrieves the number of SM\_RECOGNIZED\_WORD messages that have been sent since the last SM\_RECOGNIZE\_NEXT\_WORD function call request. The value returned is 0 or 1. This is used with a synchronous SM\_HALT\_RECOGNIZER\_REPLY message to tell if any unprocessed command words are still queued.

### Syntax

```
int SmGetNumberWordMsgs ( SM_MSG      *reply,
                          unsigned long *nwords );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nwords*      output - Pointer to an unsigned long that will be filled in with the number of words.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## 4.38 SmGetOptions

### Purpose

SmGetOptions is used to retrieve an options field from a reply structure. It is valid only for the following message types:

- SM\_DEFINE\_VOCABULARY\_REPLY

### Syntax

```
int SmGetOptions ( SM_MSG          *reply,
                  unsigned long *options );
```

### Parameters

*reply*        input reply structure from a SMAPI function.

*options*     output -output - receives options field.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmDefineVocabEx



## 4.39 SmGetPercentages

### Purpose

SmGetPercentages retrieves percentages of the completeness of processing for specified enrollments. SmGetPercentages can extract data from the following reply message structure:

- SM\_QUERY\_ENROLLIDS\_REPLY

A query of enrollids for a given user returns an array of enrollids, with related fields, including percent-complete. This procedure provides an array of the percent-complete values. The percent-complete values represent the amount of training which is complete for a given enrollid. When percent-complete is 100 the training program is finished.

### Syntax

```
int SmGetPercentages ( SM_MSG      reply,
                      unsigned long *npercentages,
                      short         **percentages );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*npercentages*        output - The pointer to the number of percentages.

*percentages*        output - The pointer to a list of percentages.

### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

### Reply Structure - Related Functions and Callbacks

```
SmQueryEnrollIds
```

## 4.40 SmGetPhoneticPronunciations

### Purpose

SmGetPhoneticPronunciations retrieves phonetic pronunciations from a reply. SmGetPhoneticPronunciations can extract data from the following reply message structure:

- SM\_QUERY\_PRONUNCIATION\_REPLY

### Syntax

```
int SmGetPhoneticPronunciations ( SM_MSG      reply,
                                unsigned long *nphonetics,
                                char          ***phonetics );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nphonetics*       output - The pointer to the number of phonetic pronunciations.

*phonetics*       output - The pointer to a list of phonetic pronunciations.

### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

### Reply Structure - Related Functions and Callbacks

SmQueryPronunciation

## 4.41 SmGetPhraseScore

### Purpose

SmGetPhraseScore is used to retrieve the phrase score from a reply structure. It is valid only for the following message types:

- SM\_RECOGNIZED\_PHRASE

### Syntax

```
int SmGetPhraseScore (SM_MSG reply, short *score);
```

### Parameters

*reply*            input - Reply structure from a SMAPI function.  
*score*            output - Receives score of phrase.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmNrecognizedPhraseCallback

## 4.42 SmGetPhraseState

### Purpose

SmGetPhraseState retrieves phrase state flags.

SmGetPhraseState can extract data from the following reply message structure:

- SM\_RECOGNIZED\_PHRASE

### Syntax

```
int SmGetPhraseState ( SM_MSG      reply,
                      unsigned long *phrase_state );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*phrase\_state*

output - The pointer to an unsigned long which can be either of the following:

You can get both of the following as output:

#### **SM\_PHRASE\_HALTED**

The engine is in a halted state and is waiting for the application to tell it what to do next.

#### **SM\_PHRASE\_ACCEPTED**

The engine has accepted the phrase.

You can get both of the following as output:

#### **SM\_PHRASE\_HALTED**

The engine is in a halted state and is waiting for the application to tell it what to do next.

#### **SM\_PHRASE\_REJECTED**

The engine has rejected the phrase, with the best guess phrase returned in the SM\_WORD array.

### Return Values

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

### Reply Structure - Related Functions and Callbacks

SmNRecognizedPhraseCallback

## 4.43 SmGetPreferred

### Purpose

SmGetPreferred is used to retrieve the number of preferred topics from a reply structure. It is valid only for the following message types:

- SM\_QUERY\_TOPICS\_REPLY

### Syntax

```
int SmGetPreferred ( SM_MSG  reply,
                    short  *preferred );
```

### Parameters

*reply*            input - Reply structure from a SMAPI function.

*preferred*       output - Receives number of preferred topics.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmQueryTopics

## 4.44 SmGetPronunciations

### Purpose

SmGetPronunciations retrieves pronunciations.

SmGetPronunciations can extract data from the following reply message structures:

- SM\_QUERY\_PRONUNCIATIONS\_REPLY
- SM\_WORD\_CORRECTION\_REPLY
- SM\_QUERY\_PRONUNCIATION\_REPLY
- SM\_RECOGNIZED\_TEXT

### Syntax

```
int SmGetPronunciations ( SM_MSG      reply,
                          unsigned long *npronun,
                          char          ***pronuns );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.  
*npronun*     output - The pointer to the number of pronunciations.  
*pronuns*     output - The pointer to a list of pronunciations.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryPronunciations  
SmWordCorrection

## 4.45 SmGetRc

### Purpose

SmGetRc retrieves the return code for the SMAPI functions.

### Syntax

```
int SmGetRc ( SM_MSG  reply,
              int      *rc );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*rc*            output - The pointer to the return code.

### Return Values

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

All functions.

## 4.46 SmGetRcDescription

### Purpose

SmGetRcDescription retrieves the ASCII string describing the return code. SmGetRcDescription remaps return codes to ASCII strings that describe the return code and, potentially, the associated failure. Messages are in US English only for diagnostic purposes. This function returns a pointer to an ASCII character string that describes the return code in the parameter rc\_description. For example, for the symbolic return code "SM\_RC\_ACOUSTICS\_TOO\_LONG", the associated return code description character string would be "The acoustics specified are too long." For a list of all return code character strings, see [Section A.3 \[SMAPI Message Explanations\]](#), page 337.

### Syntax

```
int SmGetRcDescription ( SM_MSG    reply,
                        char      **rc_description );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*rc\_description*  
                 output - Pointer to a character string.

### Return Values

SM\_RC\_OK  
SM\_RC\_REPLY\_NULL



## 4.47 SmGetRcName

### Purpose

SmGetRcName returns the symbolic name of the return code as a string.

SmGetRcName retrieves the ASCII strings containing the symbol associated with the return code as a string. For example, for a return code value of 49, the associated symbolic return code name character string would be SM\_RC\_ACOUSTICS\_TOO\_LONG. For a list of all return code values, see [Section A.1 \[SMAPI Return Codes and Messages\]](#), page 329.

### Syntax

```
int SmGetRcName ( SM_MSG    reply,
                  char      **rc_name );
```

### Parameters

*reply*            input - Any return code associated with any Sm call.

*rc\_name*        output - Pointer to a character string.

### Return Values

SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## 4.48 SmGetRequiredTrainingSpace

### Purpose

SmGetRequiredTrainingSpace retrieves the approximate number of bytes of disk space that is required by the training program. This function is limited by what can be returned in an unsigned long. Actual training space requirements may be greater. This function can extract data from the following reply message structure:

- SM\_DISCONNECT

### Syntax

```
int SmGetRequiredTrainingSpace ( SM_MSG      reply,
                                unsigned long *required_space );
```

### Parameters

*reply*            input - Reply structure from SMAPI function

*required\_space*  
                 output - Approximate number of bytes required by training.

### Return Values

SM\_RC.SM\_EINVAL.MSG\_TYPE  
SM\_RC\_OK  
SM\_RC.REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmDisconnect

## 4.49 SmGetSampleRates

### Purpose

SmGetSampleRates is used to retrieve an array of sample rates from a reply structure. It is valid only for the following message types:

- SM\_QUERY\_ENROLLIDS\_REPLY
- SM\_QUERY\_TASKS\_REPLY

### Syntax

```
int SmGetSampleRates ( SM_MSG      reply,
                      unsigned long *nrates,
                      short    **sample_rates );
```

### Parameters

*reply*            input - Reply structure from SMAPI function

*nrates*          output - Receives number of elements in array of sample\_rates.

*sample\_rates*    output - Receives array of sample\_rates.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryEnrollids  
 SmQueryTasks

## 4.50 SmGetScriptFlags

### Purpose

SmGetScriptFlags is used to retrieve an array of script flags from a reply structure. It is valid only for the following message types:

- SM\_QUERY\_SCRIPTS\_REPLY

### Syntax

```
int SmGetScriptFlags ( SM_MSG      reply,
                      unsigned long *nflags,
                      long          **flags );
```

### Parameters

*reply*        input - Reply structure from SMAPI function  
*nflags*      output - Receives number of elements in array of flags.  
*flags*        output - Receives array of flags.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmQueryScripts

## 4.51 SmGetScripts

### Purpose

SmGetScripts retrieves a list of enrollment scripts.

SmGetScripts can extract data from the following reply message structures:

- SM\_QUERY\_SCRIPTS\_REPLY
- SM\_QUERY\_ENROLLIDS\_REPLY

### Syntax

```
int SmGetScripts ( SM_MSG      reply,
                  unsigned long *nscripts,
                  char          ***scripts );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nscripts*    output - The pointer to the number of scripts.

*scripts*     output - The pointer to a list of scripts.

### Return Values

SM\_RC.SM\_EINVAL\_MSG\_TYPE

SM\_RC.OK

SM\_RC.REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryEnrollIds

## 4.52 SmGetService

### Purpose

SmGetService retrieves a pointer to the argument name describing the service requested from the speech recognition engine.

SmGetService can extract data from the following reply message structures:

- SM\_CONNECT\_REPLY
- SM\_DISCONNECT\_REPLY

### Syntax

```
int SmGetService ( SM_MSG    reply,
                  char    **service );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*service*        output - The pointer to the service name.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmConnect  
SmDisconnect

## 4.53 SmGetSessionId

### Purpose

SmGetSessionId retrieves the session ID from the reply structure.

SmGetSessionId can extract data from the following reply message structure:

- SM\_CONNECT\_REPLY

### Syntax

```
int SmGetSessionId ( SM_MSG    reply,
                    char    **session_id );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*session\_id*    output - The pointer to the ID of the session.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmConnect

## 4.54 SmGetSeverity

### Purpose

SmGetSeverity retrieves the severity associated with an unsolicited SM\_REPORT\_ENGINE\_ERROR message.

### Syntax

```
int SmGetSeverity ( SM_MSG      reply,
                   unsigned long *severity );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*severity*    output - Pointer to an unsigned long that will be filled in with the severity. Values include the following:

**SM\_ENGINE\_INFO**

Informational message/can be ignored.

**SM\_ENGINE\_WARNING**

Non-terminal error detected by engine.

**SM\_ENGINE\_ERROR**

Currently unused, but can be used in the future.

**SM\_ENGINE\_TERMINAL\_ERROR**

Engine terminated due to unrecoverable error.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL



## 4.55 SmGetSizes

### Purpose

SmGetSizes is used to retrieve an array of size values from a reply structure. It is valid only for the following message types:

- SM\_QUERY\_SCRIPTS\_REPLY

The size values represent the number of sentences in a script.

### Syntax

```
int SmGetSizes ( SM_MSG      reply,
                 unsigned long *nsizes,
                 short        **sizes );
```

### Parameters

*reply*        input - Reply structure from a SMAPI function.

*nsizes*      output - Receives number of elements in array of sizes Non-terminal error detected by engine.

*sizes*        output - Receives array of sizes.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryScripts

## 4.56 SmGetSpeechDataArchive

### Purpose

SmGetSpeechDataArchive retrieves the name of the archive file.

SmGetSpeechDataArchive extracts data associated with the following reply message structures:

- SM\_SAVE\_SPEECH\_DATA\_REPLY
- SM\_RESTORE\_SPEECH\_DATA\_REPLY

### Syntax

```
int SmGetSpeechDataArchive ( SM_MSG    reply,  
                             char      **archive );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*archive*     output - The pointer to the archive name.

### Return Values

SM\_RC\_OK  
SM\_RC\_REPLY\_NULL  
SM\_RC\_SM\_EINVAL\_MSG\_TYPE

## Reply Structure - Related Functions and Callbacks

SmSaveSpeechData  
SmRestoreSpeechData

## 4.57 SmGetSpeechDataOptions

### Purpose

SmGetSpeechDataOptions retrieves the options flags.

SmGetSpeechDataOptions extracts data associated with the following reply message structure:

- SM\_SAVE\_SPEECH\_DATA\_REPLY

### Syntax

```
int SmGetSpeechDataOptions ( SM_MSG  reply,
                             long    *options );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*options*     output - The pointer to the options flags, which can be one of the following:

#### **SM\_NORMAL\_FILE**

This indicates that the speech data was written to a flat file. It will contain only engine data.

#### **SM\_COMPOUND\_FILE**

This indicates that the speech data is written to a Windows compound file, and the data is saved in a section named "VTDSession-Archive." The file can contain both engine state and application state data.

### Return Values

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

### Reply Structure - Related Functions and Callbacks

SmSaveSpeechData

## 4.58 SmGetSpeechDataSize

### Purpose

SmGetSpeechDataSize retrieves the size of the speech data archive.

SmGetSpeechDataSize extracts data associated with the following reply message structures:

- SM\_SAVE\_SPEECH\_DATA\_REPLY
- SM\_QUERY\_SPEECH\_DATA\_REPLY

### Syntax

```
int SmGetSpeechDataSize ( SM_MSG      reply,  
                          unsigned long *size );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*size*            output - The pointer to the size of the speech data archive.

### Return Values

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

### Reply Structure - Related Functions and Callbacks

SmQuerySpeechData

SmSaveSpeechData

## 4.59 SmGetSpeechDataVersion

### Purpose

SmGetSpeechDataVersion retrieves the version identifier.

SmGetSpeechDataVersion extracts data associated with the following reply message structures:

- SM\_SAVE\_SPEECH\_DATA\_REPLY
- SM\_RESTORE\_SPEECH\_DATA\_REPLY

### Syntax

```
int SmGetSpeechDataVersion ( SM_MSG  reply,  
                             long    *version );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*version*     output - The pointer to the version identifier.

### Return Values

SM\_RC\_OK  
SM\_RC\_REPLY\_NULL  
SM\_RC\_SM\_EINVAL\_MSG\_TYPE

## Reply Structure - Related Functions and Callbacks

SmRestoreSpeechData  
SmSaveSpeechData

## 4.60 SmGetSpelling

### Purpose

SmGetSpelling gets the spelling from a message.

SmGetSpelling extracts data and retrieves the spelling associated with the following reply message structures:

- SM\_ADD\_PRONUNCIATION\_REPLY
- SM\_AUTO\_COMPLETE\_REPLY
- SM\_QUERY\_PRONUNCIATION\_REPLY
- SM\_QUERY\_SPELLING\_REPLY
- SM\_QUERY\_TOPICS\_REPLY
- SM\_QUERY\_USER\_DEFAULT\_REPLY
- SM\_REMOVE\_PRONUNCIATION\_REPLY

### Syntax

```
int SmGetSpelling ( SM_MSG    reply,
                   char      **spelling );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*spelling*    output - Pointer to a character string.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmAddPronunciation  
SmQueryPronunciation  
SmRemovePronunciation  
SmWordCorrection

## 4.61 SmGetSpellings

### Purpose

SmGetSpellings retrieves an array of spellings.

SmGetSpellings extracts data and retrieves the spelling associated with the following reply message structures:

- SM\_ADD\_PRONUNCIATION\_REPLY
- SM\_QUERY\_PRONUNCIATIONS\_REPLY
- SM\_REMOVE\_PRONUNCIATION\_REPLY
- SM\_WORD\_CORRECTION\_REPLY

### Syntax

```
int SmGetSpellings ( SM_MSG      reply,
                    unsigned long *nspellings,
                    char          ***spellings );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nspellings*   output - The pointer to the number of spellings.

*spellings*    output - The pointer to a list of spellings.

### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

### Reply Structure - Related Functions and Callbacks

```
SmAddPronunciation
SmQueryPronunciations
SmRemovePronunciation
SmWordCorrection
```

## 4.62 SmGetStates

### Purpose

SmGetStates retrieves enrollment statuses.

SmGetStates can extract data from the following reply message structure:

- SM\_QUERY\_ENROLLIDS\_REPLY
- SM\_QUERY\_SCRIPTS\_REPLY

### Syntax

```
int SmGetStates (SM_MSG reply, unsigned long *nstat, short **stat);
```

### Parameters

<i>reply</i>	input - The reply structure from a SMAPI function.
<i>nstat</i>	output - The pointer to the number of statuses.
<i>stat</i>	output - The pointer to a list of statuses. Values include the following: <ul style="list-style-type: none"><li>• SM_STAT_ENROLLMENT_RECORDING</li><li>• SM_STAT_ENROLLMENT_RUNNING</li><li>• SM_STAT_ENROLLMENT_COMPLETE</li><li>• SM_STAT_ENROLLMENT_FAILED</li><li>• SM_STAT_ENROLLMENT_BUSY</li></ul>

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure Functions by Message Type

SmQueryEnrollIds



## 4.63 SmGetStatus

### Purpose

SmGetStatus retrieves the status.

SmGetStatus can extract data from the following reply message structures:

- SM\_DISCARD\_SPEECH\_DATA\_REPLY
- SM\_PLAY\_MESSAGE\_STATUS
- SM\_PLAY\_UTTERANCE\_STATUS
- SM\_PLAY\_WORDS\_STATUS
- SM\_WORD\_CORRECTION\_REPLY

The SmGetStatus function retrieves the status from the reply structure. This status and its meaning depend on the message type of the reply structure. The following statuses are possible:

For SmNplayMessageCallback and SmNplayUtteranceCallback:

**SM\_STAT\_PLAY\_START**

The message, utterance, or word playback started.

**SM\_STAT\_PLAY\_STOP**

The message, utterance, or word playback stopped.

For SmNplayWordsStatusCallback:

**SM\_STAT\_PLAY\_START**

The message, utterance, or word playback started.

**SM\_STAT\_PLAY\_STOP**

The message, utterance, or word playback stopped.

**SM\_STAT\_BAD\_AUDIO**

The connection to the audio source was lost during playback.

**SM\_STAT\_BAD\_TAG**

An invalid tag value was specified for the word.

For an SmWordCorrection call:

**SM\_RC\_ADDED**

The word was added to the vocabulary.

**SM\_RC\_INVOCAB**

The word is already in the vocabulary.

**SM\_RC\_NOT\_INVOCAB**

The word is not in the IBM ViaVoice spelling dictionary.

**SM\_RC\_NOT\_ADDED**

The word has not been added to the vocabulary.

## Syntax

```
int SmGetStatus (SM_MSG reply, int *status);
```

## Parameters

*reply*           input - The reply structure from a SMAPI function.  
*status*           output - The pointer to the status.

## Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmWordCorrection  
SmNplayMessageStatusCallback  
SmNplayUtteranceStatusCallback  
SmNplayWordsStatusCallback

## 4.64 SmGetTagOffset

### Purpose

SmGetTagOffset retrieves the offset which was applied to all tags restored in a speech session when SmRestoreSpeechData was called with the SM\_MERGE\_TAGS flags set.

SmGetTagOffset can extract the tag offset from the following reply message structures:

- SM\_RESTORE\_SPEECH\_DATA\_REPLY

### Syntax

```
int SmGetTagOffset ( SM_MSG      reply,
                    unsigned long *offset );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*offset*           output - The offset to be added to merged tags.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmNrestoreSpeechDataCallback

## 4.65 SmGetTags

### Purpose

SmGetTags retrieves tags from the reply structure.

SmGetTags can extract data from the following reply message structures:

- SM\_PLAY\_WORDS\_STATUS
- SM\_QUERY\_ALTERNATES\_REPLY
- SM\_WORD\_CORRECTION\_REPLY

### Syntax

```
int SmGetTags ( SM_MSG      reply,
                unsigned long *ntags,
                long          **tags );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*ntags*       output - The pointer to the number of tags.

*tags*        output - The pointer to a list of tags.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryAlternates

SmWordCorrection

SmNplayWordsStatusCallback

## 4.66 SmGetTask

### Purpose

SmGetTask retrieves a domain from the reply structure.

SmGetTask is valid only for reply message structures that contain one domain. Use SmGetTasks for reply message structures with multiple domains. SmGetTask can extract data from the following reply message structures:

- SM\_CONNECT\_REPLY
- SM\_QUERY\_DEFAULT\_REPLY
- SM\_QUERY\_TASKS\_REPLY
- SM\_QUERY\_USER\_DEFAULT\_REPLY

### Syntax

```
int SmGetTask ( SM_MSG    reply,
                char      **task );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*domain*          output - The pointer to a domain.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryDefault

SmQueryTasks

## 4.67 SmGetTaskFlags

### Purpose

SmGetTaskFlags retrieves attribute information about tasks.

SmGetTaskFlags extracts the task attribute flags from the SM\_QUERY\_TASKS\_REPLY reply message sent by the speech recognition engine to the application

### Syntax

```
int SmGetTaskFlags ( SM_MSG      reply,
                    unsigned long *nflags,
                    unsigned long **flags );
```

### Parameters

- reply*        input - The reply structure from a SMAPI function.
- nflags*       output - A pointer to the number of flags in the list of return flags.
- flags*        output - A pointer to a list of task flags. Each flag in the list contains information about one or more of the following attributes of a task:
- SM\_TASK\_CONTINUOUS - This task supports continuous dictation
  - SM\_TASK\_NUMBER\_FMT - This task supports number formatting

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryTasks

## 4.68 SmGetTasks

### Purpose

SmGetTasks retrieves domains from the reply structure.

SmGetTasks extracts data from the following reply message structures:

- SM.CONNECT\_REPLY
- SM.QUERY\_DEFAULT\_REPLY
- SM.QUERY\_SCRIPTS\_REPLY
- SM.QUERY\_SESSIONS\_REPLY
- SM.QUERY\_TASKS\_REPLY
- SM.QUERY\_USER\_DEFAULT\_REPLY

### Syntax

```
int SmGetTasks ( SM_MSG      reply,
                 unsigned long *ntasks,
                 char          ***tasks );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*ndomains*   output - The pointer to the number of domains.

*domains*    output - The pointer to a list of domains.

### Return Values

SM\_RC.SM\_EINVAL.MSG\_TYPE

SM\_RC.OK

SM\_RC.REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmQueryTasks

## 4.69 SmGetTimes

This API call is ONLY here for compatibility purposes. If you are writing a new application, you should use SmGetWordTimes instead to return the start and end times for text, word, and phrase messages.

### Purpose

SmGetTimes retrieves time values of spoken words.

SmGetTimes retrieves an array of time values sent with words received in the last SM\_RECOGNIZED\_TEXT or SM\_RECOGNIZED\_WORD message from the speech recognition engine. Each time, expressed in milliseconds, specifies the time elapsed from the beginning of the utterance to the beginning of the spoken word. This function can extract data from the following reply message structures:

- SM\_RECOGNIZED\_TEXT
- SM\_RECOGNIZED\_WORD

### Syntax

```
int SmGetTimes ( SM_MSG      reply,
                 unsigned long *ntimes,
                 unsigned long **times );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*ntimes*       output - The pointer to the number of times in the list.

*times*        output - The pointer to a list of times.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmNrecognizedTextCallback  
 SmNrecognizedWordCallback  
 SmNaudioEventCallback



## 4.70 SmGetTopics

### Purpose

SmGetTopics is used to retrieve an array of topics from a reply structure. It is valid only for the following message types:

- SM\_QUERY\_TOPICS\_REPLY

### Syntax

```
int SmGetTopics ( SM_MSG      reply,
                  unsigned long *ntopics,
                  char          ***topics );
```

### Parameters

*reply*        input - reply structure from a SMAPI function.  
*ntopics*     output - receives number of elements in array of topics  
*trained*     output - receives array of topics.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmQueryTopics

## 4.71 SmGetTrained

### Purpose

SmGetTrained is used to retrieve an array of trained values from a reply structure. It is valid only for the following message types:

- SM\_QUERY\_SCRIPTS\_REPLY

The trained values represent the number of sentences of a script which have been recorded and processed by the training program.

### Syntax

```
int SmGetTrained ( SM_MSG      reply,
                  unsigned long *ntrained,
                  short         **trained);
```

### Parameters

*reply*            input - reply structure from a SMAPI function.

*ntrained*        output - receives number of elements in array of trained

*trained*        output - receives array of trained.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryScripts

## 4.72 SmGetUserId

### Purpose

SmGetUserId retrieves a user ID from the reply structure.

SmGetUserId is valid only for reply message structures that contain one user ID. Use SmGetUserIds for reply message structures with multiple user IDs. SmGetUserId can extract data from the following reply message structures:

- SM\_CONNECT\_REPLY
- SM\_QUERY\_DEFAULT\_REPLY
- SM\_QUERY\_USER\_INFO\_REPLY
- SM\_QUERY\_USER\_DEFAULT\_REPLY
- SM\_REQUEST\_NEW\_ENROLLID\_REPLY
- SM\_REQUEST\_NEW\_USERID\_REPLY
- SM\_SET\_USER\_INFO\_REPLY

### Syntax

```
int SmGetUserId ( SM_MSG reply,
                  char **user_id );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*user\_id*        output - The pointer to a user ID.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
 SM\_RC\_OK  
 SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryDefault  
 SmQueryUserInfo  
 SmQueryUsers  
 SmSetUserInfo

## 4.73 SmGetUserIds

### Purpose

SmGetUserIds retrieves user IDs from the reply structure.

SmGetUserIds can extract data from the following reply message structures:

- SM\_QUERY\_DEFAULT\_REPLY
- SM\_QUERY\_SESSIONS\_REPLY
- SM\_QUERY\_USER\_DEFAULT\_REPLY
- SM\_QUERY\_USER\_INFO\_REPLY
- SM\_QUERY\_USERS\_REPLY
- SM\_REQUEST\_NEW\_ENROLLID\_REPLY
- SM\_REQUEST\_NEW\_USERID\_REPLY
- SM\_SET\_USER\_INFO\_REPLY

### Syntax

```
int SmGetUserIds( SM_MSG      reply,
                  unsigned long *nuser_ids,
                  char          ***user_ids );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nuser\_ids*   output - The pointer to the number of user IDs.

*user\_ids*    output - The pointer to a list of user IDs.

### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

### Reply Structure - Related Functions and Callbacks

```
SmQueryUserInfo
SmQueryUsers
SmSetUserInfo
```

## 4.74 SmGetUsers

### Purpose

SmGetUsers retrieves users from the reply structure.

SmGetUsers can extract data from the following reply message structure:

- SM\_QUERY\_USERS\_REPLY
- SM\_REQUEST\_NEW\_USERID\_REPLY

### Syntax

```
int SmGetUsers ( SM_MSG      reply,
                 unsigned long *nusers,
                 char          ***users );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nusers*      output - The pointer to the number of users.

*users*        output - The pointer to a list of users.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmQueryUsers

## 4.75 SmGetUtteranceList

### Purpose

SmGetUtteranceList retrieves a list of utterances. It can extract data from the following reply message structure:

- SM\_QUERY\_UTTERANCES\_REPLY

### Syntax

```
int SmGetUtteranceList ( SM_MSG      reply,
                        unsigned long *nutterances,
                        long          **utterances );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*nutterances*        output - The pointer to the number of utterances.

*utterances*    output - The pointer to a list of utterances.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmQueryUtterances

## 4.76 SmGetUtteranceNumber

### Purpose

SmGetUtteranceNumber retrieves the utterance number from the reply structure.

SmGetUtteranceNumber can extract data from the following reply message structures:

- SM\_MIC\_ON\_REPLY
- SM\_PLAY\_UTTERANCE\_STATUS
- SM\_UTTERANCE\_COMPLETED

### Syntax

```
int SmGetUtteranceNumber ( SM_MSG      reply,
                           unsigned long *nutterance );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nutterance* output - The pointer to the utterance number.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmMicOn  
SmNplayUtteranceStatusCallback  
SmNutteranceCompletedCallback

## 4.77 SmGetVocabList

### Purpose

SmGetVocabList function retrieves a vocabulary list from the reply structure.

SmGetVocabList can extract data from the following reply message structures:

- SM\_QUERY\_ENABLED\_VOCABS\_REPLY
- SM\_QUERY\_VOCABS\_REPLY
- SM\_QUERY\_WORD\_REPLY
- SM\_SET\_VOCABS\_REPLY

### Syntax

```
int SmGetVocabList ( SM_MSG      reply,
                    unsigned long *nvocabs,
                    char          ***vocabs );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nvocabs*     output - The pointer to the number of vocabularies in the vocabulary list.

*vocabs*      output - The pointer to a list of vocabularies.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryEnabledVocabs

SmQueryVocabs

SmQueryWord



## 4.78 SmGetVocabName

### Purpose

SmGetVocabName retrieves the vocabulary name from the reply structure.

SmGetVocabName extracts data from the following reply message structures:

- SM\_ADD\_TO\_VOCAB\_REPLY
- SM\_DEFINE\_GRAMMAR\_REPLY
- SM\_DEFINE\_VOCAB\_REPLY
- SM\_DEFINE\_VOCABULARY\_REPLY
- SM\_DISABLE\_VOCAB\_REPLY
- SM\_ENABLE\_VOCAB\_REPLY
- SM\_QUERY\_PHRASE\_ALTERNATES\_REPLY
- SM\_RECOGNIZED\_PHRASE
- SM\_REMOVE\_FROM\_VOCAB\_REPLY
- SM\_UNDEFINE\_VOCAB\_REPLY

### Syntax

```
int SmGetVocabName ( SM_MSG reply,
                    char **vocabname );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

*vocabname*       output - The pointer to the name of the vocabulary.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmAddToVocab

SmDefineVocab

SmDisableVocab

SmEnableVocab

SmRemoveFromVocab

SmUndefineVocab

## 4.79 SmGetVocabPath

### Purpose

SmGetVocabPath is used to retrieve a vocabulary path from a reply structure. It is valid only for the following message types:

- SM\_DEFINE\_VOCABULARY\_REPLY

### Syntax

```
int SmGetVocabPath ( SM_MSG    reply,  
                    char      **vocab_path );
```

### Parameters

*reply*            input - reply structure from a SMAPI function.

*vocab\_path*    output - receives vocabulary path string.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE

SM\_RC\_OK

SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmDefineVocabEx

Related Functions

SmDefineVocabEx

## 4.80 SmGetVocWords

### Purpose

SmGetVocWords retrieves vocabulary words from the reply structure.

SmGetVocWords extracts data from the following reply message structures:

- SM\_ADD\_TO\_VOCAB\_REPLY
- SM\_DEFINE\_GRAMMAR\_REPLY
- SM\_DEFINE\_VOCAB\_REPLY
- SM\_DEFINE\_VOCABULARY\_REPLY
- SM\_QUERY\_VOCAB\_WORDS\_REPLY

Following an SmAddToVocab call, SmGetVocWords extracts the vocabulary words that failed to be added.

Following an SmDefineVocab call, SmGetVocWords extracts the vocabulary words that failed to be defined.

For a description of the SM\_VOCWORD data type, see [Section 6.6 \[SM\\_VOCWORD data type\]](#), [page 310](#).

### Syntax

```
int SmGetVocWords ( SM_MSG      reply,
                   unsigned long *nwords,
                   SM_VOCWORD  **words );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nwords*      output - The pointer to the number of vocabulary words. Limited to SM\_MAX\_VOCWORDS.

*words*       output - The pointer to a list of vocabulary words.

### Return Values

```
SM_RC_SM_EINVAL_MSG_TYPE
SM_RC_OK
SM_RC_REPLY_NULL
```

### Reply Structure - Related Functions and Callbacks

```
SmAddToVocab
SmDefineVocab
```

## 4.81 SmGetWords

### Purpose

SmGetWords retrieves words from the reply structure.

SmGetWords can extract data from the following reply message structures:

- SM\_QUERY\_ADDED\_WORDS\_REPLY
- SM\_QUERY\_WORD\_REPLY

### Syntax

```
int SmGetWords ( SM_MSG      reply,
                 unsigned long *nwords,
                 SM_WORD      **words );
```

### Parameters

*reply*        input - The reply structure from a SMAPI function.

*nwords*      output - The pointer to the number of words.

*words*       output - The pointer to a list of words.

### Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Reply Structure - Related Functions and Callbacks

SmQueryAddedWords  
SmQueryWord

## 4.82 SmGetWordTimes

### Purpose

SmGetWordTimes retrieves the most likely start and end times for recognized words from the reply structure.

During recognition, the speech recognition engine computes the most likely start and end times for each recognized word. These times are used for playback and for computing alternates for word correction.

The recognition engine keeps track of word start and end times in frame time offsets since the beginning of the utterance. The frame time is converted to wall time before returning the times to the application. On Windows the engine uses the Windows function GetTickCount to create the timestamp for the beginning of the utterance. On Unix the engine uses the function times. Frame time is essentially in centiseconds (hundredths of a second), but there can be slight differences in frame time versus clock time depending on the sample rate. For example, at 11025Hz sampling, a frame is represented by 110 samples, so there is a drift of 11000/11025 between frame time and clock time. In order to determine a word's start or end time offset from the beginning of the utterance, the application must subtract the word start or end time from the time returned in the SM\_SPEECH\_START engine state message. See [Section 3.83 \[SmSet\], page 175](#), SM\_NOTIFY\_ENGINE\_STATE for information on how to receive engine state messages. SmGetWordTimes can extract data from the following reply message structures:

- SM\_ADD\_PRONUNCIATION\_REPLY
- SM\_COMMAND\_WORD
- SM\_QUERY\_PHRASE\_ALTERNATIVES\_REPLY
- SM\_RECOGNIZED\_PHRASE
- SM\_RECOGNIZED\_TEXT
- SM\_RECOGNIZED\_WORD

### Syntax

```
int SmGetWordTimes ( SM_MSG      reply,
                    unsigned long *ntimes,
                    unsigned long **stimes,
                    unsigned long **etimes );
```

### Parameters

<i>reply</i>	input - The reply structure from one of the supported messages.
<i>ntimes</i>	output - The pointer to the number of times in the list.
<i>stimes</i>	output - The pointer to the list of start times.
<i>etimes</i>	output - The pointer to the list of end times.

## Return Values

SM\_RC\_SM\_EINVAL\_MSG\_TYPE  
SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

## Reply Structure - Related Functions and Callbacks

SmNrecognizedPhraseCallback  
SmNrecognizedTextCallback  
SmNrecognizedWordCallback  
SmAddPronunciation  
SmQueryPhraseAlternatives  
SmSet

## 4.83 SmReturnRc

### Purpose

SmReturnRc returns the return code from SM\_MSG.

SmReturnRc provides an alternative to SmGetRc for accessing return codes from messages received from the speech recognition engine. The return value of the function is the return code value in the given SM\_MSG.

### Syntax

```
int SmReturnRc ( SM_MSG reply );
```

### Parameters

*reply*            input - The reply structure from a SMAPI function.

### Return Values

SM\_RC\_OK  
SM\_RC\_REPLY\_NULL

### Task Related Functions and Callbacks

SmReturnRcDescription  
SmReturnRcName

## 4.84 SmReturnRcDescription

### Purpose

SmReturnRcDescription retrieves the ASCII string describing the return code. SmReturnRcDescription remaps return codes to ASCII strings that describe the return code and, potentially, the associated failure. Messages are in US English only for diagnostic purposes. For example, if this function were passed the return code, SM\_RC\_ACOUSTICS\_TOO\_LONG, a pointer to the associated return code description character string "The acoustics specified are too long." would be returned. For a list of all return code character strings, see [Section A.3 \[SMAPI Message Explanations\], page 337](#).

### Syntax

```
char *SmReturnRcDescription ( int rc );
```

### Parameters

*rc*                    input - Any return code associated with any Sm call.

### Task Related Functions and Callbacks

SmReturnRc

SmReturnRcName



## 4.85 SmReturnRcName

### Purpose

SmReturnRcName returns the symbolic name of a return code as a string. SmReturnRcName provides an alternative to SmGetRcName for retrieving an ASCII string containing the symbol associated with a return code. For example, if this function was passed the return code, SM\_RC\_ACOUSTICS\_TOO\_LONG, a pointer to the associated symbolic return code name character string "SM\_RC\_ACOUSTICS\_TOO\_LONG" would be returned. For a list of all return code values, see [Section A.1 \[SMAPI Return Codes and Messages\], page 329](#).

### Syntax

```
char *SmReturnRcName ( int rc );
```

### Parameters

*rc*            input - Any return code associated with any Sm call.

### Return Values

Function directly returns a pointer to a statically defined character string containing a description associated with the return code. Refer to Appendix A. "SMAPI Return Codes and Messages" for a complete list of the character strings, their associated return values, and their symbolic return code names.

### Task Related Functions and Callbacks

SmReturnRc  
SmReturnRcDescription



## 5 Reply Message Structures and Callbacks

This chapter describes reply structures and callbacks.

### 5.1 Reply Message Structures Received from the Speech Recognition Engine

An application can receive the following reply message structures, defined in SMCOMM.H, from the speech recognition engine.

Reply message structures received by the application that are in response to an API call request are termed solicited. Reply message structures received by the application that are not in response to an API call request are termed unsolicited.

Reply Message Structure	Sol./ Unsol.	Explanation
SM_AUTO_COMPLETE_REPLY	S	Response to an asynchronous SmAddComplete call. Data can be extracted with SmGetFlags, SmGetSpelling or SmGetSpellings.
SM_ADD_PRONUNCIATION_REPLY	S	Response to an asynchronous SmAddPronunciation call. Data can be extracted with SmGetSpelling or SmGetSpellings.
SM_ADD_TO_VOCAB_REPLY	S	Response to an asynchronous SmAddToVocab call. Data can be extracted with SmGetVocabName or SmGetVocWords.
SM_AUDIO_LEVEL	U	Controlled by SmSet SM_NOTIFY_AUDIO_LEVEL, this message provides the audio level data. Data can be extracted with SmGetAudioLevel.
SM_CANCEL_PLAYBACK_REPLY	S	Response to an asynchronous SmCancelPlayback call.

SM_COMMAND_WORD	U	Controlled by SmSet SM_NOTIFY_COMMAND_WORD, this message provides the name of the application that currently has focus as well as the last recognized input word. The SmGetFirmWords function extracts the SM_WORD and the SmGetApplication function extracts the application name.
SM_CONNECT_REPLY	S	Response to an asynchronous SmConnect call. Data can be extracted with SmGetEnrollId, SmGetEnrollIds, SmGetLanguages, SmGetMsgType, SmGetNumberUtterances, SmGetService, SmGetSessionId, SmGetTask, SmGetTasks, SmGetUserId, or SmGetUserIds.
SM_CORRECT_TEXT_REPLY	S	Response to an asynchronous SmCorrectText call.
SM_DEFINE_GRAMMAR_REPLY	S	Response to an asynchronous SmDefineGrammar call. Data can be extracted with SmSetGrammarPath.
SM_DEFINE_VOCAB_REPLY	S	Response to an asynchronous SmDefineVocab call. Data can be extracted with SmGetVocabName or SmGetVocWords.
SM_DEFINE_VOCABULARY_REPLY	S	Response to an asynchronous SmDefineVocab call. Data can be extracted with SmGetVocabName, SmGetVocWords, SmGetVocabPath, or SmGetOptions.
SM_DETACH_SESSIONS_REPLY	S	Response to an asynchronous SmDetachSessions call.
SM_DISCARD_DATA_REPLY	S	Response to an asynchronous SmDiscardData call.

SM_DISCARD_SPEECH_DATA_REPLY	S	Response to an asynchronous SmDiscardSpeechData call. Data can be extracted with SmGetFlags or SmGetStatus.
SM_DISCONNECT_REPLY	S	Response to an asynchronous SmDisconnect call. Data can be extracted with SmGetMsgType, or SmGetService.
SM_ENABLE_VOCAB_REPLY	S	Response to an asynchronous SmEnableVocab call. Data can be extracted with SmGetVocabName.
SM_ENGINE_STATE	U	Controlled by SmSetSM_NOTIFY_ENGINE_STATE. This message indicates current engine status. Data can be extracted with SmGetEngineState.
SM_EVENT_NOTIFY_REPLY	S	Response to an asynchronous SmEventNotify call. Data can be extracted with SmGetEventId.
SM_EVENT_SYNC	U	Notification of an occurrence of event specified in previous SmEventNotify. Data can be extracted with SmGetEventId or SmGetEventOptions.
SM_EVENT_TIME_REPLY	S	Reply to an asynchronous SmEventTime call. Data can be extracted with SmGetTimes or SmGetFlags.
SM_FOCUS_GRANTED	U	Asynchronous notification of focus granted. Data can be extracted with SmGetMicState.
SM_FOCUS_LOST	U	Asynchronous notification of focus lost.

SM.FOCUS.STATE	U	Controlled by SmSet SM_NOTIFY_FOCUS_STATE, this message shows a change in focus status and includes the name of the associated application. Data can be extracted with SmGetFocusState, SmGetApplication and SmGetFocusChangeReason.
SM.HALT_RECOGNIZER_REPLY	S	Response to an asynchronous SmHaltRecognizer call. Data can be extracted with SmGetNumberWordMsgs.
SM.MIC_OFF_REPLY	S	Response to an asynchronous SmMicOff call.
SM.MIC_ON_REPLY	S	Response to an asynchronous SmMicOn call. Data can be extracted with SmGetUtteranceNumber.
SM.REQUEST_MIC_OFF	U	Request to turn off the microphone.
SM.REQUEST_MIC_ON	U	Request to turn on the microphone.
SM.MIC_STATE	U	Controlled by SmSet SM_NOTIFY_MIC_STATE, this message indicated whether the microphone is on or off. Data can be extracted with SmGetMicState.
SM.NEW_CONTEXT_REPLY	S	Response to an asynchronous SmNewContext call.
SM.PLAY_MESSAGE_REPLY	S	Response to an asynchronous SmPlayMessage call. Assuming a successful call, SM_PLAY_MESSAGE_STATUS reply message structures will follow.

SM_PLAY_MESSAGE_STATUS	U	Notification of the start and completion of the audio playback initiated by an SmPlayMessage call. Data can be extracted with SmGetMsgName or SmGetStatus.
SM_PLAY_UTTERANCE_REPLY	S	Response to an asynchronous SmPlayUtterance call. Assuming a successful call, SM_PLAY_UTTERANCE_STATUS reply message structures will follow.
SM_PLAY_UTTERANCE_STATUS	U	Notification of the start and completion of the audio playback initiated by an SmPlayUtterance call. Data can be extracted with SmGetStatus or SmGetUtteranceNumber.
SM_PLAY_WORDS_REPLY	S	Response to an asynchronous SmPlayWords call. Assuming a successful call, SM_PLAY_WORDS_STATUS reply message structures will follow.
SM_PLAY_WORDS_STATUS	U	Notification of the start and completion of the audio playback initiated by an SmPlayWords call. Audio playback may be for one word or multiple words. Data can be extracted with SmGetStatus or SmGetTags.
SM_QUERY_ADDED_WORDS_REPLY	S	Response to an asynchronous SmQueryAddedWords call. Data can be extracted with SmGetWords.
SM_QUERY_ALTERNATES_REPLY	S	Response to an asynchronous SmQueryAlternates call. Data can be extracted with SmGetAlternates or SmGetTags.

SM_QUERY_DEFAULT_REPLY	S	Response to an asynchronous SmQueryDefault call. Data can be extracted with SmGetEnrollId, SmGetEnrollIds, SmGetScripts, SmGetTask, SmGetTasks, SmGetUserId or SmGetUserIds.
SM_QUERY_ENABLED_VOCABS_REPLY	S	Response to an asynchronous SmQueryEnabledVocabs call. Data can be extracted with SmGetVocabList.
SM_QUERY_ENROLLIDS_REPLY	S	Response to an asynchronous SmQueryEnrollIds call. Data can be extracted with SmGetAlphabets, SmGetDescriptions, SmGetEnrollIds, SmGetLanguages, SmGetPercentages, SmGetScripts, or SmGetStates.
SM_QUERY_LANGUAGES_REPLY	S	Response to an asynchronous SmQueryLanguages call. Data can be extracted with SmGetLanguages.
SM_QUERY_PHRASE_ALTERNATIVES_REPLY	S	Response to an asynchronous SmQueryPhraseAlternatives call. Data can be extracted with SmGetFirmWords, SmGetVocabName, SmGetAnnotations, SmGetWordTimes, and SmGetPhraseState.
SM_QUERY_PRONUNCIATION_REPLY	S	Response to an asynchronous SmQueryPronunciation call. Data can be extracted with SmGetSpelling or SmGetSpellings.
SM_QUERY_PRONUNCIATIONS_REPLY	S	Response to an asynchronous SmQueryPronunciationsEx call. Data can be extracted with SmGetPronunciations or SmGetSpellings.
SM_QUERY_REPLY	S	Response to an asynchronous SmQuery call. Data can be extracted with SmGetItemValue.



SM_QUERY_SESSIONS_REPLY	S	Response to an asynchronous SmQuerySessions call. Data can be extracted with SmGetApplications, SmGetEnrollIds, or SmGetUserIds.
SM_QUERY_SPEECH_DATA_REPLY	S	Response to an asynchronous SmQuerySpeechData call. Data can be extracted with SmGetSpeechDataSize.
SM_QUERY_SPEECH_USER_SIZE_REPLY	S	Response to an asynchronous SmQuerySpeechUserSize call. Data can be extracted with SmGetUserId, SmGetEnrollId, SmGetLanguages, SmGetScript, SmGetFlags, SmGetSpeechDataSize.
SM_QUERY_TASKS_REPLY	S	Response to an asynchronous SmQueryTasks call. Data can be extracted with SmGetAlphabets, SmGetDescriptions, SmGetLanguages or SmGetTasks.
SM_QUERY_TOPICS_REPLY	S	Response to an asynchronous SmQueryTopics call. Data can be extracted from SmGetTopics, SmGetPreferred, SmGetDescriptions, or SmGetFlags.
SM_QUERY_USER_DEFAULT_REPLY	S	Response to an asynchronous SmQueryUserDefault call. Data can be extracted with SmGetUserId, SmGetUserIds, SmGetEnrollId, SmGetEnrollIds, SmGetTask, SmGetTasks, or SmGetDefaultTopics.
SM_QUERY_USER_INFO_REPLY	S	Response to an asynchronous SmQueryUserInfo call. Data can be extracted with SmGetEnrollId, SmGetEnrollIds, SmGetNameValue, SmGetUserId, or SmGetUserIds.

SM_QUERY_USERS_REPLY	S	Response to an asynchronous SmQueryUsers call. Data can be extracted with SmGetDescriptions, SmGetUserIds, or SmGetUsers.
SM_QUERY_VOCABS_REPLY	S	Response to an asynchronous SmQueryVocabs call. Data can be extracted with SmGetVocabList.
SM_QUERY_WORD_REPLY	S	Response to an asynchronous SmQueryWord call. Data can be extracted with SmGetVocabList or SmGetWords.
SM_RECOGNIZED_PHRASE	U	Includes firm words sent during grammar vocabulary recognition. Data can be extracted with SmGetFirmWords, SmGetVocabName, SmGetPhraseState, SmGetAnnotations, SmGetWordTimes, and SmGetConfidenceScores.
SM_RECOGNIZED_TEXT	U	Includes firm words sent during dictation vocabulary recognition. Data can be extracted with SmGetFirmWords, SmGetTimes, SmGetWordTimes, and SmGetConfidenceScores.
SM_RECOGNIZED_WORD	U	Includes firm words sent during command vocabulary recognition. Data can be extracted with SmGetAlternates, SmGetFirmWords, SmGetTimes, SmGetWordTimes, and SmGetConfidenceScores.
SM_RECOGNIZE_NEXT_WORD_REPLY	S	Response to an asynchronous SmRecognizeNextWord call.
SM_RELEASE_FOCUS_REPLY	S	Response to an asynchronous SmReleaseFocus call.
SM_REMOVE_FROM_VOCAB_REPLY	S	Response to an asynchronous SmRemoveFromVocab call. Data can be extracted with SmGetVocabName.

SM.REMOVE_PRONUNCIATION_REPLY	S	Response to an asynchronous Sm-RemovePronunciation call. Data can be extracted with SmGetSpelling or SmGetSpellings.
SM.REPORT_ENGINE_ERROR	U	Notification that the speech recognition engine has encountered an error. Data can be extracted with SmGetMicState or SmGetSeverity.
SM.REQUEST_DETACH	U	Request to detach sessions.
SM.REQUEST_FOCUS_REPLY	S	Response to an asynchronous Sm-RequestFocus call.
SM.REQUEST_MIC_OFF_REPLY	S	Response to an asynchronous Sm-RequestMicOff call.
SM.REQUEST_MIC_ON_REPLY	S	Response to an asynchronous Sm-RequestMicOn call.
SM.RESTORE_SPEECH_DATA_REPLY	S	Response to an asynchronous Sm-RestoreSpeechData call. Data can be extracted using SmGetSpeechDataArchive.
SM.RESTORE_SPEECH_USER_REPLY	S	Response to an asynchronous Sm-RestoreSpeechUser call. Data can be extracted using SmGetSpeechDataArchive, SmGetUserId, SmGetEnrollId, SmGetLanguages, SmGetScript, or SmGetFlags.
SM.SAVE_SPEECH_DATA_REPLY	S	Response to an asynchronous Sm-SaveSpeechData call. Data can be extracted using SmGetSpeechDataArchive, SmGetSpeechDataOptions, or SmGetSpeechDataSize.

SM_SAVE_SPEECH_USER_REPLY	S	Response to an asynchronous SmSaveSpeechUser call. Data can be extracted using SmGetSpeechDataArchive, SmGetSpeechUserId, SmGetEnrollId, SmGetLanguages, SmGetScript, SmGetFlags, or SmGetSpeechDataSize.
SM_SET_DEFAULT_REPLY	S	Response to an asynchronous SmSetDefault call.
SM_SET_REPLY	S	Response to an asynchronous SmSet call. Data can be extracted with SmGetItemValue.
SM_SET_USER_DEFAULT_REPLY	S	Response to an asynchronous SmSetUserDefault call. No data returned.
SM_SET_USER_INFO_REPLY	S	Response to an asynchronous SmSetUserInfo call. Data can be extracted with SmGetEnrollId, SmGetEnrollIds, SmGetUserId or SmGetUserIds.
SM_UNDEFINE_VOCAB_REPLY	S	Response to an asynchronous SmUndefineVocab call. Data can be extracted with SmGetVocabName.
SM_UTTERANCE_COMPLETED	U	Notification that speech-to-text decoding of the audio input stream has been completed after the microphone was turned off. Data can be extracted with SmGetUtteranceNumber.
SM_WORD_CORRECTION_REPLY	S	Response to an asynchronous SmWordCorrection call. Data can be extracted with SmGetPronunciations, SmGetSpellings, SmGetStatus or SmGetTags.

## 5.2 Callbacks

The following callbacks are available for processing messages from the router and the speech recognition engine.

Callback	Sol./ Unsol.	Message Type
SmNautoCompleteCallback	S	SM_AUTO_COMPLETE_REPLY
SmNaddPronunciationCallback	S	SM_ADD_PRONUNCIATION_REPLY
SmNaddToVocabCallback	S	SM_ADD_TO_VOCAB_REPLY
SmNaudioLevelCallback	U	SM_AUDIO_LEVEL
SmNcancelPlaybackCallback	S	SM_CANCEL_PLAYBACK_REPLY
SmNcommandWordCallback	U	SM_COMMAND_WORD
SmNconnectCallback	S	SM_CONNECT_REPLY
SmNcorrectTextCallback	S	SM_CORRECT_TEXT_REPLY
SmNcorrectTextCancelCallback	S	SM_CORRECT_TEXT_CANCEL_REPLY
SmNdefineGrammarCallback	S	SM_DEFINE_GRAMMAR_REPLY
SmNdefineVocabCallback	S	SM_DEFINE_VOCAB_REPLY
SmNdefineVocabExCallback	S	SM_DEFINE_VOCABULARY_REPLY
SmNdetachRequestedCallback	U	SM_REQUEST_DETACH
SmNdetachSessionsCallback	S	SM_DETACH_SESSIONS_REPLY
SmNdisableVocabCallback	S	SM_DISABLE_VOCAB_REPLY
SmNdiscardDataCallback	S	SM_DISCARD_DATA_REPLY
SmNdiscardSpeechDataCallback	S	SM_DISCARD_SPEECH_DATA_REPLY
SmNdisconnectCallback	S	SM_DISCONNECT_REPLY
SmNenableVocabCallback	S	SM_ENABLE_VOCAB_REPLY
SmNengineStateCallback	U	SM_ENGINE_STATE
SmNeventNotifyCallback	S	SM_EVENT_NOTIFY_REPLY
SmNeventSynchCallback	U	SM_EVENT_SYNCH
SmNeventTimeCallback	S	SM_EVENT_TIME_REPLY
SmNfocusGrantedCallback	U	SM_FOCUS_GRANTED

SmNfocusLostCallback	U	SM_FOCUS_LOST
SmNfocusStateCallback	U	SM_FOCUS_STATE
SmNhaltRecognizerCallback	S	SM_HALT_RECOGNIZER_REPLY
SmNmicOffCallback	S	SM_MIC_OFF_REPLY
SmNmicOnCallback	S	SM_MIC_ON_REPLY
SmNmicOffRequestedCallback	U	SM_REQUEST_MIC_OFF
SmNmicOnRequestedCallback	U	SM_REQUEST_MIC_ON
SmNmicStateCallback	U	SM_MIC_STATE
SmNnewContextCallback	S	SM_NEW_CONTEXT_REPLY
SmNplayMessageCallback	S	SM_PLAY_MESSAGE_REPLY
SmNplayMessageStatusCallback	U	SM_PLAY_MESSAGE_STATUS
SmNplayUtteranceCallback	S	SM_PLAY_UTTERANCE_REPLY
SmNplayUtteranceStatusCallback	U	SM_PLAY_UTTERANCE_STATUS
SmNplayWordsCallback	S	SM_PLAY_WORDS_REPLY
SmNplayWordsStatusCallback	U	SM_PLAY_WORDS_STATUS
SmNqueryAddedWordsCallback	S	SM_QUERY_ADDED_WORDS_REPLY
SmNqueryAddedWordsExCallback	S	SM_QUERY_ADDED_WORDS_REPLY
SmNqueryAlternatesCallback	S	SM_QUERY_ALTERNATES_REPLY
SmNqueryCallback	S	SM_QUERY_REPLY
SmNqueryDefaultCallback	S	SM_QUERY_DEFAULT_REPLY
SmNqueryEnabledVocabsCallback	S	SM_QUERY_ENABLED_VOCABS_REPLY
SmNqueryEnrollIdsCallback	S	SM_QUERY_ENROLLIDS_REPLY
SmNqueryLanguagesCallback	S	SM_QUERY_LANGUAGES_REPLY
SmNqueryPhraseAlternativesCallback	S	SM_QUERY_PHRASE_ALTERNATIVES_REPLY
SmNqueryPronunciationCallback	S	SM_QUERY_PRONUNCIATION_REPLY
SmNqueryPronunciationsCallback	S	SM_QUERY_PRONUNCIATIONS_REPLY
SmNqueryPronunciationsExCallback	S	SM_QUERY_PRONUNCIATIONS_REPLY
SmNquerySessionsCallback	S	SM_QUERY_SESSIONS_REPLY
SmNquerySpeechDataCallback	S	SM_QUERY_SPEECH_DATA_REPLY

SmNquerySpeechUserSizeCallback	S	SM_QUERY_SPEECH_USER_SIZE_REPLY
SmNqueryTasksCallback	S	SM_QUERY_TASKS_REPLY
SmNqueryTopicsCallback	S	SM_QUERY_TOPICS_REPLY
SmNqueryUsersCallback	S	SM_QUERY_USERS_REPLY
SmNqueryUserDefaultCallback	S	SM_QUERY_USER_DEFAULT_REPLY
SmNqueryUserInfoCallback	S	SM_QUERY_USER_INFO_REPLY
SmNqueryVocabsCallback	S	SM_QUERY_VOCABS_REPLY
SmNqueryWordCallback	S	SM_QUERY_WORD_REPLY
SmNrecognizeNextWordCallback	S	SM_RECOGNIZE_NEXT_WORD_REPLY
SmNrecognizedPhraseCallback	U	SM_RECOGNIZED_PHRASE
SmNrecognizedTextCallback	U	SM_RECOGNIZE_TEXT
SmNrecognizedWordCallback	U	SM_RECOGNIZE_WORD
SmNreleaseFocusCallback	S	SM_RELEASE_FOCUS_REPLY
SmNremoveFromVocabCallback	S	SM_REMOVE_FROM_VOCAB_REPLY
SmNremovePronunciationCallback	S	SM_REMOVE_PRONUNCIATION_REPLY
SmNreportEngineErrorCallback	U	SM_REPORT_ENGINE_ERROR
SmNrequestFocusCallback	S	SM_REQUEST_FOCUS_REPLY
SmNrequestMicOffCallback	S	SM_REQUEST_MIC_OFF_REPLY
SmNrequestMicOnCallback	S	SM_REQUEST_MIC_ON_REPLY
SmNrestoreSpeechDataCallback	S	SM_RESTORE_SPEECH_DATA_REPLY
SmNrestoreSpeechUserCallback	S	SM_RESTORE_SPEECH_USER_REPLY
SmNsaveSpeechDataCallback	S	SM_SAVE_SPEECH_DATA_REPLY
SmNsaveSpeechUserCallback	S	SM_SAVE_SPEECH_USER_REPLY
SmNsetCallback	S	SM_SET_REPLY
SmNsetDefaultCallback	S	SM_SET_DEFAULT_REPLY
SmNsetUserDefaultCallback	S	SM_SET_USER_DEFAULT_REPLY
SmNsetUserInfoCallback	S	SM_SET_USER_INFO_REPLY
SmNundefineVocabCallback	S	SM_UNDEFINE_VOCAB_REPLY
SmNutteranceCompletedCallback	U	SM_UTTERANCE_COMPLETED

SmNwordCorrectionCallback	S	SM_WORD_CORRECTION_REPLY
---------------------------	---	--------------------------



### 5.3 Reply Structure Functions for Unsolicited Callbacks

The following is a list of all unsolicited callbacks available for processing messages from the speech recognition engine. Each callback is followed by a list of the reply structure functions that can be called from within it.

- **SmNaudioLevelCallback**
  - SmGetRc
  - SmGetAudioLevel
  - SmGetTimes
- **SmNcommandWordCallback**
  - SmGetRc
  - SmGetApplication
  - SmGetFirmWords
- **SmNdetachRequestedCallback**
  - SmGetRc
- **SmNengineStateCallback**
  - SmGetRc
  - SmGetengineState
  - SmGetTimes
- **SmNeventSynchCallback**
  - SmGetRc
  - SmGetEventId
  - SmGetEventOptions
- **SmNfocusGrantedCallback**
  - SmGetRc
- **SmNfocusLostCallback**
  - SmGetRc
- **SmNfocusStateCallback**
  - SmGetRc
  - SmGetFocusChangeReason
  - SmGetFocusState
  - SmGetApplication
- **SmNmicOffRequestedCallback**
  - SmGetRc
- **SmNmicOnRequestedCallback**
  - SmGetRc
- **SmNmicStateCallback**
  - SmGetRc

- SmGetMicState
- SmNplayMessageStatusCallback
  - SmGetRc
  - SmGetMsgName
  - SmGetStatus
- SmNplayUtteranceStatusCallback
  - SmGetRc
  - SmGetStatus
  - SmGetUtteranceNumber
- SmNplayWordsStatusCallback
  - SmGetRc
  - SmGetStatus
  - SmGetTags
- SmNrecognizedPhraseCallback
  - SmGetRc
  - SmGetAnnotations
  - SmGetFirmWords
  - SmGetVocabName
  - SmGetPhraseState
  - SmGetConfidenceScores
- SmNrecognizedTextCallback
  - SmGetRc
  - SmGetFirmWords
  - SmGetTimes
  - SmGetConfidenceScores
- SmNrecognizedWordCallback
  - SmGetRc
  - SmGetAlternates
  - SmGetFirmWords
  - SmGetTimes
  - SmGetConfidenceScores
- SmNreportEngineErrorCallback
  - SmGetRc
  - SmGetMicState
  - SmGetSeverity
- SmNutteranceCompletedCallback
  - SmGetRc
  - SmGetUtteranceNumber

## 6 Data Types

The following data types are used. They are listed in alphabetic order.

### 6.1 SmArg data type

Speech argument list structure.

```
typedef struct { char      name;  
                  SmArgVal value;  
} SmArg;
```

#### Fields:

<i>name</i>	Name of speech argument.
<i>value</i>	Value of speech argument if size of argument is less than or equal to size of (long); otherwise, a pointer to the speech argument value.

## 6.2 SmArgVal data type

Value of speech argument if size of argument is less than or equal to size of (long); otherwise, a pointer to the speech argument value.

```
typedef long SmArgVal;
```

## 6.3 SmHandler data type

Return value from callback-handler function.

```
typedef int SmHandler;
```

## 6.4 SM\_ANNOTATION data type

Annotation structure.

```
typedef struct _SM_ANNOTATION { long      type;
                                union {
                                    long      numeric;
                                    char      *string;
                                    void      *other;
                                } annodata;
};

typedef struct _SM_ANNOTATION SM_ANNOTATION;
```

### Fields:

<i>type</i>	The type of annotation.
<i>numeric</i>	Return numeric annotations.
<i>string</i>	Pointer to a string annotation.
<i>other</i>	RESERVED.

## 6.5 SM\_MSG data type

Pointer to message between speech recognition engine and application.

```
typedef void  SM_MSG;
```

## 6.6 SM\_VOCWORD data type

Vocabulary word-attribute structure.

```
typedef struct _SM_VOCWORD { long      flags;  
                             short     spelling_size;  
                             char      *spelling;  
                             };  
typedef struct _SM_VOCWORD  SM_VOCWORD;
```

### Fields:

*flags*            Reserved for internal use.

*spelling\_size*       Number of characters in word spelling plus one for terminating NULL.

*spelling*        Word spelling.



## 6.7 SM\_WORD data type

Word-attribute structure.

```
typedef struct _SM_WORD { long      tag;
                          long      flags;
                          short     score;
                          short     spelling_size;
                          char      *spelling;
                          short     vocab_size;
                          char      *vocab;
                          };
typedef struct _SM_WORD SM_WORD;
```

### Fields:

<i>tag</i>	Unique ID defined by speech recognition engine for this word.
<i>flags</i>	Formatting flags for words from predefined vocabularies.
<i>score</i>	Measure of word's probability. The score is relative and can vary from -100 to 100.
<i>spelling_size</i>	Number of characters in word spelling plus one for terminating NULL.
<i>spelling</i>	Word spelling.
<i>vocab_size</i>	Number of characters in vocabulary name plus one for terminating NULL.
<i>vocab</i>	Vocabulary name.



## 7 SMAPI Attributes

This chapter lists and describes the SMAPI attributes. These attributes describe the current state of your application's connection with the speech recognition engine (such as session type). You must set several of the SMAPI attributes when you open and connect to a speech session. You set the values of these attributes using `SmSetArg` or `SmSesSetArg`. When you set arguments, the engine creates an internal structure to track the state of your session. Refer to "Programming Tasks" in the SMAPI Developer's Guide for more information on the use of `SmSetArg`.

### 7.1 System Dependent Definition for Argument Lists

```
typedef long SmArgVal;

typedef struct { char      *name;           /* name of the argument */
                SmArgVal  value;          /* value of the argument */
                } SmArg;
```

### 7.2 Argument Attribute List

#### 7.2.1 Application Information Attributes

##### *SmNapplicationName*

Used to specify the name of your speech-aware application when initializing a recognition or enrollment session. The speech recognition engine will use this attribute to identify your application to other speech-aware applications.

#### 7.2.2 Requested Services

##### *SmNresetServices*

Set to true to specify that the type of session your application requests to the speech recognition engine be reset.

##### *SmNdatabase*

Set to true to specify a database session with the engine.

##### *SmNrecognize*

Set to true to specify a recognition session with the engine.

##### *SmNenrollment*

Set to true to specify an enrollment session with the engine.

### 7.2.3 Options Flags

*SmNdiscardSessionData*

Set to true to discard the engine's temporary directory at session close.

*SmNdiscardSessionAdaptation*

Set to true to discard language model cache updates.

*SmNsaveSessionData*

Set to true to save the engine's temporary directory at session close.

*SmNsaveSessionAdaptation*

Set to true to merge the language model updates into persistent storage.

*SmNcompleteEnrollment*

Set to true to begin the training program to modify the user's acoustic model when terminating an enrollment session.

*SmNsuspendEnrollment*

Set to true to not begin the training program when terminating an enrollment session.

### 7.2.4 External Notifier

*SmNwindowHandle*

On Windows set to the window handle in your application that will receive all speech-related messages.

*SmNexternalNotifier*

On Unix the SmNexternalNotifier attribute allows the speech recognition engine's asynchronous message dispatching loop to be integrated into an external main loop, for example: the GTK or Tcl/Tk dispatching mechanism. The SmNexternalNotifier takes as its argument the pointer to a callback function of the form:

```
int NotifierCallback( int    socket,
                     int (*recv)(),
                     void *data,
                     void *cinfo );
```

This callback function will be called when the engine initializes and begins to dispatch internal messages, and when the engine is shutting down. When the engine is shutting down, the recv function pointer will be NULL. During initialization, the application should register the socket with the main select or poll loop of the application and, upon the socket being ready for read, should call the recv function passing the data value as its argument. The cinfo parameter is connection specific information that can be set using the SmNexternalNotifierData parameter.

*SmNexternalNotifierData*

On Unix the SmNexternalNotifierData argument is used to specify application specific data which is passed to the external notifier function specified by the SmNexternalNotifier attribute.

## 7.2.5 User Definition Arguments

### *SmNaudioHost*

Used to specify the audio source when initializing a recognition or enrollment session. It can be set to use default values. Can also be used by a recognition, enrollment, or database session to specify which engine to start or connect with in a multiple speech engine environment. Refer to "Overview of the Custom Audio DLL's" in the SMAPI Developer's Guide for more information.

### *SmNconnectID*

Used to specify a connection identifier when initializing a recognition or enrollment session. Refer to "Establishing a Speech Session" in the SMAPI Developer's Guide for more information.

### *SmNenrollID*

Used to specify an enrollid when initializing a recognition or enrollment session. It can be set to use default values. Refer to "Establishing a Speech Session" in the SMAPI Developer's Guide for more information.

### *SmNenrollIdDescription*

Used to specify the description of an enrollid when initializing an enrollment session. Refer to "Establishing an Enrollment Session" in the SMAPI Developer's Guide for more information.

### *SmNnavigator*

Used to define a session as the Navigator session. Refer to "Navigator Session" in the SMAPI Developer's Guide for more information.

### *SmNscript*

Used to specify the name of a script when initializing an enrollment session. Refer to "Establishing an Enrollment Session" in the SMAPI Developer's Guide for more information.

### *SmNtask*

Used to specify a speech domain when initializing a recognition session. It can be set to use default values. Refer to "Establishing a Speech Session" in the SMAPI Developer's Guide for more information.

### *SmNuserid*

Used to specify a userid when initializing a recognition or enrollment session. It can be set to use default values. Refer to "Establishing a Speech Session" in the SMAPI Developer's Guide for more information.



## 8 SMAPI Grammar Compiler API Overview

This chapter describes the format of the Grammar Compiler API function calls that are presented in the chapter "Grammar Compiler API Function Calls". It also lists the calls that are provided by the Grammar Compiler.

### 8.1 Format of the Function Call Descriptions

The description of each function call contains the following information:

*Function Name*

The name of the function call.

*Purpose*

The purpose and description of the function call.

*Syntax*

The syntax of the function as declared in VTBNFC.H.

*Parameters*

Definitions of the parameters.

*Return Values*

Return values that are set by the Grammar Compiler.

### 8.2 Grammar Compiler API Function Calls

The following function calls are provided as part of the Grammar Compiler:

- VtAddArg
- VtCompileGrammar
- VtGetMessage
- VtGetTranslation
- VtLoadFSG
- VtSetArg
- VtUnloadFSG





## 9 SMAPI Grammar Compiler API Function Calls

This chapter lists and describes the SMAPI Grammar Compiler API function calls.

### 9.1 VtAddArg

#### Purpose

VtAddArg is a macro that adds an argument with the specified attributes to the end of a VtArg structure. This function sets the components of the arg parameter. The pointer to arg or to a list of similarly created arguments can then be passed to VtCompileGrammar. VtAddArg increments an index to point to the last argument in the argument list.

For further information on using the VtArg data structure, see [Chapter 10 \[SMAPI Grammar Compiler Data Types\]](#), page 327.

#### Syntax

```
void VtAddArg ( VtArg  arg,
                long   index,
                char   *name,
                long    value );
```

#### Parameters

<i>arg</i>	input - The argument.
<i>index</i>	output - The index into the argument structure. VtAddArg increments the index by one.
<i>name</i>	input - The name of the attribute.
<i>value</i>	input - The value of the attribute.

#### Return Values

None.

## 9.2 VtCompileGrammar

### Purpose

VtCompileGrammar compiles a BNF file and produces an FSG using the specified parameters. The Grammar Compiler parameters are provided by setting up an argument list (VtArg). For more information on creating and using the VtArg structure, see [Chapter 10 \[SMAPI Grammar Compiler Data Types\]](#), page 327.

### Syntax

```
int VtCompileGrammar ( int    nargs,
                      VtArg *args );
```

### Parameters

<i>nargs</i>	input - The number of arguments in the argument list.
<i>args</i>	input - The pointer to an argument structure that indicates the Grammar Compiler parameters to be used.

### Return Values

0 - Successful.  
Other - Use VtGetMessage to obtain more detailed error information.

## 9.3 VtGetMessage

### Purpose

VtGetMessage returns a pointer to a message string that describes the errors encountered during the last VtCompileGrammar call that returned a non-zero error code.

The messages returned by VtGetMessage are exactly the same messages (errors and warnings) that are generated by the Grammar Compiler and can be used to determine the cause of the errors and appropriate action to be taken by the application.

### Syntax

```
int VtGetMessage( char **message );
```

### Parameters

*message*     input - The pointer to a character string which holds the error information.

### Return Values

None.

## 9.4 VtGetTranslation

### Purpose

VtCompileGrammar gets a translation for a phrase contained in the "words" parameter according to the FSG file referred to by the "fsg" parameter.

### Syntax

```
int VtGetTranslation ( void  *fsg,
                      char **words,
                      char **translation );
```

### Parameters

*fsg*            input - The pointer to an FST file loaded by VtLoadFSG.

*words*        input - The pointer to a NULL-terminated array of pointers to words that you supply.

*translation*    input - The pointer to a character pointer that will hold the translation returned by VtGetTranslation.

### Return Values

0 - Successful.

Other - Use VtGetMessage to obtain more detailed error information.

### Example

```
char *name = // name of your .fsg file
void *fsg;
int rc = VtLoadFSG(name, &fsg);
char *translation;
int n = // number of words
char **words = new char*[n+1];
for (int i=0; i<n; i++)
    words[i] = // assign the ith word
words[n] = NULL;
rc = VtGetTranslation(fsg, words, &translation);
printf("translation is '%s'\n", translation);
rc = VtUnloadFSG(fsg);
```

## 9.5 VtLoadFSG

### Purpose

VtLoadFSG loads an FSG or FST file produced by the command-line compiler executable, or by the compiler API VtCompileGrammar. After loading a grammar with VtLoadFSG, you may use VtGetTranslation to get translations for recognized phrases returned by the ViaVoice recognition engine.

### Syntax

```
int VtLoadFSG ( char    *name,  
               void    **fsg );
```

### Parameters

*name*        input - The file name of the FST or FSG file you want to load.  
*fsg*        input - The pointer to a void\* to receive the loaded grammar.

### Return Values

0 - Successful.  
Other - Use VtGetMessage to obtain more detailed error information.

## 9.6 VtSetArg

### Purpose

VtSetArg is a macro that fills a VtArg structure with the specified attributes. This function sets the components of the arg parameter. The pointer to arg or to a list of similarly created arguments can then be passed to VtCompileGrammar. For further information on using the VtArg data type, see [Chapter 10 \[SMAPI Grammar Compiler Data Types\]](#), page 327.

### Syntax

```
void VtSetArg ( VtArg  arg,
                char   *name,
                long    value );
```

### Parameters

*arg*           input - The argument.  
*name*          input - The name of the attribute.  
*value*         input - The value of the attribute.

### Return Values

None.

## 9.7 VtUnloadFSG

### Purpose

VtUnloadFSG unloads an FSG or FST file that was previously loaded by VtLoadFSG.

### Syntax

```
void VtUnloadFSG ( void *fsg );
```

### Parameters

*fsg*            input - The pointer to an FST loaded by VtLoadFSG.

### Return Values

0 - Successful.

Other - Use VtGetMessage to obtain more detailed error information.





## 10 SMAPI Grammar Compiler Data Types

The following data type is used by the SMAPI Grammar Compiler APIs:

### 10.1 VtArg

Argument structure for VtCompileGrammar.

```
typedef struct {
    char *name;
    long value;
} VtArg;
```

#### Fields:

*name* Name of a VtCompileGrammar argument.

*value* Value of the VtCompileGrammar argument if the size of the argument is less than or equal to the size of (long); otherwise, a pointer to the VtCompileGrammar argument value.

VtArg is analogous to SmArg in SMAPI.; therefore, VtSetArg is analogous to SmSetArg. Possible arguments for VtSetArg (and VtAddArg) are:

#### **VtNbnfFile**

The input BNF file.

**VtfsgFile** The output FSG file. This is equivalent to -o on the command line.

#### **VtNfsgDirectory**

The output directory for multiple roots. This is equivalent to the -d parameter on the command line.

#### **VtNequalizeArcProbabilities**

The non-uniform probability computation. When set to 1, arc probabilities are equalized. (This is the default if not specified.) When set to 0, arc probabilities are not equalized. This argument is equivalent to the -n parameter on the command line.

#### **VtNfsgFlags**

SmDefineGrammar flags. The possible flag values are:

- SM\_PHRASE\_ALLOW\_SILENCES
- SM\_PHRASE\_SHOW\_SILENCES
- SM\_PHRASE\_NO\_SILENCES
- SM\_PHRASE\_ALLOW\_INSERTIONS
- SM\_PHRASE\_SHOW\_INSERTIONS
- SM\_PHRASE\_NO\_INSERTIONS

**VtNtrMode**

Specified with a parameter value of 1. It is used to compile grammars that contain translations to produce an FST file for use by VtLoadFSG. It is equivalent to the "-tr" for the VTBNFC command-line compiler.

**VtNenMode**

Specified with a parameter value of 1. It is used to compile grammars that contain translations to produce an FSG file for use by the engine (SmDefineGrammar). It is equivalent to the "-en" for the VTBNFC command-line compiler.

**Please note:**

For more information on command line parameters, refer to "Grammar Compiler" in the IBM SMAPI Developer's Guide.

The following example illustrates how to fill in a VtArg structure before calling VtCompileGrammar and how to use VtGetMessage if the return from VtCompileGrammar is not equal to zero:

```
int    n = 0;
VtArg args[10];
VtAddArg( args, n, VtNbnfFile, "mygrammar.bnf" );
VtAddArg( args, n, VtNfsgFlags, SM_PHRASE_SHOW_SILENCES |
                                     SM_PHRASE_ALLOW_INSERTIONS);
// Additional parameters specified by additional VtAddArg calls
int rc = VtCompileGrammar( n, args );
if ( rc !=0 ) {
    char *message;
    VtGetMessage( &message );
    printf( message );
}
```

## Appendix A SMAPI Return Codes, Messages, and Message Types

The return codes and messages, defined in SMRC.H, and the message types, defined in SMCOMM.H, are generated by the SMAPI.

### A.1 SMAPI Return and Status Codes

#### A.1.1 SMAPI Return Codes

The following list contains the return code values in numeric order for the SMAPI.

-28	SM_API_RC_MIN_NAMES
-27	SM_RC_ATTACH_MUTEX_SEM_FAILED
-26	SM_RC_CREATE_MUTEX_SEM_FAILED
-25	SM_RC_OPEN_QUEUE_FAILED
-24	SM_RC_SUB_UNSET_ERROR
-23	SM_RC_CLOSE_EVENT_SEM_FAILED
-22	SM_RC_FREE_MEM_ERROR
-21	SM_RC_QUEUE_CLOSE_ERROR
-20	SM_RC_DEALLOCATING_SH_MEM
-19	SM_RC_ASSOC_EVENT_SEM_FAILED
-18	SM_RC_CREATE_EVENT_SEM_FAILED
-17	SM_RC_CREATE_MBOX_FAILED
-16	SM_RC_EALLBUSY
-15	SM_RC_ENOACCEPT
-14	SM_RC_EBADAPPNAME
-13	SM_RC_EALLOC
-12	SM_RC_ENOMEM
-11	SM_RC_EBADHANDLE
-10	SM_RC_ENOHANDLES
-9	SM_RC_EMMSGSIZE
-8	SM_RC_EUNKMSG
-7	SM_RC_ETIMEOUT
-6	SM_RC_EUNEXP
-5	SM_RC_EINVAL
-4	SM_RC_ENOSERVER
-3	SM_RC_ENOCONN
-2	SM_RC_ENOMSG
-1	SM_RC_EAPIVERSION
0	SM_RC_OK
1	SM_RC_NOT_VALID_REQUEST
2	SM_RC_BAD_MODE
3	SM_RC_NOT_WHILE_MIC_ON

4	SM_RC_MIC_ALREADY_ON
5	SM_RC_MIC_ALREADY_OFF
6	SM_RC_MIC_ON_PENDING
7	SM_RC_MIC_OFF_PENDING
8	SM_RC_NOT_WHILE_PLAYING
10	SM_RC_BAD_AUDIO
11	SM_RC_RECORD_OPEN_ERROR
12	SM_RC_PLAY_OPEN_ERROR
13	SM_RC_AUDIO_IN_USE
14	SM_RC_BAD_AUDIO_PROTOCOL
15	SM_RC_AUDIO_TIMEOUT
16	SM_RC_AUDIO_DISCONNECTED
17	SM_RC_AUDIO_OVERRUN
18	SM_RC_AUDIO_FORCED_MIC_OFF
19	SM_RC_NO_MORE_AUDIO_FILES
20	SM_RC_BAD_AP
28	SM_API_RC_MAX_NAMES
30	SM_RC_BAD_DECO
40	SM_RC_BAD_ADDWORD
41	SM_RC_ADDED
42	SM_RC_NOT_ADDED
43	SM_RC_MULTIPLE_SPELLINGS
44	SM_RC_ILLEGAL_SPELLING
45	SM_RC_ILLEGAL_SPOKENLIKE
46	SM_RC_MISMATCHED_ACOUSTICS
47	SM_RC_BAD_ACOUSTICS
48	SM_RC_SPELLING_TOO_LONG
49	SM_RC_ACOUSTICS_TOO_LONG
50	SM_RC_ADDWORD_LIMIT_EXCEEDED
60	SM_RC_SERVER_ERROR
61	SM_RC_SERVER_MALLOC_ERROR
62	SM_RC_SERVER_FILE_OPEN_ERROR
63	SM_RC_SERVER_FILE_WRITE_ERROR
64	SM_RC_SERVER_FILE_READ_ERROR
65	SM_RC_SERVER_FILE_CLOSE_ERROR
66	SM_RC_SERVER_PROCESS_ERROR
67	SM_RC_SERVER_TERMINATED
70	SM_RC_BAD_TAG
71	SM_RC_BAD_UTTNO
72	SM_RC_BAD_MESSAGE
80	SM_RC_NOT_DELETED
81	SM_RC_NOT_INVOCAB
82	SM_RC_INVOCAB
83	SM_RC_BAD_VOCAB
84	SM_RC_MISSING_EXTERN
90	SM_RC_BAD_USERID

91	SM_RC_BAD_ENROLLID
92	SM_RC_BAD_PASSWORD
93	SM_RC_BAD_TASKID
94	SM_RC_BAD_CLIENT
95	SM_RC_USERID_EXISTS
96	SM_RC_ENROLLID_EXISTS
97	SM_RC_USERID_BUSY
98	SM_RC_ENROLLID_BUSY
99	SM_RC_BAD_SCRIPT
100	SM_RC_BAD_DESCRIPTION
101	SM_RC_ENROLLID_RUNNING
102	SM_RC_ENROLLMENT_NOT_COMPLETE
103	SM_RC_MISMATCHED_LANGUAGE
104	SM_RC_MISMATCHED_ALPHABET
105	SM_RC_MISMATCHED_SCRIPT
106	SM_RC_BAD_LANGUAGE
107	SM_RC_BAD_NAME
108	SM_RC_INVALID_WINDOW_HANDLE
110	SM_RC_BAD_ITEM
111	SM_RC_BAD_VALUE
120	SM_RC_BUSY_LAST_UTTERANCE
121	SM_RC_BUSY_WORD_CORRECTION
130	SM_RC_NO_SPACE
131	SM_RC_NO_SPACE_INIT_RECO
132	SM_RC_NO_SPACE_INIT_ENROLL
133	SM_RC_NO_SPACE_TERM_ENROLL
134	SM_RC_NO_SPACE_MIC_ON
140	SM_RC_INVALID_PARM_MAX_LEN
170	SM_RC_NO_FOCUS_APP
171	SM_RC_FOCUS_GRANTED
172	SM_RC_FOCUS_REQUEST_PENDING
173	SM_RC_FOCUS_DENIED
174	SM_RC_NAV_ALREADY_DEFINED
175	SM_RC_NOT_IN_NOTIFY
176	SM_RC_EXISTS_IN_NOTIFY
178	SM_RC_INCOMPATIBLE_ENROLLMENT
200	SM_RC_SM_NOT_OPEN
201	SM_RC_WRONG_SM_VERSION
202	SM_RC_SM_NOT_ACTIVE_CLIENT
203	SM_RC_SM_INVALID_MSG_TYPE
204	SM_RC_REPLY_NULL
205	SM_RC_NO_MORE_CONNECTIONS
206	SM_RC_NO_TOPLEVEL_WIDGET
207	SM_RC_CONNECTION_CHANGED
208	SM_RC_CALLBACK_LIST_CHANGED
209	SM_RC_ASCII_ALREADY_SET

210	SM_RC_NOTHING_TO_DISPATCH
211	SM_RC_MAX_MSG_QUEUES_EXCEEDED
214	SM_RC_OPEN_SYNCH_QUEUE_FAILED
215	SM_RC_ALREADY_CONNECTED
216	SM_RC_ALREADY_OPENED
250	SM_RC_MAX_NAMES
1000	SM_RC_NOT_YET

### A.1.2 SMAPI Status Codes

100	SM_STAT_PLAY_START
101	SM_STAT_PLAY_STOP
102	SM_STAT_BAD_AUDIO
103	SM_STAT_BAD_TAG
104	SM_STAT_BAD_UTTERANCE
105	SM_STAT_BAD_MESSAGE
106	SM_STAT_ENROLLMENT_RECORDING
107	SM_STAT_ENROLLMENT_RUNNING
108	SM_STAT_ENROLLMENT_FAILED
109	SM_STAT_ENROLLMENT_COMPLETE
110	SM_STAT_ENROLLMENT_BUSY

## A.2 SMAPI Message Types

These message values are defined in the SMCOMM.H file.

SM.SET_REPLY	53
SM.QUERY_REPLY	54
SM.QUERY_LANGUAGES_REPLY	55
SM.QUERY_USERS_REPLY	56
SM.QUERY_ENROLLIDS_REPLY	57
SM.QUERY_TASKS_REPLY	58
SM.INIT_RECOGNIZER_REPLY	59
SM.SPARE_PUBLIC_1_REPLY	60
SM.MIC_ON_REPLY	61
SM.MIC_OFF_REPLY	62
SM.REQUEST_MIC_ON_REPLY	63
SM.REQUEST_MIC_OFF_REPLY	64
SM.RECOGNIZE_NEXT_WORD_REPLY	65
SM.PLAY_WORDS_REPLY	66
SM.PLAY_UTTERANCE_REPLY	67
SM.PLAY_MESSAGE_REPLY	68
SM.CANCEL_PLAYBACK_REPLY	69
SM.NEW_CONTEXT_REPLY	70
SM.EVENT_NOTIFY_REPLY	71
SM.QUERY_ALTERNATES_REPLY	72
SM.WORD_CORRECTION_REPLY	73
SM.CORRECT_TEXT_REPLY	74
SM.CORRECT_TEXT_CANCEL_REPLY	75
SM.QUERY_WORD_REPLY	76
SM.ADD_PRONUNCIATION_REPLY	77
SM.REMOVE_PRONUNCIATION_REPLY	78
SM.QUERY_PRONUNCIATIONS_REPLY	79
SM.DEFINE_VOCAB_REPLY	80
SM.ADD_TO_VOCAB_REPLY	81
SM.REMOVE_FROM_VOCAB_REPLY	82
SM.QUERY_ADDED_WORDS_REPLY	83
SM.ENABLE_VOCAB_REPLY	84
SM.DISABLE_VOCAB_REPLY	85
SM.UNDEFINE_VOCAB_REPLY	86
SM.QUERY_VOCABS_REPLY	87
SM.QUERY_ENABLED_VOCABS_REPLY	88
SM.SET_USER_INFO_REPLY	89
SM.QUERY_USER_INFO_REPLY	90
SM.DISCARD_DATA_REPLY	91
SM.HALT_RECOGNIZER_REPLY	92

SM_TERMINATE_RECOGNIZER_REPLY	93
SM_INIT_ENROLLMENT_REPLY	94
SM_TERMINATE_ENROLLMENT_REPLY	95
SM_INIT_DATABASE_REPLY	96
SM_TERMINATE_DATABASE_REPLY	97
SM_QUERY_PRONUNCIATION_REPLY	98
SM_REQUEST_FOCUS_REPLY	99
SM_RELEASE_FOCUS_REPLY	100
SM_QUERY_SESSIONS_REPLY	101
SM_DETACH_SESSIONS_REPLY	102
SM_SET_DEFAULT_REPLY	103
SM_QUERY_DEFAULT_REPLY	104
SM_RECOGNIZED_TEXT	105
SM_RECOGNIZED_WORD	106
SM_UTTERANCE_COMPLETED	107
SM_PLAY_WORDS_STATUS	108
SM_PLAY_UTTERANCE_STATUS	109
SM_PLAY_MESSAGE_STATUS	110
SM_SERVER_STATUS	111
SM_EVENT_SYNCH	112
SM_REPORT_ENGINE_ERROR	113
SM_FOCUS_LOST	114
SM_FOCUS_GRANTED	115
SM_REQUEST_DETACH	116
SM_AUDIO_LEVEL	117
SM_COMMAND_WORD	118
SM_MIC_STATE	119
SM_FOCUS_STATE	120
SM_ENGINE_STATE	121
SM_REQUEST_MIC_ON	122
SM_REQUEST_MIC_OFF	123
SM_CONNECT_REPLY	124
SM_DISCONNECT_REPLY	125
SM_CLIENT_DETACH	126
SM_ADD_USER_REQUEST	127
SM_REQUEST_NEW_ENROLLID	128
SM_QUERY_SCRIPTS	129
SM_REQUEST_SCRIPT_TEXT	130
SM_SELECT_SCRIPT	131
SM_QUERY_UTTERANCES	132
SM_SET_UTTERANCE_NUMBER	133
SM_DISCARD_UTTERANCE	134
SM_ADD_USER_REPLY	135
SM_REQUEST_NEW_ENROLLID_REPLY	136
SM_QUERY_SCRIPTS_REPLY	137
SM_REQUEST_SCRIPT_TEXT_REPLY	138



## Appendix A: SMAPI Return Codes, Messages, and Message Types

SM.SELECT_SCRIPT_REPLY	139
SM.QUERY_UTTERANCES_REPLY	140
SM.SET_UTTERANCE_NUMBER_REPLY	141
SM.DISCARD_UTTERANCE_REPLY	142
SM.DEFINE_GRAMMAR	143
SM.DEFINE_GRAMMAR_REPLY	150
SM.QUERY_SPEECH_DATA	169
SM.QUERY_SPEECH_DATA_REPLY	170
SM.SAVE_SPEECH_DATA	171
SM.SAVE_SPEECH_DATA_REPLY	172
SM.RESTORE_SPEECH_DATA	173
SM.RESTORE_SPEECH_DATA_REPLY	174
SM.RECOGNIZED_PHRASE	175
SM.ADD_ENROLLID	176
SM.ADD_ENROLLID_REPLY	177
SM.SET_BINARY	178
SM.SET_BINARY_REPLY	179
SM.QUERY_BINARY	180
SM.QUERY_BINARY_REPLY	181
SM.QUERY_ACOUSTICIDS	182
SM.QUERY_ACOUSTICIDS_REPLY	183
SM.QUERY_PHRASE_ALTERNATIVES	184
SM.QUERY_PHRASE_ALTERNATIVES_REPLY	185
SM.DISCARD_SPEECH_DATA	186
SM.DISCARD_SPEECH_DATA_REPLY	187
SM.SET_TOPICS	188
SM.SET_TOPICS_REPLY	189
SM.QUERY_TOPICS	190
SM.QUERY_TOPICS_REPLY	191
SM.AUTO_COMPLETE	192
SM.AUTO_COMPLETE_REPLY	193
SM.SET_USER_DEFAULT	194
SM.SET_USER_DEFAULT_REPLY	195
SM.QUERY_USER_DEFAULT	196
SM.QUERY_USER_DEFAULT_REPLY	197
SM.REMOVE_USER	198
SM.REMOVE_USER_REPLY	199
SM.REMOVE_ENROLLID	200
SM.REMOVE_ENROLLID_REPLY	201
SM.EVENT_TIME	202
SM.EVENT_TIME_REPLY	203
SM.SAVE_SPEECH_USER	206
SM.SAVE_SPEECH_USER_REPLY	207
SM.RESTORE_SPEECH_USER	208
SM.RESTORE_SPEECH_USER_REPLY	209
SM.QUERY_SPEECH_USER_SIZE	210

SM\_QUERY\_SPEECH\_USER\_SIZE\_REPLY 211

## A.3 SMAPI Message Explanations

The following are explanations for the SMAPI messages.

### SM\_RC\_ACOUSTICS\_TOO\_LONG

The acoustics specified are too long.

**Explanation:**

The recording made for a specified word is longer than the phonetic representation of the requested word.

**User response:**

Enter a phonetic spelling or record the word again.

### SM\_RC\_ADDED

The specified word has been added to the speech vocabulary.

**Explanation:**

The request to add a word was successfully completed.

**User response:**

None

### SM\_RC\_ADDWORD\_LIMIT\_EXCEEDED

The added-word limit has been reached.

**Explanation:**

Only a specified number of words can be added to a speaker's vocabulary. The speech system has reached your added-word limit.

**User response:**

Review your added words and delete the ones that are either unneeded or infrequently used.

### SM\_RC\_ALREADY\_CONNECTED

Already connected to speech recognition engine.

**Explanation:**

Either SmConnect() or SmOpen() was called after the application had already successfully connected to the engine.

**User response:**

Correct the application programming error.

## SM\_RC\_ALREADY\_OPENED

Already opened for connection to speech recognition engine.

**Explanation:**

A second call to SmOpen() was made after the application had already successfully opened the local API data structure.

**User response:**

Correct the application programming error.

## SM\_RC\_ASSOC\_EVENT\_SEM\_FAILED

An IPC error: Unable to associate semaphore.

**Explanation:**

Unable to associate semaphore.

**User response:**

Submit a problem report.

## SM\_RC\_AUDIO\_FORCED\_MIC\_OFF

The speech system has forced the microphone off.

**Explanation:**

The speech system turned off the microphone unexpectedly. This event can occur if the speech system has an internal error or if the disk is full.

**User response:**

Try the request again. Save the log files of the session and submit a problem report.

## SM\_RC\_AUDIO\_IN\_USE

The speech system is already in use.

**Explanation:**

The speech system is being used by another session.

**User response:**

Correct the application programming error.

## SM\_RC\_AUDIO\_OVERRUN

The speech system has overrun its recording buffers.

**Explanation:**

The speech system was not able to continue sending audio data because the recording buffers are full.

**User response:**

Up to 100 seconds of audio can be stored in an internal audio memory buffer. For details, see "Processing Speech Engine Audio" in the SMAPI Developer's Guide.

## **SM\_RC\_BAD\_ACOUSTICS**

Invalid acoustic data has been found.

**Explanation:**

The recorded data was not found for the word requested to be added.

**User response:**

Ensure that the disk is not full, which would cause files to be lost.

## **SM\_RC\_BAD\_ADDWORD**

The add-word process could not be initiated.

**Explanation:**

The speech recognition engine detected an error with the add word component.

**User response:**

Save the log files of the session and submit a problem report.

## **SM\_RC\_BAD\_AP**

The system detected an error with the acoustic processor.

**Explanation:**

The speech recognition engine detected an error with the acoustic-processor component.

**User response:**

Save the log files of the session and submit a problem report.

## **SM\_RC\_BAD\_AUDIO**

The system detected an error with the audio source.

**Explanation:**

An application will get this return code if there is a problem with the audio source (for example, if there is a problem initializing the audio hardware, or audio driver ).

**User response:**

Try the request again. If the problem persists, save the log files of the session, report the error, and restart.

## SM\_RC\_BAD\_DECO

The system detected an error with the speech recognition engine.

**Explanation:**

The speech recognition engine detected an error with the decoding-processor component.

**User response:**

Save the log files of the session and submit a problem report.

## SM\_RC\_BAD\_DESCRIPTION

An invalid description has been specified.

**Explanation:**

An attempt was made to write an enrollment description containing a non-ASCII character or an enrollment description that exceeded the maximum length.

**User response:**

Enter a description that is equal to or less than the maximum length and use only valid, printable ASCII characters.

## SM\_RC\_BAD\_ENROLLID

An invalid enrollment identifier has been specified.

**Explanation:**

The enrollment ID for the specified speaker is not valid.

**User response:**

Select a valid enrollment ID for the specified user ID.

## SM\_RC\_BAD\_ITEM

An invalid item name has been specified.

**Explanation:**

An attempt was made to write an item name containing a non-ASCII character.

**User response:**

Enter an item name using only valid, printable ASCII characters.

## SM\_RC\_BAD\_LANGUAGE

An invalid language has been specified, or this language is not installed.

**Explanation:**

The requested language is not valid or is not installed.

**User response:**

Specify a valid language for the installed speech recognition engine.

## SM\_RC\_BAD\_MESSAGE

An invalid audio message name has been specified.

**Explanation:**

An invalid audio message name was passed by the application to the SMAPI. This is an application programming error.

**User response:**

Correct the application programming error.

## SM\_RC\_BAD\_MODE

The request is not valid in this speech recognition engine mode.

**Explanation:**

The speech recognition engine was initialized for a mode that does not support the requested function. This is an application programming error.

**User response:**

Correct the application programming error.

## SM\_RC\_BAD\_NAME

An invalid name tag has been specified.

**Explanation:**

An attempt was made used to use an invalid name.

**User response:**

Specify a valid name.

## SM\_RC\_BAD\_SCRIPT

A bad script has been specified, or no scripts are available.

**Explanation:**

An attempt was made to use an invalid script for enrollment.

**User response:**

Specify a valid script.

## SM\_RC\_BAD\_TAG

An invalid word tag value has been specified.

**Explanation:**

An invalid tag value was passed by the application to the SMAPI. This is an application programming error.

**User response:**

Correct the application programming error.

## SM\_RC\_BAD\_TASKID

A match could not be found for the specified domain ID.

**Explanation:**

The domain ID is not valid.

**User response:**

Select a valid domain ID.

## SM\_RC\_BAD\_USERID

A match could not be found for the specified user ID.

**Explanation:**

The user ID is not valid.

**User response:**

Select a valid user ID.

## SM\_RC\_BAD\_UTTNO

An invalid utterance number has been specified.

**Explanation:**

An invalid utterance number was passed by the application to the SMAPI. This is an application programming error.

**User response:**

Correct the application programming error.

## SM\_RC\_BAD\_VALUE

An invalid value has been specified.

**Explanation:**

An attempt was made to write a value containing a non-ASCII character.

**User response:**

Enter a value using only valid, printable ASCII characters.

## SM\_RC\_BAD\_VOCAB

The specified vocabulary is not valid.

**Explanation:**

An invalid vocabulary name was passed by the application to the SMAPI. This is an application programming error.

**User response:**

Correct the application programming error.



## SM\_RC\_BUSY\_LAST\_UTTERANCE

The speech recognition engine is busy processing the last utterance.

**Explanation:**

An attempt was made to play a message or word before all the speech had been processed after an SmMicOff call.

**User response:**

Wait a few seconds and repeat the request until you receive a response.

## SM\_RC\_BUSY\_WORD\_CORRECTION

The speech recognition engine is busy processing the last word correction.

**Explanation:**

An attempt was made to correct a word and turn on the microphone before the previous request was completed.

**User response:**

Wait a few seconds and repeat the request until you receive a response.

## SM\_RC\_CALLBACK\_LIST\_CHANGED

The callback list has been changed during the execution of another callback.

**Explanation:**

The callback list has been modified while executing a callback procedure.

**User response:**

Correct the application programming error.

## SM\_RC\_CONNECTION\_CHANGED

New or closed connection when executing callback procedure.

**Explanation:**

A new connection to the SMAPI was created or an old one was removed from within a callback procedure.

**User response:**

Submit a problem report.

## SM\_RC\_CREATE\_EVENT\_SEM\_FAILED

An IPC error: Unable to create event semaphore.

**Explanation:**

Unable to create event semaphore.

**User response:**

Submit a problem report.

## SM\_RC\_CREATE\_MBOX\_FAILED

An IPC error: Unable to create mbox.

**Explanation:**

Unable to create mbox.

**User response:**

Submit a problem report.

## SM\_RC\_DEEALLOCATING\_SH\_MEM

Unable to deallocate shared memory.

**Explanation:**

There is not sufficient memory for messages to pass between the SMAPI and the speech recognition engine.

**User response:**

Try the request again. If the problem persists, record the error and submit a problem report.

## SM\_RC\_EALLBUSY

IBM ViaVoice is busy.

**Explanation:**

IBM ViaVoice is busy.

**User response:**

Try the request again. If the problem persists, record the error and submit a problem report.

## SM\_RC\_EALLOC

Unable to allocate memory for speech recognition engine messages.

**Explanation:**

There is not sufficient memory for messages to pass between the SMAPI and the speech recognition engine.

**User response:**

Record the error and submit a problem report.

## SM\_RC\_EAPIVERSION

Incorrect API version.

**Explanation:**

The speech recognition engine and the SMAPI are not compatible.

**User response:**

Install a complete and compatible version.

## SM\_RC\_EBADAPPNAME

An invalid application name has been specified.

**Explanation:**

An invalid application name was specified for a session between the SMAPI and the speech recognition engine.

**User response:**

Correct the application programming error.

## SM\_RC\_EBADHANDLE

Invalid handle passed to the speech recognition engine.

**Explanation:**

An invalid internal handle was used by the SMAPI.

**User response:**

Correct the application programming error.

## SM\_RC\_EINVAL

Invalid value passed to speech recognition engine.

**Explanation:**

The value passed from the application to the SMAPI could not be passed to the speech recognition engine.

**User response:**

Correct the application programming error.

## SM\_RC\_ENOCONN

No connection to speech recognition engine.

**Explanation:**

The application is unable to establish a session with the speech recognition engine; most likely, this is because the speech recognition engine is not running.

**User response:**

Restart the speech recognition engine. If the system is running, verify that the level of the client software and the level of the speech recognition engine are compatible.

## SM\_RC\_ENOHANDLES

No handles for speech recognition engine.

**Explanation:**

An internal resource needed by the SMAPI is not available.

**User response:**

Restart the speech recognition engine.

## SM\_RC\_ENOMEM

No memory available for speech recognition engine.

**Explanation:**

No memory is available for the speech recognition engine communications process.

**User response:**

Restart the speech recognition engine.

## SM\_RC\_ENOMSG

No message available from speech recognition engine.

**Explanation:**

A request was made to the speech recognition engine but no response was received. A communications error or a speech recognition engine error prevented the speech recognition engine from responding to the application request.

**User response:**

Report the error and try the request again.

## SM\_RC\_ENOSERVER

Unable to find an available speech recognition engine.

**Explanation:**

All speech recognition engines are being used by other users.

**User response:**

Wait a few minutes and try again to establish a session.

## SM\_RC\_ENROLLID\_BUSY

Enrollment is busy.

**Explanation:**

The enrollment ID is still being processed.

**User response:**

An attempt was made to use an enrollment ID that was locked. Try to use the enrollment ID later.

## SM\_RC\_ENROLLID\_EXISTS

The specified enrollment identifier already exists.

**Explanation:**

An attempt was made to specify an enrollment ID that already exists for the specified user ID.

**User response:**

Provide a unique enrollment ID to the enrollment program.

## SM\_RC\_ENROLLID\_RUNNING

The enrollment data set specified is being processed.

**Explanation:**

A fully recorded enrollment data set is being processed and will be available for dictation when it has been processed.

**User response:**

None.

## SM\_RC\_ENROLLMENT\_NOT\_COMPLETE

Enrollment has not completed for this identifier.

**Explanation:**

An enrollment data set that is still being processed offline was specified for decoding.

**User response:**

Try to use the enrollment data set for dictating after the offline processing is completed.

## SM\_RC\_FREE\_MEM\_ERROR

An IPC error: Error freeing memory.

**Explanation:**

Error freeing memory.

**User response:**

Submit a problem report.

## SM\_RC\_EXISTS\_IN\_NOTIFY

Application is already in specified group

**Explanation:**

An application asks to enable a notification group in which the application is already registered.

**User response:**

None.

## SM\_RC\_FOCUS\_DENIED

A focus request has been denied.

**Explanation:**

The current application with speech focus has "grabbed" the focus and changes are being blocked by the engine.

**User response:**

None.

## SM\_RC\_FOCUS\_GRANTED

The application already has the speech focus.

**Explanation:**

The requesting application has already been granted speech focus and an asynchronous notification message will not be sent by the engine.

**User response:**

None.

## SM\_RC\_INCOMPATIBLE\_ENROLLMENT

Enrollment was not compatible with the current engine.

**Explanation:**

An enrollment created for a previous release may not be supported on the current engine.

**User response:**

Re-enroll.

## SM\_RC\_ILLEGAL\_SPOKENLIKE

An invalid spoken-like spelling has been specified.

**Explanation:**

For a given language, certain ASCII characters are not valid.

**User response:**

Provide a valid spoken-like spelling for the language you are using.

## SM\_RC\_ILLEGAL\_SPELLING

An invalid spelling has been specified.

**Explanation:**

For a given language, certain ASCII characters are not valid.

**User response:**

Provide a valid spelling for the language you are using.

## SM\_RC\_INVALID\_PARM\_MAX\_LEN

The specified parameter exceeds the maximum length.

**Explanation:**

The specified parameter exceeds the maximum length permitted by the speech recognition engine.

**User response:**

Specify the parameter again.

## SM\_RC\_INVOCAB

The specified word is in the speech recognition engine vocabulary.

**Explanation:**

The specified word already exists in the speech recognition engine vocabulary.

**User response:**

None.

## SM\_RC\_MIC\_ALREADY\_OFF

The microphone is already off.

**Explanation:**

A request was made to turn off the microphone, but it is already off.

**User response:**

None.

## SM\_RC\_MIC\_ALREADY\_ON

The microphone is already on.

**Explanation:**

A request was made to turn on the microphone, but it is already on. This is an application programming error.

**User response:**

Correct the application programming error.

## SM\_RC\_MIC\_OFF\_PENDING

A microphone-off request is already in progress.

**Explanation:**

A request was made to turn off the microphone while it was in the process of being turned off. This is an application programming error.

**User response:**

Correct the application programming error.

## SM\_RC\_MIC\_ON\_PENDING

A microphone-on request is already in progress.

**Explanation:**

A request was made to turn on the microphone while it was in the process of being turned on. This is an application programming error.

**User response:**

Correct the application programming error.

## SM\_RC\_MISMATCHED\_ACOUSTICS

An acoustics mismatch has been found.

**Explanation:**

The recording does not match the spelling provided for the added word.

**User response:**

Try recording the word again, or specify a spelling that phonetically matches the requested word.

## SM\_RC\_MISMATCHED\_ALPHABET

The domain and enrollment identifier specified are in different alphabets.

**Explanation:**

The domain specified is not for the same alphabet as the enrollment ID.

**User response:**

Specify a valid domain for the alphabet in which you want to dictate.

## SM\_RC\_MISMATCHED\_LANGUAGE

The domain and enrollment identifier specified are in different languages.

**Explanation:**

The specified domain is not for the same language as the enrollment ID.

**User response:**

Specify a valid domain for the language in which you want to dictate.

## SM\_RC\_MISMATCHED\_SCRIPT

The script specified does not match the previously specified script.

**Explanation:**

A partially recorded enrollment used a script that is different from the one currently specified.

**User response:**

Use the script specified in the last session to continue the enrollment session.



## SM\_RC\_MISSING\_EXTERN

A required external vocabulary does not exist.

**Explanation:**

An external vocabulary referenced in a grammar has not been defined.

**User response:**

None.

## SM\_RC\_MULTIPLE\_SPELLINGS

Multiple spellings specified.

**Explanation:**

More than one spelling was specified for an added-word or more than one tag was specified. This is an application programming error.

**User response:**

Correct the application programming error.

## SM\_RC\_NAV\_ALREADY\_DEFINED

Navigator application already defined.

**Explanation:**

A second application has attempted to assert the navigator field.

**User response:**

None.

## SM\_RC\_NO\_FOCUS\_APP

No application has the speech focus.

**Explanation:**

A request has been made to turn the microphone on, but no application has speech focus.

**User response:**

Request speech focus for an application.

## SM\_RC\_NO\_MORE\_CONNECTIONS

Duplicate SmOpen call.

**Explanation:**

The SmOpen function was called again without previously closing the connection to the SMAPI through SmClose.

**User response:**

This is an application programming error.

## **SM\_RC\_NO\_SPACE**

No space is left on the disk.

**Explanation:**

No space is left on the disk.

**User response:**

None.

## **SM\_RC\_NO\_SPACE\_INIT\_ENROLL**

No space is left on the disk for an enrollment session.

**Explanation:**

Not enough disk space is available to record the next sentence during enrollment.

**User response:**

Restart the speech recognition engine.

## **SM\_RC\_NO\_SPACE\_INIT\_RECO**

No space is left on the disk for a recognition session.

**Explanation:**

Not enough disk space is available to record the next sentence during dictation.

**User response:**

None.

## **SM\_RC\_NO\_SPACE\_MIC\_ON**

No space is left on the disk for PCM.

**Explanation:**

No space is left on the disk for PCM.

**User response:**

None.

## **SM\_RC\_NO\_SPACE\_TERM\_ENROLL**

Not enough disk space to complete training.

**Explanation:**

Not enough disk space available to complete training.

**User response:**

Restart the speech recognition engine.

## SM\_RC\_NOT\_ADDED

The specified word has not been added to the speech recognition engine vocabulary.

**Explanation:**

The add-word request was not successfully completed. For a given language, a request to add a word might fail.

**User response:**

Try a new recording of the word or provide a phonetic spelling.

## SM\_RC\_NOT\_DELETED

The speech recognition engine was unable to delete the specified added word.

**Explanation:**

The speech recognition engine attempted to delete a word that is not in the specified vocabulary.

**User response:**

Correct the application programming error.

## SM\_RC\_NOT\_IN\_NOTIFY

Application is not in the specified group.

**Explanation:**

An application asks to disable a notification group in which the application is not registered.

**User response:**

None.

## SM\_RC\_NOT\_INVOCAB

The specified word is not in the speech recognition engine vocabulary.

**Explanation:**

The specified word does not exist in the speech recognition engine vocabulary.

**User response:**

None.

## SM\_RC\_NOT\_VALID\_REQUEST

The engine is not in the proper state to handle this request.

**Explanation:**

The engine is not in the proper state to handle this request.

**User response:**

Correct the application programming error.

## SM\_RC\_NOT\_WHILE\_MIC\_ON

The given request is not valid while the microphone is on.

**Explanation:**

Some requests are not valid when the microphone is on. This is an application programming error.

**User response:**

Turn off the microphone before making the request.

## SM\_RC\_NOT\_WHILE\_PLAYING

The given request is not valid while playing audio.

**Explanation:**

Some requests are not valid while audio is playing. This is an application programming error.

**User response:**

Wait until the audio has finished playing before making this request.

## SM\_RC\_NOT\_YET

SMAPI function call not supported.

**Explanation:**

The application made a function call that is not supported by the installed SMAPI. The supporting function in the speech recognition engine is not present. All applications must match the installed SMAPI.

**User response:**

If a new application is installed, ensure that it matches the installed SMAPI or install the required SMAPI and upgrade the existing applications.

## SM\_RC\_OK

Successful completion.

**Explanation:**

The most recent application request to the SMAPI was successfully completed.

**User response:**

None.

## SM\_RC\_OPEN\_SYNC\_QUEUE\_FAILED

SmOpen failed to get required resources.

**Explanation:**

A system problem occurred during the SmOpen call when certain system resources are allocated.

**User response:**

Try the request again. If the problem persists, report the error and restart the speech system. Save the log files of the session and submit a problem report on the speech recognition engine.

## SM\_RC\_PLAY\_OPEN\_ERROR

The system was unable to open the play device.

**Explanation:**

The speech recognition engine was not able to open the play device. Another application is using the IBM VoiceType Dictation Adapter or the device driver was replaced by another application.

**User response:**

Try the request again. If the problem persists, report the error and restart the speech system. Save the log files of the session and submit a problem report on the speech recognition engine.

## SM\_RC\_QUEUE\_CLOSE\_ERROR

An IPC error: Error closing queue.

**Explanation:**

Error closing queue.

**User response:**

Try the request again. If the problem persists, submit a problem report.

## SM\_RC\_RECORD\_OPEN\_ERROR

The system was unable to open the recording device.

**Explanation:**

The speech recognition engine was not able to open the recording device. Another application is using the IBM VoiceType Dictation Adapter or the device driver was replaced by another application.

**User response:**

Try the request again. If the problem persists, report the error and restart the speech system. Save the log files of the session and submit a problem report on the speech recognition engine.

## SM\_RC\_REPLY\_NULL

NULL reply parameter.

**Explanation:**

The pointer to the reply structure that was given to one of the reply message functions is NULL.

**User response:**

Correct the application programming error.

## SM\_RC\_SERVER\_ERROR

The speech recognition engine detected an internal error.

**Explanation:**

The speech recognition engine has detected an internal error.

**User response:**

Save the log files of the session. Ensure that sufficient disk space is available for the system. Restart the speech recognition engine.

## SM\_RC\_SERVER\_FILE\_CLOSE\_ERROR

The speech recognition engine has an internal close error.

**Explanation:**

The speech recognition engine was not able to close one of its files. It is possible that, because of the additional applications running on the speech recognition engine, the maximum number of open file handles, the microprocessor limits, or the page space limits have been reached.

**User response:**

Save the log files of the session. Ensure that sufficient disk space is available for the system. Restart the speech recognition engine.

## SM\_RC\_SERVER\_FILE\_OPEN\_ERROR

The speech recognition engine has an internal open error.

**Explanation:**

The speech recognition engine was not able to open one of its files. It is possible that, because of the additional applications running on the speech recognition engine, the maximum number of open file handles, the microprocessor limits, or the page space limits have been reached.

**User response:**

Save the log files of the session. Ensure that sufficient disk space is available for the system. Restart the speech recognition engine.

## SM\_RC\_SERVER\_FILE\_READ\_ERROR

The speech recognition engine has an internal read error.

**Explanation:**

The speech recognition engine was not able to read one of its files. It is possible that, because of the additional applications running on the speech recognition engine, the maximum number of open file handles, the microprocessor limits, or the page space limits have been reached.

**User response:**

Save the log files of the session. Ensure that sufficient disk space is available for the system. Restart the speech recognition engine.

## SM\_RC\_SERVER\_FILE\_WRITE\_ERROR

The speech recognition engine has an internal write error.

**Explanation:**

The speech recognition engine was not able to write one of its files. It is possible that, because of the additional applications running on the speech recognition engine, the maximum number of open file handles, the microprocessor limits, or the page space limits have been reached.

**User response:**

Save the log files of the session. Ensure that sufficient disk space is available for the system. Restart the speech recognition engine.

## SM\_RC\_SERVER\_MALLOC\_ERROR

The speech recognition engine has an internal malloc error.

**Explanation:**

The speech recognition engine was not able to allocate memory for its process.

**User response:**

Save the log files of the session. Ensure that sufficient memory is available for the system. Restart the speech recognition engine.

## SM\_RC\_SERVER\_PROCESS\_ERROR

The speech recognition engine has an internal process error.

**Explanation:**

The speech recognition engine had an unrecoverable error. It is possible that, because of the additional applications running on the speech recognition engine, the maximum number of open file handles, the microprocessor limits, or the page space limits have been reached.

**User response:**

Save the log files of the session. Ensure that sufficient disk space is available for the system. Restart the speech recognition engine.

## SM\_RC\_SERVER\_TERMINATED

Speech recognition engine terminated.

**Explanation:**

The speech recognition engine has stopped.

**User response:**

Save the log files of the session and restart the speech recognition engine.

## SM\_RC\_SM\_EINVAL\_MSG\_TYPE

Invalid message type.

**Explanation:**

This is a program logic error. The program has used an access function against an invalid message type.

**User response:**

Use an access function that is valid for the message type.

## SM\_RC\_SM\_NOT\_ACTIVE\_CLIENT

The speech-aware application is not active.

**Explanation:**

The speech-aware application is not active.

**User response:**

Correct the application programming error.

## SM\_RC\_SM\_NOT\_OPEN

Application has not opened SMAPI using an SmOpen call.

**Explanation:**

The speech application has not been opened by an SmOpen call.

**User response:**

Correct the application programming error.

## SM\_RC\_SUB\_UNSET\_ERROR

An IPC error: Error unsetting memory.

**Explanation:**

The speech application has not been opened by an SmOpen call.

**User response:**

Correct the application programming error.



## SM\_RC\_SPELLING\_TOO\_LONG

The spelling specified is too long.

**Explanation:**

The specified word is too long.

**User response:**

The spelling of the specified word is phonetically longer than the recording made for the word. Enter a phonetic spelling or record the word again.

## SM\_RC\_USERID\_BUSY

User identifier is busy.

**Explanation:**

An attempt was made to use a user ID that was locked by another client workstation or application.

**User response:**

Try to use the user ID later.

## SM\_RC\_USERID\_EXISTS

User identifier already exists.

**Explanation:**

An attempt was made to add a user ID that already exists on the speech recognition engine.

**User response:**

Provide a unique user ID to the program.

## SM\_RC\_WRONG\_SM\_VERSION

Incompatible versions of compiled/dynamically linked SMAPI.

**Explanation:**

The version of the SMAPI with which the application has been compiled is different from that of the installed run-time libraries.

**User response:**

Compile your application with the same version of the API currently installed on your computer.

## SM\_STAT\_BAD\_AUDIO

Bad audio.

**Explanation:**

The connection to the audio source was lost during playback.

**User response:**

Try the request again.

## SM\_STAT\_BAD\_MESSAGE

Bad message.

**Explanation:**

An invalid message was specified for playback. This is an application programming error.

**User response:**

Try again with a valid message.

## SM\_STAT\_BAD\_TAG

Bad tag.

**Explanation:**

An invalid word tag was specified for playback. This is an application programming error.

**User response:**

Try again with a valid word tag.

## SM\_STAT\_BAD\_UTTERANCE

Bad utterance.

**Explanation:**

An invalid utterance was specified for playback. This is an application programming error.

**User response:**

Try again with a valid utterance.

## SM\_STAT\_ENROLLMENT\_BUSY

Enroll ID is busy.

**Explanation:**

This enroll ID is locked through use from a different machine.

**User response:**

Try to use the enroll ID later.

## SM\_STAT\_ENROLLMENT\_COMPLETE

Enrollment complete.

**Explanation:**

Training completed successfully for this enroll ID. This is informational only.

**User response:**

None.

## SM\_STAT\_ENROLLMENT\_FAILED

Enrollment failed.

**Explanation:**

The training process failed for this enroll ID.

**User response:**

Refer to the help for Enrollment.

## SM\_STAT\_ENROLLMENT\_RECORDING

Enrollment recording.

**Explanation:**

The user is recording the enrollment script for this enroll ID. This is informational only.

**User response:**

None.

## SM\_STAT\_ENROLLMENT\_RUNNING

Enrollment running.

**Explanation:**

The engine is running training for this enroll ID. This is informational only.

**User response:**

None.

## SM\_STAT\_PLAY\_START

Audio play started.

**Explanation:**

The message, word, or utterance has started playing. This is informational only.

**User response:**

None.

## **SM\_STAT\_PLAY\_STOP**

Audio play stopped.

**Explanation:**

The message, word, or utterance has stopped playing. This is informational only.

**User response:**

None.

## A.4 SMAPI Logging

Logs used for debugging are disabled by default. Logging is controlled by the value setting of the parameter `api_log_level` in the following file: (ViaVoice)\bin\engine.cfg This file should be used for debugging and diagnostics. The content of the file is not defined as a public interface. You must create a key in the "defaults:" section of the engine.cfg file. The key must appear as follows: `api_log_level = <log level value>` The example below shows how this should appear in engine.cfg file:

```
defaults:
    discard_session_data = true
    api_log_level = 2
```

**Please note:** Be sure to leave a blank line after '`api_log_level = 2`'.

The log level value settings are as follows:

- 0 - Disables logging
- 1 - Message headers
- 2 - Message headers + content
- 3 - Message headers + content + internal engine messages
- >3 - Reserved

After this logging parameter has been changed the engine must be restarted before the new parameter value can take effect. Logs of interest are located in the following IBM ViaVoice host-drive directory: (ViaVoice\Temp) The log file, `router.msg`, includes a trace of IBM ViaVoice SMAPI calls. Differences between SMAPI names in this log file and the documentation include the following:

- "SPCH\_" prefix in the log file and "Sm" prefix in the documentation.
- Underscore("\_") is used in the log file, but not in the documentation.

An explanation of reply message structures that can be logged but will not be seen by the application include the following:

### *SM\_INIT\_DATABASE\_REPLY*

Response to an asynchronous `SmConnect` that is establishing a database session but is presented to the application as a `SM.CONNECT.REPLY` reply message structure.

### *SM\_INIT\_RECOGNIZER\_REPLY*

Response to an asynchronous `SmConnect` that is establishing a recognition session but is presented to the application as a `SM.CONNECT.REPLY` reply message structure.

### *SM\_TERMINATE\_DATABASE\_REPLY*

Response to an asynchronous `SmDisconnect` that is disconnecting from the speech recognition engine during a database session but is presented to the application as a `SM.DISCONNECT.REPLY` reply message structure.

*SM\_TERMINATE\_RECOGNIZER\_REPLY*

Response to an asynchronous SmDisconnect that is disconnecting from the speech recognition engine during a recognition session but is presented to the application as a SM\_DISCONNECT\_REPLY reply message structure.

## Appendix B Speech Recognition Engine Error Messages

These error messages are generated by the speech recognition engine. They can be found in the error.log and engine.log files, which are located in the engine's working directory.

**Bad Data in File '%s'**

Indicates problems with data found in a successfully opened file. Files have been damaged or incompatible speech data is being used. Restore the file indicated from backup or reinstall IBM ViaVoice.

**Bad Value '%s' for Tag '%s' in File '%s'**

Control data is inconsistent. Files have been damaged or incompatible speech data is being used. Restore the file indicated from backup or reinstall the IBM ViaVoice.

**Duplicate Use of Vocabulary Name '%s'**

A duplicate name has been used to define a vocabulary (static or dynamic). This is an application programming error.

**Exceeded Limit of %d %s**

A system limit has been exceeded; for example, the maximum number of defined vocabularies is 100.

**Failed to Open File '%s'**

The specified file failed to open successfully. The file has probably been inadvertently erased. Restore the file indicated from backup or reinstall IBM ViaVoice.

**Missing Tag '%s' in File '%s'**

A required control parameter (tag) was not found in the specified file. Control data is inconsistent. Files have been damaged or incompatible speech data is being used. Restore the file indicated from backup or reinstall IBM ViaVoice.

**Severe Error in %s (%d)**

A variety of severe error conditions that typically terminate the recognition session. File write errors indicate file system problems (for example, the hard disk is full).

**Mailbox** Errors might occur if one of the engine tasks fails and communication breaks down. Therefore, this error is most likely secondary to the primary problem.

**Label Buffer Overrun**

In these cases the session is not ended, only the microphone is forced off. This situation is caused by a very long pause in dictation (approximately 30 seconds) or by an ill-behaved application that turns on the microphone and fails to ask for recognized words.





## Appendix C Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only that IBM product, program, or service may be used.

Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service.

The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armand, NY 10504-1785  
USA

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department T01B, 3039 Cornwallis, Research Triangle Park, NC 27709-2195, USA. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

### C.1 Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM  
ViaVoice  
VoiceType  
Visual Age

Adobe Acrobat is a trademark or registered trademark of Adobe Systems Incorporated.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation in the United States and/or other countries.

## Appendix C: Notices

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Other company, product, and service names may be trademarks or service marks of others.

# Index

## A

access functions, data ..... 5  
 added pronunciations, querying ..... 111  
 added words, querying ..... 90, 92  
 adding new pronunciation ..... 20  
 adding single callback routine ..... 17  
 adding words to vocabulary ..... 23  
 administrative functions ..... 8  
 alphabets, retrieving ..... 199  
 alternative words, requesting a list ..... 94  
 alternative words, retrieving ..... 200  
 annotations, retrieving ..... 201  
 API calls, parallel session ..... 9  
 API starter set command and control ..... 11  
 API starter set dictation ..... 13  
 application closing down and disconnecting engine  
     ..... 13  
 application detaching from speech engine ..... 50  
 application establishing recognition session .... 12,  
     14  
 application processing recognized commands ... 13  
 application receiving reply message structures  
     ..... 289  
 application, callbacks executed by ..... 59  
 application, retrieving name of ..... 203, 204  
 archive size, retrieving ..... 258  
 archived file name, retrieving ..... 256  
 ASCII string, returning ..... 286  
 associating pronunciation with a spelling ..... 20  
 attribute handling functions ..... 5  
 attributes, SMAPI ..... 313  
 audio data, discarding ..... 53, 54  
 audio file, playing prerecorded ..... 78  
 audio functions ..... 9  
 audio message name, retrieving ..... 229  
 audio signal, retrieving level ..... 205  
 available languages, querying ..... 104

## B

beginning text dictation at a new document  
     location ..... 73

## C

callback dispatching ..... 59  
 callback functions ..... 5  
 callback removing routine ..... 143  
 callbacks, speech engine ..... 299  
 cancelling request for playback ..... 30  
 changing text context sent to speech engine .... 73  
 checking specified word in active vocabularies  
     ..... 137  
 checking version of SMAPI ..... 27  
 closing connection to speech engine ..... 57  
 codepage, retrieving ..... 207  
 command processing recognized ..... 13  
 command starter set APIs ..... 11  
 command vocabulary, dynamically change ..... 23  
 communication, closing with speech engine ..... 57  
 compatibility, checking SMAPI and API ..... 27  
 connecting to speech engine ..... 33  
 connection functions ..... 7  
 connection initializing values ..... 77  
 connection, closing SMAPI ..... 32  
 connection, detaching all applications from speech  
     engine ..... 50  
 control starter set APIs ..... 11  
 correcting errors ..... 15  
 correcting misrecognized word ..... 196  
 current speech data, saving ..... 168  
 current vocabularies enabled, querying ..... 100

## D

data access functions ..... 5, 199  
 data type, SM\_ANNOTATION ..... 308  
 data type, SM\_MSG ..... 309  
 data type, SM\_VOCWORD ..... 310  
 data type, SM\_WORD ..... 311  
 data type, SmArg ..... 305  
 data type, SmArgVal ..... 306  
 data type, SmHandler ..... 307  
 data type, VtArg ..... 327  
 data, discarding audio and error-correction .... 53,  
     54  
 data, querying speech ..... 118, 119  
 database functions ..... 7  
 default values querying ..... 98

## Index

default values, setting ..... 185  
defined vocabulary, disabling ..... 51  
defined vocabulary, enabling ..... 60  
defining grammar based vocabulary ..... 40  
deleting dynamic vocabulary ..... 194  
detaching one or more sessions ..... 50  
dictation starter set APIs ..... 13  
direct calls to speech engine ..... 17  
disabling defined vocabulary ..... 51  
discarding audio data ..... 53, 54  
discarding error-correction data ..... 53, 54  
disconnecting engine ..... 13  
disconnecting speech engine ..... 57  
dispatching functions ..... 5  
domains, querying list ..... 123, 125  
dynamic actions, change command vocabularies  
..... 23  
dynamic actions, creating new vocabulary .. 43, 46  
dynamic actions, defining new grammar ..... 40  
dynamic vocabulary, deleting ..... 194

## E

enabling defined vocabulary ..... 60  
enabling recognition of next word ..... 140  
enabling vocabularies ..... 12, 14  
engine, directing to process speech ..... 12, 14  
engine, disconnecting ..... 13  
engine, establishing recognition session .... 12, 14  
engine/SMAPI return codes ..... 329  
enrollment ID descriptions, retrieving ..... 210  
enrollment status, retrieving ..... 262  
enrollment, query user IDs ..... 102  
error messages, speech engine ..... 365  
error-correction, data, discarding ..... 53, 54  
error-correction, playing back spoken words .... 82  
error-correction, utterance playback ..... 80  
errors, correcting speech recognition ..... 15  
establishing connection to speech engine ..... 33  
event ID, retrieving ..... 215

## F

firm words, retrieving ..... 219  
flags, retrieving ..... 220  
focus state, retrieving ..... 222

FSG file, retrieving path ..... 224  
function call descriptions, format ..... 3  
function calls to speech engine ..... 17  
function calls, SAPI ..... 4  
functions, administrative ..... 8  
functions, attribute handling ..... 5  
functions, audio ..... 9  
functions, callback and dispatching ..... 5  
functions, callbacks executed by application .... 59  
functions, connection ..... 7  
functions, data access ..... 5  
functions, database ..... 7  
functions, session ..... 7  
functions, speech engine state ..... 8  
functions, vocabulary ..... 8

## G

grammar compiler function calls ..... 319

## H

halt recognition temporarily ..... 67

## I

improving future recognition ..... 36, 38  
increment values, retrieving ..... 225  
initializing values in a connection ..... 77

## L

languages, querying available ..... 104  
languages, retrieving ..... 227  
list of alternative words, requesting ..... 94

## M

message explanations, SAPI ..... 337  
message type, retrieving ..... 230  
message types, SAPI ..... 333  
message word count, retrieving ..... 237  
message, cancelling request for playback ..... 30  
message, receiving and dispatching callbacks ... 59  
message, receiving from speech engine ..... 139  
microphone state, retrieving ..... 228

microphone, turning off ..... 69, 154  
 microphone, turning on ..... 71, 155  
 misrecognized word, correcting ..... 196

## N

name values, retrieving ..... 231  
 naming conventions ..... 4  
 Nbest feature ..... 105  
 next word to decode, searching ..... 140  
 notification, requesting specific time ..... 64  
 notification, requesting when speech engine  
     completes decoding ..... 62

## O

opening SMAPI ..... 77  
 options field, retrieving ..... 238  
 options flags, retrieving ..... 257

## P

parallel session API calls ..... 9  
 parameters, querying engine ..... 85  
 parameters, querying value of engine ..... 96  
 percent complete, retrieving ..... 239  
 phonetic pronunciations, retrieving ..... 240  
 phrase score, retrieving ..... 241  
 phrase state, retrieving ..... 242  
 playback request, cancelling ..... 30  
 playing back spoken utterance ..... 80  
 playing back spoken words ..... 82  
 playing prerecorded audio file ..... 78  
 predefined vocabularies, adding words to ..... 23  
 preferred topics count, retrieving ..... 243  
 processing recognized commands ..... 13  
 processing recognized text ..... 15  
 processing speech, directing engine ..... 12, 14  
 pronunciation, adding new ..... 20  
 pronunciation, querying added ..... 111, 113  
 pronunciation, querying existence ..... 107, 109  
 pronunciation, removing from personal pool... 148  
 pronunciations, retrieving ..... 244

## Q

querying added words ..... 90, 92  
 querying available languages ..... 104  
 querying currently enabled vocabularies ..... 100  
 querying default userid, enrollid, taskid ..... 98  
 querying engine parameters ..... 85  
 querying enrollment ID info ..... 129  
 querying existence of pronunciation ..... 107  
 querying list of added pronunciations .... 111, 113  
 querying sessions ..... 117  
 querying speech data ..... 118  
 querying speech engine state ..... 8  
 querying user enrollment IDs ..... 102  
 querying user ID info ..... 129  
 querying users in system ..... 131  
 querying vocabularies, defined ..... 135

## R

reason code for focus change, retrieving ..... 221  
 receiving messages from speech engine ..... 139  
 receiving messages, dispatching callbacks ..... 59  
 recognition session, enabling defined vocabulary  
     ..... 60  
 recognition session, establishing ..... 12, 14  
 recognition session, querying ..... 117  
 recognition session, temporary halt ..... 67  
 recognition, improving future ..... 36, 38  
 recognized commands, processing ..... 13  
 recognized text, processing ..... 15  
 releasing speech focus ..... 142  
 removing callback routine ..... 143  
 removing pronunciation from personal pool... 148  
 removing words from dynamic vocabulary .... 146  
 reply message structures ..... 289  
 reply structure functions unsolicited callbacks  
     ..... 303  
 requesting completions of specified string ..... 28  
 requesting microphone turnoff ..... 154  
 requesting microphone turnon ..... 155  
 requesting speech focus ..... 152  
 resetting utterance number ..... 160  
 retrieving alphabets ..... 199  
 retrieving alternative words ..... 200  
 retrieving annotations ..... 201

## Index

retrieving archive size .....	258	retrieving time values .....	270
retrieving archive version .....	259	retrieving topics array .....	271
retrieving archived file name .....	256	retrieving trained values .....	272
retrieving array of increment values .....	225	retrieving user ID .....	273, 274
retrieving audio message name .....	229	retrieving users .....	275
retrieving audio signal .....	205	retrieving utterance number .....	277
retrieving codepage .....	207	retrieving value of item .....	206, 226
retrieving engine state .....	211	retrieving vocabulary list .....	278
retrieving enroll ID .....	213, 214	retrieving vocabulary name .....	279
retrieving enrollment ID descriptions .....	210	retrieving vocabulary path .....	280
retrieving enrollment status .....	262	retrieving vocabulary words .....	281
retrieving event ID .....	215	retrieving words .....	282
retrieving firm words .....	219	return code description, retrieving .....	246
retrieving focus state .....	222	return code name, retrieving .....	247
retrieving languages .....	227	return code, retrieving .....	245
retrieving message type .....	230	return codes, engine/SMAPI .....	329
retrieving message word count .....	237	returning ASCII string .....	286
retrieving microphone state .....	228	returning SM_MSG return code .....	285
retrieving name of application .....	203, 204	returning symbolic name .....	287
retrieving name values .....	231	router.msg logging .....	363
retrieving options field .....	238		
retrieving options flags .....	257	<b>S</b>	
retrieving options for event .....	216	sample rates, retrieving .....	249
retrieving path name of FSG file .....	224	saving current speech data .....	166, 168
retrieving percent complete .....	239	script flags, retrieving .....	250
retrieving phonetic pronunciations .....	240	searching next word to decode .....	140
retrieving phrase score .....	241	service description, retrieving .....	252
retrieving phrase state .....	242	session detaching .....	50
retrieving preferred topics count .....	243	session disabling a defined vocabulary .....	51
retrieving pronunciations .....	244	session functions .....	7
retrieving reason code for focus change .....	221	session ID, retrieving .....	253
retrieving return code .....	245	session querying .....	117
retrieving return code description .....	246	session starting a speech engine connection ....	33
retrieving return code name .....	247	session, recognition, establishing .....	12, 14
retrieving sample rates .....	249	setting default values .....	185
retrieving script flags .....	250	setting files used by engine .....	187
retrieving scripts .....	251	setting up vocabularies .....	12, 14
retrieving service description .....	252	setting user information .....	191
retrieving session ID .....	253	setting value of speech engine parameter ..	175, 183
retrieving severity .....	254	severity, retrieving .....	254
retrieving single flags .....	220	single callback routine, adding .....	17
retrieving size values .....	255	size values, retrieving .....	255
retrieving spelling .....	260, 261	Sm AddToVocab .....	23
retrieving start/end times .....	283	SM_ANNOTATION .....	308
retrieving status .....	263	SM_MSG .....	309
retrieving task attributes .....	268		

## Index

SM_MSG return code, returning .....	285	SmDispatch .....	59
SM_VOCWORD .....	310	SmEnableVocab .....	60
SM_WORD .....	311	SmEventNotify .....	62
SmAddCallback .....	17	SmEventNotifyEx .....	64
SmAddEnrollid .....	18	SmEventTime .....	66
SmAddPronunciation .....	20	SmGetAlphabets .....	199
SmAddUser .....	25	SmGetAlternates .....	200
SMAPI administrative functions .....	8	SmGetAnnotations .....	201
SMAPI attributes .....	313	SmGetApplication .....	203
SMAPI audio functions .....	9	SmGetApplications .....	204
SMAPI callback and dispatching functions .....	5	SmGetAudioLevel .....	205
SMAPI checking version .....	27	SmGetBinaryItemValue .....	206
SMAPI closing connection .....	32	SmGetCodePage .....	207
SMAPI connecting .....	33	SmGetComment .....	208
SMAPI connection functions .....	7	SmGetConfidenceScores .....	209
SMAPI database functions .....	7	SmGetDescriptions .....	210
SMAPI engine return codes .....	329	SmGetEngineState .....	211
SMAPI establishing connection .....	77	SmGetEnrollId .....	213
SMAPI function calls by group .....	4	SmGetEnrollIds .....	214
SMAPI message explanations .....	337	SmGetEventId .....	215
SMAPI message types .....	333	SmGetEventOptions .....	216
SMAPI opening .....	77	SmGetExpectedRecordingSpace .....	217
SMAPI parallel session calls .....	9	SmGetExpectedTrainingSpace .....	218
SMAPI return codes .....	329	SmGetFirmWords .....	219
SMAPI router.msg log .....	363	SmGetFlags .....	220
SMAPI session functions .....	7	SmGetFocusChangeReason .....	221
SMAPI status codes .....	329	SmGetFocusState .....	222
SmApi VersionCheck .....	27	SmGetFreeSpace .....	223
SMAPI vocabulary functions .....	8	SmGetGrammarPath .....	224
SmArg .....	305	SmGetIncrements .....	225
SmArgVal .....	306	SmGetItemValue .....	226
SmAutoComplete .....	28	SmGetLanguages .....	227
SmCancelPlayback .....	30	SmGetMicState .....	228
SmClose .....	32	SmGetMsgName .....	229
SmConnect .....	33	SmGetMsgType .....	230
SmCorrectText .....	36	SmGetNameValue .....	231
SmCorrectTextEx .....	38	SmGetNextAlternate .....	232
SmDefineGrammar .....	40	SmGetNumberProcessed .....	233
SmDefineVocab .....	43	SmGetNumberRecorded .....	234
SmDefineVocabEx .....	46	SmGetNumberRequired .....	235
SmDetachSessions .....	50	SmGetNumberUtterances .....	236
SmDisableVocab .....	51	SmGetNumberWordMsgs .....	237
SmDiscardData .....	53	SmGetOptions .....	238
SmDiscardSpeechData .....	54	SmGetPercentages .....	239
SmDiscardUtterance .....	56	SmGetPhoneticPronunciations .....	240
SmDisconnect .....	57	SmGetPhraseScore .....	241

## Index

SmGetPhraseState .....	242	SmNewContextEx .....	75
SmGetPreferred .....	243	SmOpen .....	77
SmGetPronunciations .....	244	SmPlayMessage .....	78
SmGetRc .....	245	SmPlayUtterance .....	80
SmGetRcDescription .....	246	SmPlayWords .....	82
SmGetRcName .....	247	SmQuery .....	85
SmGetRequiredTrainingSpace .....	248	SmQueryAddedWords .....	90
SmGetSampleRates .....	249	SmQueryAddedWordsEx .....	92
SmGetScriptFlags .....	250	SmQueryAlternates .....	94
SmGetScripts .....	251	SmQueryBinary .....	96
SmGetService .....	252	SmQueryDefault .....	98
SmGetSessionId .....	253	SmQueryEnabledVocabs .....	100
SmGetSeverity .....	254	SmQueryEnrollIds .....	102
SmGetSizes .....	255	SmQueryLanguages .....	104
SmGetSpeechDataArchive .....	256	SmQueryPhraseAlternatives .....	105
SmGetSpeechDataOptions .....	257	SmQueryPronunciation .....	107
SmGetSpeechDataSize .....	258	SmQueryPronunciationEx .....	109
SmGetSpeechDataVersion .....	259	SmQueryPronunciations .....	111
SmGetSpelling .....	260	SmQueryPronunciationsEx .....	113
SmGetSpellings .....	261	SmQueryScripts .....	115
SmGetStates .....	262	SmQuerySessions .....	117
SmGetStatus .....	263	SmQuerySpeechData .....	118
SmGetTagOffset .....	265	SmQuerySpeechDataEx .....	119
SmGetTags .....	266	SmQuerySpeechUserSize .....	121
SmGetTask .....	267	SmQueryTasks .....	123
SmGetTaskFlags .....	268	SmQueryTopics .....	125
SmGetTasks .....	269	SmQueryUserDefault .....	127
SmGetTimes .....	270	SmQueryUserInfo .....	129
SmGetTopics .....	271	SmQueryUsers .....	131
SmGetTrained .....	272	SmQueryUtterances .....	133
SmGetUserId .....	273	SmQueryVocabs .....	135
SmGetUserIds .....	274	SmQueryWord .....	137
SmGetUsers .....	275	SmReceiveMsg .....	139
SmGetUtteranceList .....	276	SmRecognizeNextWord .....	140
SmGetUtteranceNumber .....	277	SmReleaseFocus .....	142
SmGetVocabList .....	278	SmRemoveCallback .....	143
SmGetVocabName .....	279	SmRemoveEnrollid .....	144
SmGetVocabPath .....	280	SmRemoveFromVocab .....	146
SmGetVocWords .....	281	SmRemovePronunciation .....	148
SmGetWords .....	282	SmRemoveUser .....	150
SmGetWordTimes .....	283	SmRequestFocus .....	152
SmHaltRecognizer .....	67	SmRequestMicOff .....	154
SmHandler .....	307	SmRequestMicOn .....	155
SmMicOff .....	69	SmRequestNewEnrollid .....	156
SmMicOn .....	71	SmRequestScriptText .....	158
SmNewContext .....	73	SmRestoreSpeechData .....	160



SmRestoreSpeechDataEx ..... 162  
 SmRestoreSpeechUser ..... 164  
 SmReturnRc ..... 285  
 SmReturnRcDescription ..... 286  
 SmReturnRcName ..... 287  
 SmSaveSpeechData ..... 166  
 SmSaveSpeechDataEx ..... 168  
 SmSaveSpeechUser ..... 171  
 SmSelectScript ..... 173  
 SmSet ..... 175  
 SmSetArg ..... 182  
 SmSetBinary ..... 183  
 SmSetDefault ..... 185  
 SmSetDirectory ..... 187  
 SmSetUserDefault ..... 189  
 SmSetUserInfo ..... 191  
 SmSetUtteranceNumber ..... 193  
 SmUndefineVocab ..... 194  
 SmWordCorrection ..... 196  
 specified word, checking active vocabularies... 137  
 speech attributes, querying default ..... 98  
 speech data, querying ..... 118, 119  
 speech data, saving current ..... 166, 168  
 speech engine callbacks ..... 299  
 speech focus, releasing ..... 142  
 speech focus, requesting ..... 152  
 speech recognition errors, correcting ..... 15  
 spelling, associating pronunciation with ..... 20  
 spelling, retrieving ..... 260, 261  
 spoken words, playing back ..... 82  
 start/end times, retrieving ..... 283  
 starter set APIs command and control ..... 11  
 starter set APIs dictation ..... 13  
 status codes, SMAPI ..... 329  
 status, retrieving ..... 263  
 symbolic name, returning ..... 287

## T

task attributes, retrieving ..... 268  
 temporary halt of recognition session ..... 67  
 terminating SMAPI connection ..... 32  
 text, processing recognized ..... 15  
 time values, retrieving ..... 270  
 topics array, retrieving ..... 271  
 trained values, retrieving ..... 272

turning off microphone ..... 69  
 turning on microphone ..... 71

## U

unsolicited callbacks, reply structure functions ..... 303  
 updating user's voice model ..... 36, 38  
 user ID, retrieving ..... 273, 274  
 user information, setting ..... 191  
 user, querying enrollment IDs ..... 102  
 users, retrieving ..... 275  
 utterance number, retrieving ..... 277  
 utterance, cancelling request for playback ..... 30  
 utterance, playing back ..... 80  
 utterance, resetting number ..... 160

## V

value of item, retrieving ..... 226  
 version, archive, retrieving ..... 259  
 version, checking SMAPI and API ..... 27  
 vocabularies, checking for specified word ..... 137  
 vocabularies, querying added words ..... 90, 92  
 vocabularies, querying currently enabled ..... 100  
 vocabularies, setting up and enabling ..... 12, 14  
 vocabulary functions ..... 8  
 vocabulary list, retrieving ..... 278  
 vocabulary name, retrieving ..... 279  
 vocabulary path, retrieving ..... 280  
 vocabulary words, retrieving ..... 281  
 vocabulary, adding words to ..... 23  
 vocabulary, deleting dynamic ..... 194  
 vocabulary, disabling defined ..... 51  
 vocabulary, dynamically creating new ..... 43, 46  
 vocabulary, enabling defined ..... 60  
 vocabulary, removing words from ..... 146  
 voice model, updating user's ..... 36, 38  
 VtAddArg ..... 319  
 VtCompileGrammar ..... 320  
 VtGetMessage ..... 321  
 VtGetTranslation ..... 322  
 VtLoadFSG ..... 323  
 VtSetArg ..... 324  
 VtUnloadFSG ..... 325

## Index

### W

- words, alternative, requesting list of ..... 94
- words, cancelling request for playback ..... 30
- words, retrieving ..... 282