INTEL® PERCEPTUAL COMPUTING SDK

Reference Manual

Voice Modules

API Version 1.0

LEGAL DISCLAIMER

THIS DOCUMENT CONTAINS INFORMATION ON PRODUCTS IN THE DESIGN PHASE OF DEVELOPMENT.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

INTEL MAY MAKE CHANGES TO SPECIFICATIONS AND PRODUCT DESCRIPTIONS AT ANY TIME, WITHOUT NOTICE. DESIGNERS MUST NOT RELY ON THE ABSENCE OR CHARACTERISTICS OF ANY FEATURES OR INSTRUCTIONS MARKED "RESERVED" OR "UNDEFINED." INTEL RESERVES THESE FOR FUTURE DEFINITION AND SHALL HAVE NO RESPONSIBILITY WHATSOEVER FOR CONFLICTS OR INCOMPATIBILITIES ARISING FROM FUTURE CHANGES TO THEM. THE INFORMATION HERE IS SUBJECT TO CHANGE WITHOUT NOTICE. DO NOT FINALIZE A DESIGN WITH THIS INFORMATION.

THE PRODUCTS DESCRIBED IN THIS DOCUMENT MAY CONTAIN DESIGN DEFECTS OR ERRORS KNOWN AS ERRATA WHICH MAY CAUSE THE PRODUCT TO DEVIATE FROM PUBLISHED SPECIFICATIONS. CURRENT CHARACTERIZED ERRATA ARE AVAILABLE ON REQUEST.

CONTACT YOUR LOCAL INTEL SALES OFFICE OR YOUR DISTRIBUTOR TO OBTAIN THE LATEST SPECIFICATIONS AND BEFORE PLACING YOUR PRODUCT ORDER.

COPIES OF DOCUMENTS WHICH HAVE AN ORDER NUMBER AND ARE REFERENCED IN THIS DOCUMENT, OR OTHER INTEL LITERATURE, MAY BE OBTAINED BY CALLING 1-800-548-4725, OR BY VISITING INTEL'S WEB SITE HTTP://WWW.INTEL.COM.

ANY SOFTWARE SOURCE CODE REPRINTED IN THIS DOCUMENT IS FURNISHED UNDER A SOFTWARE LICENSE AND MAY ONLY BE USED OR COPIED IN ACCORDANCE WITH THE TERMS OF THAT LICENSE ANY SOFTWARE SOURCE CODE REPRINTED IN THIS DOCUMENT IS FURNISHED UNDER A SOFTWARE LICENSE AND MAY ONLY BE USED OR COPIED IN ACCORDANCE WITH THE TERMS OF THAT LICENSE

INTEL, THE INTEL LOGO, INTEL CORE, INTEL MEDIA SOFTWARE DEVELOPMENT KIT (INTEL MEDIA SDK) ARE TRADEMARKS OR REGISTERED TRADEMARKS OF INTEL CORPORATION OR ITS SUBSIDIARIES IN THE UNITED STATES AND OTHER COUNTRIES.

MPEG IS AN INTERNATIONAL STANDARD FOR VIDEO COMPRESSION/DECOMPRESSION PROMOTED BY ISO. IMPLEMENTATIONS OF MPEG CODECS, OR MPEG ENABLED PLATFORMS MAY REQUIRE LICENSES FROM VARIOUS ENTITIES, INCLUDING INTEL CORPORATION.

*OTHER NAMES AND BRANDS MAY BE CLAIMED AS THE PROPERTY OF OTHERS.

COPYRIGHT © 2011-2012, INTEL CORPORATION. ALL RIGHTS RESERVED.

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

Table of Contents

Intel® Perceptual Computing SDK 1

Document Conventions 1

Acronyms and Abbreviations 1

Programming Guide 2

Voice Recognition Procedure 2

Voice Synthesis 5

Interface and Function Reference 7

PXCVoiceRecognition 7

AddGrammar 7

CreateGrammar 8

DeleteGrammar 9

ProcessAudioAsync 9

QueryProfile 10

SetGrammar 10

SetProfile 11

SubscribeAlert 11

SubscribeRecognition 12

PXCRecognition::Alert::Handler 13

OnAlert 13

PXCRecognition::Recognition::Handler 13

OnRecognized 14

PXCVoiceSynthesis 14

ProcessAudioAsync 15

QueryProfile 15
QueueSentence 16
SetProfile 17
Structure Reference 18
PXCVoiceRecognition::Alert 18
PXCVoiceRecognition::Recognition 18
PXCVoiceRecognition::ProfileInfo 19
PXCVoiceSynthesis::ProfileInfo 20
Enumerator Reference 21
PXCVoiceRecognition::Alert::Label 21
PXCVoiceRecognition::ProfileInfo::Language 21
PXCVoiceSynthesis::ProfileInfo::Language 21
PXCVoiceSynthesis::ProfileInfo::Voice 22

Intel® Perceptual Computing SDK

The Intel® Perceptual Computing SDK is a library of pattern detection and recognition algorithm implementations exposed through standardized interfaces. The library's purpose is to lower barriers to using these algorithms and shift the application developers' focus from coding the algorithm details to innovating on the usage of these algorithms for next generation human computer experience.

This document describes the voice modules of the Intel® Perceptual Computing SDK Application Programming Interface (API). The other Perceptual Computing SDK Manuals that are released with the SDK describe different perceptual computing algorithms and their API definitions.

Document Conventions

The SDK API uses the Verdana typeface for normal prose. With the exception of section headings and the table of contents, all code-related items appear in the `Courier New` typeface (`pxcStatus`). Hyperlinks appear in underlined boldface, such as **`PXCSession`**.

Acronyms and Abbreviations

API	Application Programming Interface

Programming Guide

The SDK voice modules provide voice command and control, dictation, and text to speech capabilities. Voice recognition translates input speech to text. In the command and control mode, the application predefines a list of context words as the command list and the SDK module recognizes speech based only on the command list. In the dictation mode, the SDK module returns the most likely dictated sentence.

Voice Recognition Procedure

The application can use the **UtilPipeline** class to access the voice recognition capability as follows (Example 1). See the **UtilPipeline** definition for additional details.

- The application needs to call the **EnableVoiceRecognition** function to enable voice recognition. This is usually done in the constructor of the **UtilPipeline** derived class.
- Optionally, the application can override the **OnVoiceRecognitionSetup** function to fine-tune any parameters during the module initialization.
- For voice command and control, the application needs to call the **SetVoiceCommands** function to enter the voice command list. This must be done before the **Init** or **LoopFrames** function.
- The application needs to override the **OnRecognized** function to receive recognition events. Similarly, the application can override the **OnAlert** function to receive alert notification.
- At each frame, the application can use the **QueryVoiceRecognition** function to obtain the **PXCVoiceRecognition** instance and call into its member functions.

```
class MyVoicePipeline: public UtilPipeline {
public:
      MyVoicePipeline(PXCSession *session):UtilPipeline(session) {
            EnableVoiceRecognition();
      }
      virtual void PXCAPI OnRecognized(
                              PXCVoiceRecognition::Recognition *data) {
            …
      }
};
```

Example 1: Create A Voice Recognition Module Instance

Alternatively, the application can use the following procedure to create an instance of the voice recognition module and deliver data to it:

1. **Locate the Module Implementation:** The application uses the **PXCSession::CreateImpl** function to create an instance of the voice recognition module interface **PXCVoiceRecognition**, as illustrated in Example 2.

```
PXCVoiceRecognition *rec=0;
session→CreateImpl(PXCVoiceRecognition::CUID,(void**)&rec);
```

Example 2: Create A Voice Recognition Module Instance

2. **Initialize the Module**: The two functions for module initialization are **QueryProfile** and **SetProfile**. The former function returns available configurations. The latter sets a configuration to be the current active one. In Example 3, the application queries the first supported configuration; uses it to locate an input device that can provide data; and then initializes the module with the configuration. Note that the module input needs are specified as part of the **ProfileInfo** structure. The utility class **UtilCapture** will locate an input device that matches the module data needs.

```
PXCVoiceRecognition::ProfileInfo pinfo;
rec→QueryProfile(0,&pinfo);
UtilCapture capture(&session);
capture.LocateStreams(&pinfo.inputs);
rec→SetProfile(&pinfo);
```

Example 3: Initialize A Voice Recognition Module

3. **Initialize Grammar**: For dictation, the application calls the **SetGrammar** function with **gid==0** to set the voice recognition module to the dictation mode. For command and control, the application creates one or multiple grammar contexts as follows:
   - The **CreateGrammar** function creates an empty grammar and returns the grammar identifier.
   - The application can call the **AddGrammar** function multiple times to add words to the grammar.

   The application uses the **SetGrammar** function to activate a grammar context. The application can activate any compiled grammar multiple times. In Example 4, the application configures the word list to be "One," "Two," and "Three."

```
pxcUID gid;
rec→CreateGrammar(&gid);
rec→AddGrammar(gid,1,L"One");
rec→AddGrammar(gid,2,L"Two");
rec→AddGrammar(gid,3,L"Three");
rec→SetGrammar(gid);
```

Example 4: Configure the Command & Control List

4. **Event Handling**: The application implements the **Handler** handler and subscribes to the recognition event by using the **SubcribeRecognition** function as illustrated in Example 5:

```
Class MyRec: public PXCVoiceRecognition::Recognition::Handler {
public:
      virtual void PXCAPI OnRecognized(
                        PXCVoiceRecognition::Recognition *data) {
            // Process Recognition Data
      }
};
Rec→SubscribeRecognition(100,new MyRec);
```

Example 5: Voice Recognition Event Handler

The SDK recognition module will invoke the handler function upon a successful recognition event. As speech recognition takes time (typically 1-2 seconds for a word/phase or 10-15 seconds for a sentence depending on how fast one speaks), the event notification is asynchronous to speech sample input processing. The application must maintain the lifecycle of the handler function until after the application deletes the SDK module instance.

Optionally, the application may implement the alert event handler **Alert::Handler** and subscribe to the alert event notification for any alert messages.

5. **Data Processing Loop**: In the loop, the application passes on samples from the input device to the voice recognition module. The application calls the **ProcessAudioAsync** function to deliver the samples, as illustrated in Example 6.

The application can call the **ProcessAudioAsync** function with a **NULL** pointer to pause audio processing. To resume, simply restart delivering the samples.

```
PXCSmartPtr<PXCAudio> sample;
PXCSmartSPArray sps(2);
for (;;) {
      // Get samples from input device and pass to the module
      capture.ReadStreamAsync(sample.ReleaseRef(),sps.ReleaseRef(0));
      rec→ProcessAudioAsync(sample,sps.ReleaseRef(1));
      sps.SynchronizeEx();
}
```

Example 6: Voice Recognition Module Data Processing Loop

Voice Synthesis

The application uses the following procedure to create an instance of the voice synthesis module and deliver data to it:

6. **Locate the Module Implementation:** The application uses the **PXCSession::CreateImpl** function to create an instance of the voice synthesis module interface **PXCVoiceSynthesis**, as illustrated in Example 7.

```
PXCVoiceSynthesis *tts=0;
session→CreateImpl(PXCVoiceSynthesis::CUID,(void**)&tts);
```

Example 7: Create A Voice Synthesis Module Instance

7. **Initialize the Module**: The two functions for module initialization are **QueryProfile** and **SetProfile**. The former function returns available configurations. The latter sets a configuration to be the current active one.

- **Data Processing Loop**: The application uses the **QueueSentence** function to queue sentences to be synthesized, and then calls the **ProcessAudioAsync** function to retrieve the audio buffer that contains the synthesized speech, and passes the buffer content to any audio output device, as illustrated in Example 8.
  The application may queue and synthesize multiple sentences. The **QueueSentence** function returns a queued sentence identifier that the application can use in the **ProcessAudioAsync** function. The SDK module synthesizes each sentence independently, and in parallel if the implementation allows.

```
pxcUID sid;
tts→QueueSentence(L”Speak this”,10,&sid);
PXCSmartPtr<PXCAudio> sample;
PXCSmartSP sp;
for (;;) {
      pxcStatus sts=tts→ProcessAudioAsync(sid,sample.ReleaseRef(),
                                                  sp.ReleaseRef());
      if (sts<PXC_STATUS_NO_ERROR) break;
      sp.SynchronizeEx();
      // send sample to audio output device
}
```

Example 8: Voice Synthesis Module Data Processing Loop

Interface and Function Reference

This section describes SDK functions and their operations.

In each function description, only commonly used status codes are documented. The function may return additional status codes in certain case. See the **pxcStatus** enumerator for a list of all status codes.

PXCVoiceRecognition

The **PXCVoiceRecognition** interface provides member functions to perform voice command and control. The application can create an instance of this interface by using the **CreateImpl** function with the interface identifier **PXCVoiceRecognition::CUID**.

The **PXCVoiceRecognition** interface exposes the following member functions:

Member Functions	Description
QueryProfile	Return the algorithm configuration information.
SetProfile	Set the algorithm configuration information.
CreateGrammar	Create a new grammar context.
AddGrammar	Add a voice command to the grammar context.
SetGrammar	Activate the grammar context.
DeleteGrammar	Delete the grammar context.
ProcessAudioAsync	Process the audio stream input for voice recognition.
SubscribeRecognition	Subscribe to the recognition event.
SubscribeAlert	Subscribe to the alert event.

AddGrammar

Syntax

```
pxcStatus AddGrammar(pxcUID gid, pxcEnum label, pxcCHAR *command);
```

Parameters

`gid`	The grammar identifier.
`label`	The command label.
`command`	The command string.

Description

This function adds a new or updates an existing voice command of the specified grammar context. A voice command is a **NULL**-terminated text string of words.

Command examples:

- A game instruction: jump up
- A volume control command: computer volume up please

Return Status

`PXC_STATUS_NO_ERROR`	The function completed successfully.
`PXC_STATUS_PARAM_UNSUPPORTED`	Unable to add the command for unsupported annotations or duplicated commands.

Change History

This function is introduced in SDK API 1.0.

CreateGrammar

Syntax

```
pxcStatus CreateGrammar(pxcUID *gid);
```

Parameters

`gid`	The grammar identifier, to be returned.

Description

This function creates a new grammar context, and returns the identifier.

Return Status

`PXC_STATUS_NO_ERROR`	The function completed successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	The SDK module failed to create more contexts.

DeleteGrammar

Syntax

```
pxcStatus DeleteGrammar(pxcUID gid);
```

Parameters

gid	The grammar identifier.

Description

This function deletes the specified grammar context.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.

Change History

This function is introduced in SDK API 1.0.

ProcessAudioAsync

Syntax

```
pxcStatus ProcessAudioAsync(PXCAudio *audio, PXCScheduler::SyncPoint
**sp);
```

Parameters

audio	The audio data buffer.
sp	The SP to be returned.

Description

This function processes the input audio streams for voice recognition.

The application can temporarily pause voice recognition by passing the **NULL** audio data buffer pointer to this function. The SDK module signals the SP when processing of any previous audio data is completed.

To resume recognition, the application can resume passing valid audio data buffers.

This is an asynchronous function.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.

Change History

This function is introduced in SDK API 1.0.

QueryProfile

Syntax

```
pxcStatus QueryProfile(pxcU32 pidx, ProfileInfo *pinfo);
pxcStatus QueryProfile(ProfileInfo *pinfo);
```

Parameters

`pidx`	The zero-based configuration index. Use **`WORKING_PROFILE`**, the default value, to access the current working parameters.
`pinfo`	The **`ProfileInfo`** structure to retrieve the configuration parameters.

Description

This function returns the algorithm configuration parameters.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	The specified configuration was not found.

Change History

This function is introduced in SDK API 1.0.

SetGrammar

Syntax

```
pxcStatus SetGrammar(pxcEnum gid);
```

Parameters

`gid`	The grammar identifier. Use **`gid==0`** to set the SDK module to the dictation mode.

Description

This function sets the specified grammar context as the active grammar in the SDK recognition module.

Return Status

`PXC_STATUS_NO_ERROR`	The function completed successfully.

SetProfile

Syntax

```
pxcStatus SetProfile(ProfileInfo *pinfo);
```

Parameters

`pinfo`	The configuration parameters in the **`ProfileInfo`** structure.

Description

This function sets the algorithm configuration parameters. The values in the **`ProfileInfo`** structure do not have to exactly match what the **`QueryProfile`** function returns.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.
`PXC_STATUS_PARAM_UNSUPPORTED`	There were unsupported parameters in the configuration.

Change History

This function is introduced in SDK API 1.0.

SubscribeAlert

Syntax

```
pxcStatus SubscribeAlert(Alert::Handler *handler);
```

Parameters

`handler`	The instance of the alert event handler **`Alert::Handler`**, or NULL to unsubscribe.

Description

This function subscribes to the alert event notifications.

The application can unsubscribe to the event notification by calling this function with zero as the event handler.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.

Change History

This function is introduced in SDK API 1.0.

SubscribeRecognition

Syntax

```
pxcStatus SubscribeRecognition(pxcU32 threshold, Recognition::Handler
*handler);
```

Parameters

`threshold`	The recognition confidence threshold, ranging from 0 to 100, that the algorithm should trigger a recognition notification.
`handler`	The instance of the recognition event handler **`Recognition::Handler`**, or NULL to unsubscribe.

Description

This function subscribes to the recognition event notifications.

The application can unsubscribe to the event notification by calling this function with zero as the event handler.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.

Change History

This function is introduced in SDK API 1.0.

PXCRecognition::Alert::Handler

The **Handler** interface implements the alert event handling. The application must implement this interface.

The **Handler** interface exposes the following member functions:

Member Functions	Description
OnAlert	This SDK invokes this function when the SDK issues an alert message.

OnAlert

Syntax

```
void OnAlert(Alert *data);
```

Parameters

data	The alert message details. See the **Alert** structure for definition.

Description

The SDK module calls back this function when an alert message is available.

The application should not perform any lengthy operation that blocks execution.

Change History

This function is introduced in SDK API 1.0.

PXCRecognition::Recognition::Handler

The **Handler** interface implements the recognition event handling. The application must implement this interface.

The **Handler** interface exposes the following member functions:

Member Functions	Description
OnRecognized	This SDK invokes this function when the SDK completes voice recognition.

OnRecognized

Syntax

```
void OnRecognized(Recognition *data);
```

Parameters

data	The recognition result. See the Recognition structure for definition.

Description

The SDK module calls back this function to notify the application the recognition result.

The application should not perform any lengthy operation that blocks execution.

Change History

This function is introduced in SDK API 1.0.

PXCVoiceSynthesis

The **PXCVoiceSynthesis** interface provides member functions to produce audio from text (text to speech). The application can create an instance of this interface by using the CreateImpl function with the interface identifier **PXCVoiceSynthesis::CUID**.

The **PXCVoiceSynthesis** interface exposes the following member functions:

Member Functions	Description
QueryProfile	Return the algorithm configuration information.
SetProfile	Set the algorithm configuration information.
QueueSentence	Queue sentence for speech synthesis.

ProcessAudioAsync	Start speech synthesis and receive the synthesized speech.

ProcessAudioAsync

Syntax

```
pxcStatus ProcessAudioAsync(pxcU32 id, PXCAudio **audio,
PXCScheduler::SyncPoint **sp);
```

Parameters

id	The identifier of the queued sentence.
audio	The audio data buffer, to be returned.
sp	The SP to be returned.

Description

This function synthesizes speech from the queued sentence. The synthesized speech is returned in the output audio buffer.

This is an asynchronous function.

Return Status

PXC_STATUS_NO_ERROR	The function returned successfully.
PXC_STATUS_ITEM_UNAVAILABLE	Reaching end of synthesis; there is no more audio content.

Change History

This function is introduced in SDK API 1.0.

QueryProfile

Syntax

```
pxcStatus QueryProfile(pxcU32 pidx, ProfileInfo *pinfo);
pxcStatus QueryProfile(ProfileInfo *pinfo);
```

Parameters

pidx	The zero-based configuration index. Use

	WORKING_PROFILE, the default value, to access the current working parameters.
pinfo	The **ProfileInfo** structure to retrieve the configuration parameters.

Description

This function returns the algorithm configuration parameters.

Return Status

PXC_STATUS_NO_ERROR	The function returned successfully.
PXC_STATUS_ITEM_UNAVAILABLE	The specified configuration was not found.

Change History

This function is introduced in SDK API 1.0.

QueueSentence

Syntax

```
pxcStatus QueueSentence(pxcCHAR *sentence, pxcU32 nchar, pxcU32 *uid);
```

Parameters

sentence	The string of the sentence to be synthesized.
nchar	The number of characteristics in the sentence.
uid	The identifier of the queued sentence, to be returned.

Description

This function queues the sentence to be synthesized. The function returns the identifier of the queued sentence.

Return Status

PXC_STATUS_NO_ERROR	The function returned successfully.

Change History

This function is introduced in SDK API 1.0.

SetProfile

Syntax

```
pxcStatus SetProfile(ProfileInfo *pinfo);
```

Parameters

`pinfo`	The configuration parameters in the **ProfileInfo** structure.

Description

This function sets the algorithm configuration parameters. The values in the **ProfileInfo** structure do not have to exactly match what the **QueryProfile** function returns.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.
`PXC_STATUS_PARAM_UNSUPPORTED`	There were unsupported parameters in the configuration.

Change History

This function is introduced in SDK API 1.0.

Structure Reference

In the following structure references, all reserved fields must be zero.

PXCVoiceRecognition::Alert

Definition

```
struct Alert {
      pxcU64            timeStamp;
      Alert::Label      label;
      pxcU32            reserved[6];
};
```

Description

The `Alert` structure describes the details of an alert message.

Members

`timestamp`	The recognition time stamp, in 100 ns.
`label`	The label of the recognized command. See the **`Alert::Label`** enumerator for definitions.

Change History

This structure is introduced in SDK API 1.0.

PXCVoiceRecognition::Recognition

Definition

```
struct Recognition {
      pxcU64      timeStamp;
      pxcEnum     label;
      pxcU32      confidence;
      pxcU32      reserved[12];
```

```
        pxcCHAR     dictation[1024];
};
```

Description

The `Recognition` structure describes the details of a voice command or a dictation sentence.

Members

`timestamp`	The recognition time stamp, in 100 ns.
`label`	The label of the recognized command.
`confidence`	The recognition confidence level, from 0 to 100.
`dictation`	The dictation sentence with words separated by space and terminated with a NULL character.

Change History

This structure is introduced in SDK API 1.0.

PXCVoiceRecognition::ProfileInfo

Definition

```
struct ProfileInfo {
        PXCCapture::AudioStream::DataDesc   inputs;
        pxcCHAR             speaker[128];
        Language            language;
        Alert::Label        alerts;
        pxcU32              endOfSentence;
        pxcU32              reserved[13];
};
```

Description

The `ProfileInfo` structure describes the voice command module configurations.

Members

`inputs`	The module input requirements as defined in the **`PXCCapture::AudioStream::DataDesc`** structure.

speaker	Optional speaker name. If provided, the SDK module will adapt speech recognition to the speaker.
language	The language identifier. See the **Language** enumerator for definitions.
alerts	Supported alert messages. If the application specifies a mask, the SDK module will only send the specified alert messages to the alert handler function. See the **Alert::Label** enumerator for definitions.
endOfSentence	Suggested length in milliseconds for end of sentence detection.

Change History

This structure is introduced in SDK API 1.0.

PXCVoiceSynthesis::ProfileInfo

Definition

```
struct ProfileInfo {
      PXCCapture::AudioStream::DataDesc   outputs;
      Language     languages;
      Voice        voice;
      pxcU32       reserved[6];
};
```

Description

The `ProfileInfo` structure describes the voice TTS module configurations.

Members

language	The language of the TTS module.
voice	Choose a voice configuration.
outputs	Output audio format description; see the **PXCCapture::AudioStream::DataDesc** structure for details.

Change History

This structure is introduced in SDK API 1.0.

Enumerator Reference

PXCVoiceRecognition::Alert::Label

Description

The **`Label`** enumerator itemizes the alert messages.

Name/Description

`LABEL_VOLUME_HIGH`	The volume of the speech is too high for recognition.
`LABEL_VOLUME_LOW`	The volume of the speech is too low for recognition.
`LABEL_SNR_LOW`	The speech signal-to-noise ratio is too low for recognition.

Change History

This enumerator is introduced in SDK API 1.0.

PXCVoiceRecognition::ProfileInfo::Language

Description

The **`Language`** enumerator itemizes the supported languages.

Name/Description

`LANGUAGE_US_ENGLISH`	US English

Change History

This enumerator is introduced in SDK API 1.0.

PXCVoiceSynthesis::ProfileInfo::Language

Description

The **`Language`** enumerator itemizes the supported languages for the TTS module.

Name/Description

`LANGUAGE_US_ENGLISH`	US English

Change History

This enumerator is introduced in SDK API 1.0.

PXCVoiceSynthesis::ProfileInfo::Voice

Description

The **`Voice`** enumerator itemizes the supported voices for the TTS module.

Name/Description

`VOICE_MALE`	Use a male voice.
`VOICE_FEMALE`	Use a female voice.

Change History

This enumerator is introduced in SDK API 1.0.