Reference Guide

AudioCodes Voice Recognition Engine

AC Voca API Guide

for Windows

Version 1.0

audiocodes

ACVoca
powered by AudioCodes

Table of Contents

WEEE EU Directive

Pursuant to the WEEE EU Directive, electronic and electrical waste must not be disposed of with unsorted waste. Please contact your local recycling authority for disposal of this product.

Customer Support

For customer support, please contact your support representative or the AudioCodes support team at support@acvoca.com.

Abbreviations and Terminology

Each abbreviation, unless widely used, is spelled out in full when first used.

Document Revision Record

LTRT	Description
13180	Initial document release for Version 1.0.

Related Documentation

Document Name
AC Voca Release Notes
AC Voca Administration Guide
AC Voca API Guide for iOS and Android

Documentation Feedback

AudioCodes continually strives to produce high quality documentation. If you have any comments (suggestions or errors) regarding this document, please fill out the Documentation Feedback form on our Web site at http://online.audiocodes.com/documentation-feedback.

1 Introduction

The AC Voca Software Developer Kit (SDK) for Windows is designed for software developers and integrators, allowing the enhancement of any application with voice recognition capabilities. This promotes easy access for application functionality using natural voice commands.

Using an intuitive set of natural voice commands with high recognition accuracy levels, alongside the AC Voca pure offline capability (as the voice-engine doesn’t require any Internet connectivity in order to operate in real-time), users can enjoy an innovative, friendly and simple user-experience, driving their application usage by natural voice.

This page is intentionally left blank.

2 Getting Started

2.1 Purpose

The *NSC API Manual* includes the definitions of all the functions and data structures needed for interfacing with the NSCServer utility – a component of the NSC SpeechBlade™ Server product. It is intended for application developers building applications working with the NSCServer.

2.2 Terms and Definitions

The following basic terms are used in this manual:

Grammar	Defines the legal words or phrases to be recognized at a given point in an application.
Vocabulary	A list that defines the words that can be recognized.
Runtime-Vocabulary	A vocabulary defined in a grammar, and set at runtime (before recognition).
Resource	A single independent implementation of the NSC Speecher/Spotter speech recognizer, executing all functions defined in this API manual; simply referred to as a Resource.
Host	The hardware platform on which the voice-driven application resides.
Barge-In	The ability to speak (and be recognized) while the system is playing a prompt.

This page is intentionally left blank.

3 API Files

The following API files are used.

3.1 NSC API Files

Library:	nsc.dll, nsc.lib
Functions Header:	nsc.h
Error Codes Header:	nsc_err.h

3.2 NSC ASR API Files

In addition to the files mentioned in Section 3.1:

Functions Header:	nscasr.h
Error Codes Header:	nscasr_err.h

This page is intentionally left blank.

4 Function Definitions

The set of functions that are supported by the NSC API are defined below. References to definition of structures, types and constants are given throughout the document.

4.1 NSC API

The following describes NSC API functions.

4.1.1 Initialization/Termination Functions

4.1.1.1 NSC_Init

Description

This function initializes the NSC API.

Syntax

```
NSC_ERR_TP   NSC_Init
   (NSC_Callback_EventReport     fncEventCallBack,
   const char                    *pstrLogPathName,
   const char                    *pstrLogFileName);
```

Input

`FncEventCallBack`	Defines the name of the application callback function (see below).
`pstrLogPathName`	Defines the pointer to a string that gives the path where application (API) logs[1] will be saved.
`pstrLogFileName`	Defines the pointer to a string that gives the name of the application (API) log file.

Output

None

Return Values

NSC_ERR_TP (see Section 4.1.11).

[1]The application logs are the logs that document all the API commands passed from the application to the NSCServer and back through the NSC API (This log has no relation to recognition results).

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- This function must be called before using any other function in this API.
- The *fncEventCallBack* is used when the method for event handling is "call back" (as explained in section 6.3.1 of NSC Speecher User's Guide). When the polling method is used, this parameter is set to NULL.
- If *pstrLogPathName* and *pstrLogFileName* are both set to NULL, the NSC Speecher/Spotter will create a default log file named nsc.log, in the application's current directory.

4.1.1.2 NSC_Version_Get

Description

This function retrieves the current API version number.

Syntax

```
NSC_ERR_TP               NSC_Version_Get
   (char                 *pstrVersion,
   unsigned short        *pusSizeBytes);
```

Input

`pstrVersion`	Defines an allocated string.
`pusSizeByte`	Defines the allocated size of the string.

Output

`pstrVersion`	Defines the string containing the API version number.
`pusSizeByte`	Defines the actual size of the string returned.

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

The error message NSC_MORE_DATA will be returned if the size of the string is too small to include the version number. The actual size required is returned in pusSizeBytes and the string can be re-allocated in order to call the function again.

4.1.1.3 NSC_Terminate

Description

This function terminates the connection between the API and the application.

Syntax

```
NSC_ERR_TP NSC_Terminate()
```

Input

None

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

None

4.1.2 Server Management Functions

4.1.2.1 NSC_Server_Find

Description

This function retrieves the server ID of an active NSCServer at the provided IP address.

Syntax

```
NSC_ERR_TP NSC_Server_Find
   (const NSC_HEADER_ST      *pHeader,
   const char                *pstrIP,
   short                     *psServerID);
```

Input

`*pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structure that can be used for application purposes.
`*pstrIP`	Defines the string representing the IP address that the server ID will be retrieved from.

Output

`*psServerID`	Defines the unique ID of the server that was found. '-1' is returned if no server was found at the given IP address.

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

None

4.1.2.2 NSC_Server_Add

Description

This functions adds a specific NSCServer found on the network to the application's list (bank) of servers.

Syntax

```
NSC_ERR_TP NSC_Server_Add
   (const NSC_HEADER_ST     *pHeader,
   const     char           *pstrIP,
   unsigned long            ulPort,
   long                     lTimeout,
   short                    *psServerID);
```

Input

`*pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structure that can be used for application purposes.
`*pstrIP`	Defines the IP address of the server being added to the bank of available servers.
`ulPort`	Defines the number of the socket, the server is receiving API calls through.
`lTimeout`	Defines the timeout in milliseconds to handshake with the server.

Output

`*psServerID`	Defines the unique server ID.

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

The port number in ulPort is set according to the configuration of the NSCServer port (Please refer to the *NSCServer Configuration Manual*).

4.1.2.3 NSC_Server_Remove

Description

This functions removes a specific NSCServer from the application's list (bank) of servers.

Syntax

```
NSC_ERR_TP NSC_Server_Remove
   (const NSC_HEADER_ST      *pHeader,
   short                     sServerID,
   long                      lTimeout;
```

Input

`*pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structure that can be used for application purposes.
sServerID	Defines the unique server ID.
lTimeout	Defines the timeout in milliseconds for contacting the server.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

None

4.1.2.4 NSC_Server_InfoGet

Description

This function retrieves the information on NSCServer found on the application's server list. It can also be used to retrieve a list of all the servers that are found on the application's server list.

Syntax

```
NSC_ERR_TP NSC_Server_InfoGet
   (const NSC_HEADER_ST     *pHeader,
   short                    sServerID,
   NSC_SERVER_INFO_LIST_ST  *pServerInfo);
```

Input

`*pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structure that can be used for application purposes.
`sServerID`	Defines the unique server ID. To retrieve the full server list this value should be '-1'.
`*pServerInfo`	Defines, in synchronous mode, a pointer to an allocated NSC_SERVER_INFO_LIST_ST structure (see Section 4.1.7.5 and notes below). In asynchronous mode, this pointer can be NULL (see notes below).

Output

`*pServerInfo`	In synchronous mode, this will be a pointer to the list of available servers (see notes below).

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- When calling this function synchronously, the server list will be returned in the structure pointed to by *pServerInfo. The application will need to allocate the structure before calling the function, and setting the allocated size (bytes) in pServerInfo > usNumServers. NSC_MORE_DATA will be returned in case the application did not allocate enough memory. In this case, the correct number of defined servers is returned in pServerInfo > usNumServers. The application can call this function again after re-allocating the actual required memory.
- When calling this function asynchronously, the data will be returned in the event data as described below for function NSC_Event_Get(). In this case the pEventData>pData>cData should be cast to NSC_SERVER_INFO_LIST_ST* (see Section 4.1.7.5).

4.1.3 Event Management Functions

4.1.3.1 NSC_Event_Get

Description

This function retrieves an event from the event queue.

Syntax

```
NSC_ERR_TP NSC_Event_Get
   (short               sResourceID,
   long l               Timeout,
   NSC_EVENT_DATA_ST    *pEventData);
```

Input

`sResourceID`	Defines the Resource ID for which an event is retrieved.
`lTimeout`	Defines the maximal time to wait for an event (milliseconds). If set to '0', the function returns immediately.

Output

`*pEventData`	Defines the pointer to the event data that was returned (see Section 4.1.7.2 and notes below).

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- This function is used to retrieve four possible event types:
  - i. Function Termination Events - Events generated at function termination (generated only by functions called in asynchronous mode)
  - ii. Application Events - Events generated by application(s) (see NSC_Event_Put() in Section 4.1.3.2)
  - iii. Notification Events - generated by the server (see Section 4.1.9):
    - ♦ NSC_NOTIFY_SERVER_CONNECT
    - ♦ NSC_NOTIFY_SERVER_DISCONNECT
  - iv. Request Events (see Section 4.1.9):
    - ♦ NSC_REQUEST_AUDIOMAINSTREAM
    - ♦ NSC_REQUEST_AUDIOPROMPTSTREAM
- If sResourceID is set to NSC_ANY_RESOURCE (see Section 4.1.10) events are retrieved from any available resource.
- If sResourceID is set to NSC_NO_RESOURCE (see Section 4.1.10) events that have no relation to a specific resource are retrieved.
- If during lTimeout no events are retrieved, the function will return with NSC_NO_EVENT (see Section 4.1.11).
- See Section 4.1.9 for the list of events.
- Most of the events do not include any data (in the structure NSC_EVENT_DATA_ST member pData). Some specific events (e.g., type i. above) are used to pass data to the application. An example of such a function is NSC_Server_InfoGet (see Section 4.1.2.4).
- The data is written to pEventData > pData, which is an NSC_DATA_ST structure (see Section 4.1.7.1).
  In this case, the application will perform the following steps before calling NSC_ Event_Get():
  1) The pEventData□pData□cData should be allocated with at least a size of 1 Kbytes.
  2) Set pEventData□pData□ulSizeBytes to the size (bytes) allocated at stage 1.

  When an event with data is retrieved, the following steps should be followed in order to retrieve the data:
  1) If pEventData>Error is NSC_MORE_DATA then the application did not allocate enough memory. The size of the memory to be allocated is returned in pEventData>pData>ulSizeBytes. Repeat Steps 1) and 2) above with this size.
  2) Cast pEventData>pData>cData to a pointer to the structure required. For example for the function NSC_Server_InfoGet (see Section 4.1.2.4) the casting is to a pointer to NSC_SERVER_INFO_LIST_ST (see Section 4.1.7.5).

4.1.3.2 NSC_Event_Put

Description

This function puts an event into the event queue.

Syntax

```
NSC_ERR_TP NSC_Event_Put
   (const NSC_HEADER_ST     *pEventData;
```

Input

`pEventData`	Defines the pointer to the event data (see Section 4.1.7.2) to be generated in the event queue.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- This function generates events of type ii (see above)
- If sResourceID is set to NSC_NO_RESOURCE the event is generated without relation to a specific resource.

4.1.4 Resource Management Functions

4.1.4.1 NSC_Resource_Get

Description

This function allocates an NSC Speecher/Spotter resource from the available pool of NSCServer resources.

Syntax

```
NSC_ERR_TP NSC_Resource_Get
   (const char          *pstrResourceType,
   NSC_FLAG_TP          ClearHistory,
   long                 lTimeout,
   const char           *pstrServerIP,
   short                *psResourceID);
```

Input

`*pstrResourceType`	Specifies the type of resource required. If NULL or empty string, then the first available resource will be obtained with no selection criteria. Refer to the *NSCServer Configuration Manual* for resource configuration.
`ClearHistory`	If set to NSC_YES, this function will clean the resource history. If set to NSC_NO, resource history is not cleaned.
`lTimeout`	Defines the maximal period of time in milliseconds to wait for an available resource. When set to"–1", the function will wait indefinitely (until a suitable resource is available).
`*pstrServerIP`	Defines a specific IP address of the server from which this resource should be allocated.
`*pstrServerIP`	Defines a specific IP address of the server from which this resource should be allocated.

Output

`*psResourceID`	Defines the NSC Speecher/Spotter resource ID.

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous):

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- The resource history includes:
    - Loaded grammars
    - All parameter values set by the application (e.g., log directory, recognition parameters)
- The resource type consists of three optional strings (Please refer to *NSCServer Configuration Guide* for further information on setting resources):
    - Language code (e.g., En-Us)
    - Tier (e.g., Tier=4)
    - Label (e.g., En-US-T4-CONFIRM), which enables to create sets of resources to be accessed by their labels only.

  If timeout is over and there were no available resources, the function will return:
    - NSC_TIME_OUT – if all resources of the requested type have been allocated.
    - NSC_PARAM_INVL – the requested type of resource is not available (was not configured)
- In order to force getting a resource from a specific server location, assign the pstrServerIP with a specific server IP; otherwise, use NULL to indicate that resource origin is chosen internally by NSC.

4.1.4.2 NSC_Resource_Free

Description

Frees an NSCServer resource. The resource is added to the pool of available NSCServer resources.

Syntax

```
NSC_ERR_TP NSC_Resource_Free (short sResourceID);
```

Input

`sResourceID`	Defines the NSCServer resource ID.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

None

4.1.4.3 NSC_Resource_Abort

Description

This function aborts the operation of an NSCServer resource.

Syntax

```
NSC_ERR_TP NSC_Resource_Abort
   (const NSC_HEADER_ST      *pHeader,
   short                     sResourceID);
```

Input

`*pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structures that can be used for application purposes.
`sResourceID`	Defines the NSC Speecher/Spotter resource ID.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous):

Both

Relevant to (NSC Speecher/NSC Spotter/Both):

Both

Notes

- If a resource operation is aborted while the resource is not in idle state, the operation will return with *NSC_ABORTED* error code (see Section 4.1.11).
- In asynchronous mode this function creates an *NSC_EVENT_RESOURCE_ABORT* (see Section 4.2.7).

4.1.4.4 NSC_Resource_InfoGet

Description

This function retrieves information on a specific resource.

Syntax

```
NSC_ERR_TP NSC_Resource_InfoGet
   (const NSC_HEADER_ST     *pHeader,
   short                    sResourceID,
   NSC_RESOURCE_INFO_ST     *pResourceInfo);
```

Input

`*pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structures that can be used for application purposes.
`sResourceID`	Defines the NSC Speecher/Spotter resource ID.
`*pResourceInfo`	In Synchronous mode, this is a pointer to an allocated NSC_RESOURCE_INFO_ST structure (see Section 4.1.7.6 and the notes below). In Asynchronous mode this pointer can be NULL (see the notes below).

Output

`*pResourceInfo`	In synchronous mode, this is a pointer to the resource information (see notes below).

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- The resource type that is returned in pResourceInfo->strType will contain the resource label if it exists; otherwise it will return the resource type (e.g., En-Us,Tier=2).
- When calling this function synchronously the resource information will be returned in the structure pointed to by *pResourceInfo. The application will need to allocate the structure before calling this function, allocating the memory required for NSC_RESOURCE_INFO_ST. NSC_MORE_DATA will be returned in case the application did not allocate enough memory. The application can call this function again after re-allocation of the required memory.
- When calling this function asynchronously the data will be returned in the event data as described in Section 4.1.3.1 for function NSC_Event_Get(). In this case the pEventData>pData>cData should be cast to NSC_RESOURCE_INFO_ST* (see Section 4.1.7.6).

4.1.5 Audio Channel Management Functions

The NSC SpeechBlade™ Server supports three methods of sending PCM audio samples to the NSC Speecher/Spotter resources (discussed below). The method of sending PCM audio samples is set at run-time by the application. The method is set before recognition is started by calling the *NSC_Resource_AudioChannelSet()* function.

The supported methods of sending PCM audio samples are:

1. A direct connection to the telephony board(s), using a CT bus interface. The connection between the boards is done through a standard CT bus cable (see the *NSC SpeechBlade™ Server Hardware Installation Guide*). When using this method the application sets the connection between a given audio PCM channel (stream, slot) and the NSC Speecher/Spotter resource, using the *NSC_Resource_AudioChannelSet()* function.
   Note: NSCServer is configured according to the CT bus settings of the telephony board(s) (e.g., SC, H100). This configuration is done once for each server having a CT bus connection between the telephony board(s) and NSCSpeechBlade™(s). Please refer to *NSCServer Configuration Manual* for the CT bus configuration procedure.
2. 'Feeding' PCM samples at run time through the PCI bus (H/W interface between NSCSpeechBlade™(s) and the Host). The application sets the connection to the NSC Speecher/Spotter resource as PCI, using the *NSC_Resource_AudioChannelSet()* function. After recognition is started, the application sends blocks of PCM samples using the *NSC_Resource_AudioMainStream()* function for the main audio channel and *NSC_Resource_AudioPromptStream()* for the prompt audio channel.
3. RTP protocol - RTP/RTCP protocol session actually consists of two separate channels – one for audio samples (RTP stream) and another one for statistical and informational data related to the audio stream (RTCP channel). NSC Server is an RTP/RTCP "receiver" that utilizes audio and statistical information supplied by RTP "sender" (the application).

The NSC engines (Speecher/Spotter) supports PCM audio sampled at 8000 samples/sec, in one of three PCM Formats:

1. Linear (16-bit), defined by NSC_AUDIO_LINEAR (not supported in the RTP mode).
2. A-law (8-bit), defined by NSC_AUDIO_ALAW.
3. µ-law (8-bit), defined by NSC_AUDIO_ULAW.

4.1.5.1 NSC_Resource_AudioChannelSet

Description

This function sets the type of PCM audio samples and the method that will be used to send PCM audio samples to the resource (refer to 4.1.5).

Syntax

```
NSC_ERR_TP NSC_Resource_AudioChannelSet
   (const    NSC_HEADER_ST *pHeader,
   short                    sResourceID,
   unsigned short           usMainStreamFormat,
   unsigned short           usPromptStreamFormat,
   short                    sMainStream,
   short                    sMainSlot,
   short                    sPromptStream,
   short                    sPromptSlot);
```

Input

`*pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structures that can be used for application purposes.
`sResourceID`	Defines the NSC Speecher/Spotter resource ID.
`usMainStreamFormat`	Defines the PCM format of the main stream (see Section 4.1.10 for possible values).
`usPromptStreamFormat`	Defines the PCM format of the prompt stream (see Section 4.1.10 for possible values).
`sMainStream`	Defines the main CT bus stream ID. (Set the value to NSC_FEED_STREAMING_MODE if the CT interface is not used).
`sMainSlot`	Defines the main CT bus timeslot within the stream where PCM samples are allocated. (Set the value to NSC_FEED_STREAMING_MODE if the CT interface is not used).

`sPromptStream`	Defines the prompt CT bus stream ID. (Set the value to NSC_FEED_STREAMING_MODE if the CT interface is not used).
`sPromptSlot`	Defines the prompt CT bus timeslot within the stream where PCM samples are allocated. (Set the value to NSC_FEED_STREAMING_MODE if the CT interface is not used).

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- Working in Barge-In mode is not relevant to NSC Spotter for this reason parameters sPromptStream and sPromptSlot are ignored when using the NSC Spotter and should be set to NSC_NO_AUDIO.
- When using the CT bus method for transferring the PCM audio, this function connects the CT bus PCM streams to a specific resource.
- The main stream (represented by: sMainStream and sMainSlot) is the stream containing the utterance for recognition.
- The prompt stream (represented by: sPromptStream and sPromptSlot) is the stream containing the played prompt and is used for a system running with barge-in that requires echo cancellation.

- If sMainStream, sMainSlot, sPromptStream and sPromptSlot are all set to NSC_FEED_STREAMING_MODE. The PCM bus method for transferring the PCM audio is used.
- If sPromptStream and sPromptSlot are set to NSC_NO_AUDIO, and usPromptStreamFormat is set to NSC_NO_AUDIO, echo cancellation is not activated on this resource.

4.1.5.2 NSC_Resource_AudioMainStream

Description

Sends a block of PCM samples representing a played prompt (or part of it) from the prompt channel to a resource. This function is used only in case of using feed streaming mode (see Section 4.1.5.1).

Syntax:

```
NSC_ERR_TP NSC_Resource_AudioMainStream
   (short                    sResourceID,
   long                      lSizeBytes,
   const void                *pvAudioStream);
```

Input

- sResourceID - NSC Speecher/Spotter resource ID
- lSizeBytes – The size of the data block to be transferred, in bytes.
- *pvAudioStream – Pointer to the PCM data block

`sResourceID`	Defines the NSC Speecher/Spotter resource ID.
`lSizeBytes`	Defines the size of the data block to be transferred, in bytes.
`*pvAudioStream`	Defines the pointer to the PCM data block.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

N/A

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- The application is responsible for sending the blocks in sequential order; each block (pointed by pvAudioStream) is sent after getting the NSC_REQUEST_AUDIOMAINSTREAM event from the resource. The first NSC_REQUEST_AUDIOMAINSTREAM is generated by the resource immediately after calling the recognition function (see Section 4.2.2.1).
- There are two normal cases of termination of the process of streaming audio blocks:
    - Recognition is finished (decision is made by NSC Speecher/Spotter resource). In this case, no more NSC_REQUEST_AUDIOMAINSTREAM events are produced and recognition results are available.
    - Application has no more audio blocks to send (e.g., streaming from audio files), and recognition has not ended yet. In this case, the application needs to send one empty block with lSizeBytes assigned as NSC_NO_AUDIO. This will force the recognition to terminate and extract recognition results. Otherwise, if the application stops streaming, the resource is in idle waiting (no timeout) to get more audio blocks.
    - **Note:** It is always possible to terminate the recognition by aborting the resource (call NSC_Resource_Abort). In this case there will be no recognition results.
- The resource will disregard all blocks sent before or after the recognition takes place (this does not produce an error).
- Block size (lSizeBytes) must be an even number (except when assigned as NSC_NO_AUDIO for streaming termination.
- Recommended (not mandatory) block sizes to send to the resource are:
    - For Linear PCM format (sample size 2 bytes): 768 bytes
    - For A-law/u-law formats (sample size 1 byte): 896 bytes

4.1.5.3 NSC_Resource_AudioPromptStream

Description

Sends a block of PCM samples representing an utterance (or part of it), from the main channel to a resource. This function is used only in case of using feed streaming mode (see Section 4.1.5.1) and the Barge-in operation (see Section 7.3.1 in the *NSC Speecher/Spotter User's Guide*).

Syntax

```
NSC_ERR_TP NSC_Resource_AudioPromptStream
   (short                sResourceID,
   long                 lSizeBytes,
   const void           *pvAudioStream);
```

Input

`sResourceID`	Defines the NSC Speecher/Spotter resource ID.
`lSizeBytes`	Defines the size of the data block to be transferred, in bytes.
`*pvAudioStream`	Defines the pointer to the PCM data block.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

N/A

Relevant to (NSC Speecher/NSC Spotter/Both)

NSC Speecher only

Notes

- This function is used for echo cancellation for applications running with barge-in using NSC echo cancellation capabilities.
- There are two normal cases of termination of the process of streaming audio blocks:
  - Recognition is finished (decision is made by NSC Speecher resource). In this case, no more NSC_REQUEST_AUDIOPROMPTSTREAM events are produced and recognition results are available.
  - Application has no more audio blocks to send (e.g., streaming from audio files), and recognition has not ended yet. In this case, the application needs to send one empty block with lSizeBytes assigned as NSC_NO_AUDIO. This will force the recognition to terminate and extract recognition results. Otherwise, if the application stops streaming, the resource is in idle waiting (no timeout) to get more audio blocks.
  - **Note:** It is always possible to terminate the recognition by aborting the resource (call NSC_Resource_Abort). In this case there will be no recognition results.
- The resource will disregard all blocks sent before or after the operation (e.g. recognition) took place. In this case, blocks and an appropriate error are given (refer to 4.1.11).
- Block size (lSizeBytes) must be an even number.
- Experience shows that the most effective block sizes to send to the resource are:
  - For Linear PCM format (sample size 2 bytes): 768 bytes
  - For A-law/u-law formats (sample size 1 byte): 896 bytes

4.1.6 RTP Management Functions

The following is a list of new RTP related API functions and data structures.

4.1.6.1 NSC_RTPSession_Open

Description

Start a new RTP/RTCP session. RTP/RTCP session is created on a specific server (passed as an input parameter). RTP/RTCP endpoints are returned to the client inside session info parameter. Created session is not associated with ASR resource and is not ready to process RTP/RTCP packets.

Syntax

```
NSC_ERR_TP NSC_RTPSession_Open
    (const NSC_HEADER_ST               *i_pHeader,
    const char*                        i_strClientIp,
    Short                              i_sClientRtcpPort,
    const char*                        i_strServerIp,
    NSC_RTP_SESSION_ST                 *o_pSessionInfo)
```

Input

`NSC_HEADER_ST *i_pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structure.
`i_strClientIp`	Defines the client IP for sending RTCP protocol Receiver Reports (RRs).
`i_sClientRtcpPort`	Defines the client port for sending RTCP protocol Receiver Reports (RRs) i_strServerIp - IP address of the relevant NSC server. When NULL is passed the configured server is used, if more than one configured servers exist and this parameter is set to NULL an error is returned.
`o_pSessionInfo`	Defines the output structure defining the properties of a newly created RTP session.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

N/A

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

4.1.6.2 NSC_RTPSession_Close

Description

This function closes the RTP/RTCP session.

Syntax

```
NSC_ERR_TP NSC_RTPSession_Close
   (const NSC_HEADER_ST     *i_pHeader,
   NSC_RTP_SESSION_ST       *i_pSessionInfo)
```

Input

`i_pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structure.
`i_pSessionInfo`	Defines the structure defining the properties of an already open RTP session.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

N/A

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

4.1.6.3 NSC_Resource_RTPChannel_Set

Description

Associates RTP session with the particular ASR resource. Stores ASR resource number in the session. Setting the resource is allowed on idle sessions only.

Syntax

```
NSC_ERR_TP NSC_Resource_RTPChannel_Set
    (const NSC_HEADER_ST      *i_pHeader,
    short                     i_sResourceId
    unsigned long             i_lRtpSessionId)
```

Input

`i_pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structure.
`i_sResourceId`	Defines the Resource ID.
`i_lRtpSessionId`	Defines the RTP Session id (returned by the NSC_RTPSession_Open call) that identifies sessions that should be associated with a given resource.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

N/A

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

4.1.6.4 NSC_Resource_RTPChannel_Reset

Description

De-associate RTP session from ASR resource. Resetting resource is allowed on IDLE sessions only.

Syntax

```
NSC_ERR_TP NSC_Resource_RTPChannel_Reset
        (const NSC_HEADER_ST *i_pHeader,
        short i_sResourceId)
```

Input

`i_pHeader`	Defines the pointer to NSC_HEADER_ST which includes the flag for synchronous/asynchronous operation and additional data structure.
`i_sResourceId`	Defines the Resource ID.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.1.11)

Include Files

nsc.h, nsc_err.h

Supported Modes (Synchronous/Asynchronous)

N/A

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

4.1.7 NSC API Structures

The following describes NSC API structures.

4.1.7.1 NSC_DATA_ST

NSC_DATA_ST (Data Structure)		
Member Name	**Type**	**Description**
ulSizeBytes	unsigned long	Defines the size of data to transfer (in bytes).
cData[1]	unsigned char	Defines the transferred data.

Defines the general data structure for transferring data between the host and the NSC Speecher/Spotter resources.

4.1.7.2 NSC_EVENT_DATA_ST

NSC_EVENT_DATA_ST (Event Data Structure)		
Member Name	**Type**	**Description**
sResourceID	Short	Defines the ID of the resource that generated the event.
usDescriptor	unsigned short	Defines the code associated with the event.
Error	NSC_ERR_TP	Defines the error code associated with the event (see Section 4.1.11).
sAuxData	short [NSC_AUXILIARY_DATA_LEN]	Defines the Bytes sent by the host in function call; returns to the host without any change (e.g., for debugging).
*pData	NSC_DATA_ST (see Section 4.1.7.1)	Defines the data associated with the event if relevant, otherwise NULL.

Defines a structure holding the event data retrieved from the NSCServer event queue.

4.1.7.3 NSC_HEADER_ST

NSC_HEADER_ST (Function Header Structure)		
Member Name	**Type**	**Description**
SyncAsyncFlag	NSC_FLAG_TP	Defines the size of data to transfer (in bytes)
sAuxData	short [NSC_AUXILIARY_DATA_LEN]	Defines the bytes sent by the host in function call; returns to the host (through the *NSC_EVENT_DATA_ST)* without any change (e.g., for debugging).

Defines a structure provided as a header for all functions that supports asynchronous operational mode.

4.1.7.4 NSC_SERVER_INFO_ST

NSC_SERVER_INFO_ST (Single Server Info Structure)		
Member Name	**Type**	**Description**
sID	Short	Defines the ID of the Server.
strip	Char [NSC_SERVER_IP_LEN+1]	Defines the I.P. address of the server.
ulPort	unsigned long	Defines the port used for the server.
lLastConnectTime	Long	Defines the server's last connection time.
lLastDisconnectTime	Long	Defines the server's last disconnection time.
usNumReservedResources	unsigned short	Defines the number of resources reserved by the server for use by this application.

Defines the structure representing server information defined in the system.

4.1.7.5 NSC_SERVER_INFO_LIST_ST

NSC_SERVER_INFO_LIST_ST (Server List Structure)		
Member Name	**Type**	**Description**
usNumServers	unsigned short	Number of servers.
Info[1]	NSC_SERVER_INFO_ST	Structure holds server's information.

Defines the structure representing information for a list of servers.

4.1.7.6 NSC_RESOURCE_INFO_ST

NSC_ RESOURCE_INFO_ST (Single Resource Structure)		
Member Name	**Type**	**Description**
strType	char [NSC_RESOURCE_TYPE_LE N+1]	Defines the resource Type/Label.
sID	short	Defines the NSC Speecher/Spotter resource ID.
sInternalID	short	Defines the internal resource ID on the server the resource was allocated on.
sServerID	short	Defines the unique server ID.

Defines the structure representing resource information.

4.1.7.7 NSC_PARAM_ST

NSC_ PARAM_ST (Single Parameter Structure)		
Member Name	**Type**	**Description**
sID	short	Defines the parameter ID.
ulSizeBytes	unsigned long	Defines the parameter size (bytes).
sValue	short	Defines the short parameter value.
lValue	Long	Defines the long parameter value.
pstrValue	char*	Defines the char* parameter value.

Defines the structure representing parameter ID, size and value.

4.1.7.8 NSC_PARAM_LIST_ST

NSC_ PARAM_LIST _ST (Parameter List Structure)		
Member Name	**Type**	**Description**
usNumParameters	unsigned short	Defines the number of parameters on the list.
Param[1]	NSC_PARAM_ST	Defines the array of NSC_PARAM_ST containing the data of the parameters.

Defines the structure representing a list of parameters data.

4.1.7.9 NSC_RTP_SESSION_ST

NSC_RTP_SESSION_ST (Parameter List Structure)		
Member Name	**Type**	**Description**
lSessionId	unsigned long	Unique number that identifies each RTP session. This ID is generated by the NSCServer that creates the session.
sServerRtpPort	unsigned short	RTP port that should be used when sending RTP stream to the server
sServerRtcpPort	unsigned short	RTCP port that should be used when sending RTCP stream to the server
strServerName[256]	Char	Hostname of the server computer that will handle incoming RTP/RTCP streams

Data structure that contains RTP session information

4.1.8 NSC API Types

Name	Type	Description
NSC_ERR_TP	short	NSC Error Type. For values refer to 4.1.11
NSC_FLAG_TP	short	NSC Flag Type. For values refer to 4.1.10
NSC_Callback_EventReport	_cdecl*	Type of 'call back' function

4.1.9 NSC API Event Codes

Name	Type	Description
Event Codes		
Server Related Events		
NSC_EVENT_SERVER_ADD	201	Completion Event for NSC_Server_Add()
NSC_EVENT_SERVER_REMOVE	202	Completion Event for NSC_Server_Remove()
NSC_EVENT_SERVER_FIND	203	Completion Event for NSC_Server_Find()
NSC_EVENT_SERVER_INFOGET	204	Completion Event for NSC_Server_InfoGet()
Resource Related Events		
NSC_EVENT_RESOURCE_INFOGET	301	Completion Event for NSC_Resource_InfoGet()
NSC_EVENT_RESOURCE_ABORT	302	Completion Event for NSC_Resource_Abort()
NSC_EVENT_RESOURCE_AUDIOCHANNELSET	303	Completion Event for NSC_Resource_AudioChannelSet()
NSC Generated Events		
NSC_REQUEST_AUDIOMAINSTREAM	1001	Generated when the NSC Speecher/Spotter is waiting for data representing the audio stream on the Main Channel

Name	Type	Description
NSC_REQUEST_AUDIOPROMPTSTREAM	1002	Generated when the NSC Speecher/Spotter is waiting for data representing the audio stream on the Prompt Channel
Notifications		
NSC_NOTIFY_SERVER_CONNECT	2001	Server has connected
NSC_NPTIFY_SERVER_DISCONNECT	2002	Server has disconnected
RTP Related Events		
NSC_NOTIFY_RTP_SESSION_ERR	2010	Asynchronous notification sent when RTP error occurs
NSC_NOTIFY_RTP_SESSION_START	2011	Asynchronous notification sent when server is waiting for RTP stream to start
NSC_EVENT_RTPSESSION_OPEN	501	Asynchronous method call completes
NSC_EVENT_RTPSESSION_CLOSE	502	Asynchronous method call completes
NSC_EVENT_RESOURCE_RTPCHANNEL_SET	503	Asynchronous method call completes
NSC_EVENT_RESOURCE_RTPCHANNEL_RESET	504	Asynchronous method call completes

4.1.10 NSC API Constants

Name	Type	Description
General Definitions		
NSC_YES	1	True
NSC_NO	0	False
NSC_SYNC_FLAG	1	Flag to indicate synchronous operational mode
NSC_ASYNC_FLAG	0	Flag mode to indicate asynchronous operational

Name	Type	Description
NSC_FEED_STREAMING_MODE	-1	Used to set the streaming mode to FEED PCM audio samples (used in NSC_Resource_AudioChannelSet)
NSC_SERVER_IP_LEN	64	The maximal length allocated for IP address
NSC_EVENT_USER_OFFSET	10000	Offset used for event ID's for use in application (using NSC_Event_Put())
Type of PCM Format		
NSC_NO_AUDIO	-1	Indication that channel has no input audio (e.g., when setting channel not in EC mode)
NSC_LINEAR	0	Raw PCM audio data
NSC_ULAW	1	U-law encoded PCM audio data
NSC_ALAW	2	A-law encoded PCM audio data
Echo Cancellation Operational Modes		
NSC_EC_DISABLE	0	NSC Speecher echo-cancellation is disabled
NSC_EC_ADAPT_RESTART	1	Echo-cancellation is restarted (initialized)
NSC_EC_ADAPT_CONTINUE	2	Echo-cancellation adaptation continued from previous recognition
NSC_EC_ADAPT_FREEZE	3	Adaptation is 'frozen' (continues from same point as in previous recognition)
Auxiliary Data Length in Message		
NSC_AUX_DATA_LEN	4	Auxiliary data length of message
Resource Type and Selection of Resources		
NSC_RESOURCE_TYPE_LEN	256	The maximal length for resource type (e.g. "En-Us,Tier=2")
NSC_ANY_RESOURCE	-1	Code to get event from any resource
NSC_NO_RESOURCE	-9999	Code to put/get an event that is not on a specific resource (e.g. Server events).
Parameter Constants		

4.1.11 NSC API Error Codes

NSC_ERR_TP (Function Error Return Codes – As defined in Nsc_err.h)		
Name	**Value**	**Description**
NSC_NO_ERR	0	No error
NSC_INIT_FAIL	11	Initialization failure
NSC_MSG_FAIL	12	Message failed to reach the server (no connection to server)
NSC_TIME_OUT	13	The timeout limit has been reached
NSC_RESOURCE_INVL	21	Function addressed invalid resource ID
NSC_RESOURCE_BUSY	22	Attempt to perform an operation on a busy resource
NSC_MALLOC_FAIL	31	Memory allocation failure
NSC_MORE_DATA	32	Attempt to perform an operation on a busy resource
NSC_PARAM_INVL	41	Invalid parameter has been provided
NSC_WRONG_USAGE	42	Function was used incorrectly
NSC_PATH_TOO_LONG	51	The path provided is longer than allowed
NSC_FILE_ERR	61	Error in file manipulation (e.g., *fopen()*, *fwrite()*, *fread()*).
NSC_RT_OVERRUN	71	Processing cannot be completed in real time
NSC_MSG_QUE_OVFL	72	Host failed to read message(s) from a resource in real time
NSC_PCM_RX_FAIL	73	Resource failed to receive PCM samples
NSC_ABORTED	101	Abort request encountered
NSC_NO_EVT	201	No events were returned during the period given in the *NSC_Event_Get()* function
NSC_SERVER_ERROR	3001	General error in NSCServer
NSC_SERVER_INVL	3002	Attempt to perform an operation on an invalid server
NSC_SERVER_ALREADY_EXISTS	3003	Attempt to add a server that already exists on the list
NSC_SERVER_NO_SOCKET	3004	IP Address provided is invalid or a server does not exist on the provided IP address.
NSC_SERVER_BUSY	3005	Attempt to perform an operation on a busy server

4.2 NSC ASR API

The following describes NSC ASR API functions.

4.2.1 Initialization/Termination Functions

4.2.1.1 NSCASR_Init

Description

This function initializes the NSCASR API.

Syntax

```
NSC_ERR_TP NSCASR_Init();
```

Input

None

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous Only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

This function must be called before calling any functions using NSCASR API.

4.2.1.2 NSCASR_Version_Get

Description

This function retrieves the current NSCASR API version number.

Syntax

```
NSC_ERR_TP    NSCASR_Version_Get
                (char              *pstrVersion,
                 short             *psSize);
```

Input

`*pstrVersion`	Defines the allocated string.
`*psSize`	Defines the allocated size of the string.

Output

`*pstrVersion`	Defines the string containing the API version number.
`*psSize`	Defines the actual size of the string returned.

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous Only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

The NSC_MORE_DATA error message will be returned, if the size of the string is too small to include the Version Number. The actual size required is returned in *psSize* and the string can be re-allocated in order to call the function again.

4.2.1.3 NSCASR_Terminate

Description

This function terminates the NSCASR API.

Syntax

```
NSC_ERR_TP NSCASR_Terminate();
```

Input

None

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9).

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous Only

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

After this function is called, no other NSCASR API function except *NSC_ASR_Init()*, can be called.

4.2.2 Recognition Functions

4.2.2.1 NSCASR_Recognize

Description

Calling this function starts the speech recognition process on the given resource. The recognition is performed on audio streams (provided through the CT bus or through the PCI bus) according to the grammars and parameters that were set by the application. The function can be activated both in synchronous and asynchronous modes. The NSCASR_Recognize_ResultGet function is used to retrieve the recognition results.

Syntax

```
NSC_ERR_TP NSCASR_Recognize
   (const NSC_HEADER_ST      *pHeader,
   short                      sResourceID,
   const NSCASR_RCG_PARAMS_ST *pRecognitionParameters,
   short                      sGrammarIdx);
```

Input

`pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the ID of the resource that will be used for the recognition that was received after calling NSC_Resource_Get (see Section 4.1.4.1).
`pRecognitionParameters`	Defines the pointer to the Recognition Parameters Structure (see Section 4.2.6.2).
`sGrammarIdx`	Defines the index of the grammar to be used in the recognition.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

NSC Speecher Only

Notes

- All the parameters that are not set as NSC_PARAM_DEFAULT_VALUE (in the structure of *pRecognitionParameters) will overwrite the current default parameters assigned for this resource. These parameters are applied only for the current recognition operation. After recognition is complete, the default parameters are not changed and remain valid for the next recognition process.
- For recognition using the active grammars, this function should be called with grammar index set to NSCASR_ACTIVE_GRAMMAR_INDEX (see Section 4.2.8).
- The NSCASR_EVENT_CONTENT_SPOTTED (see Section 4.2.7) event is generated to indicate that relevant speech has been detected based on the content of the grammar.
- This event can be used in barge-in mode in order to stop the playing of the prompt.
- If the PCM audio streams (see Section 4.1.5) are received through the PCI bus interface, an NSC_REQUEST_AUDIOMAINSTREAM event is generated when the resource is ready to receive PCM streams after this function is called.
- If the recognition was activated with echo cancellation (see Section 3.7.1 of the *NSC Speecher User's Guide*) an NSC_REQUEST_AUDIOPROMPTSTREAM event is generated when the resource is ready to receive PCM streams of the prompt being played.

4.2.2.2 NSCASR_Recognize_TimersStart

Description

This function starts the 'no input timeout' timer on the given resource. It is useful in barge-in mode where ‘no input timeout' might be dependent on prompt duration (see Section 7.3.2 of the *NSC Speecher User’s Guide*).

Syntax

```
NSC_ERR_TP NSCASR_Recognize_TimersStart
        (const NSC_HEADER_ST *pHeader,
        short                sResourceID);
```

Input

`pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the ID of the resource that will be used for the recognition that was received after calling NSC_Resource_Get (see Section 4.1.4.1).

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

NSC Speecher Only

Notes

- Each time this function is called, the "no input timeout" timer will reset. The "no input timeout" duration limitation is determined by the parameter usNoInputTimeout (a member of NSCASR_ACQ_PARAMS_ST).
- This function can only be called on a resource that is currently in the recognition process.
- If the parameter StartTimers (a member of NSCASR_ACQ_PARAMS_ST) (see Section 4.2.6.1) is set to NSC_NO, the "no input timeout" timer will not start until this function is called.

4.2.2.3 NSCASR_Recognize_ResultsGet

Description

This function retrieves the recognition results structure containing all the information of the recognized utterance.

This function must be called after recognition is completed successfully. If NSCASR_Recognize() is called asynchronously this function must be called only after an NSCASR_EVENT_RECOGNIZE event is received.

Syntax

```
NSC_ERR_TP NSCASR_Recognize_ResultsGet
        (short               sResourceID,
        NSCASR_RCG_RES_ST    *pRecognitionResults);
```

Input

`sResourceID`	Defines the ID of the resource that will be used for the recognition that was received after calling NSC_Resource_Get (see Section 4.1.4.1).
`*pRecognitionResults`	Defines the pointer to NSCASR_RCG_RES_ST (see Section 4.2.6.5).

Output

`*pRecognitionResults`	Defines the pointer to NSCASR_RCG_RES_ST (see Section 4.2.6.5) containing the results.

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Synchronous only

Relevant to (NSC Speecher/NSC Spotter/Both)

NSC Speecher Only

Notes

- The pRecognitionResults pointer should be allocated with at least a size of 1Kbytes.
- The *pRecognitionResults > Size should be assigned with the size (bytes) of the allocation.
- Error of NSC_MORE_DATA is returned if the size of the allocation is too small for storing the recognition results.
- The resource assigns the actual needed allocation size in *pRecognitionResults > Size.

4.2.3 Grammar Management Functions

4.2.3.1 NSCASR_Grammar_Load

Description

This function loads a compiled grammar from the host memory to a specific resource in the NSC Speecher/Spotter memory. This adds the grammar to the pool of grammars available for recognition of the resource.

Syntax

```
NSC_ERR_TP NSCASR_Grammar_Load
   (const NSC_HEADER_ST        *pHeader,
   short                       sResourceID,
   short                       sGrammarIdx,
   const NSC_DATA_ST           *pGrammarData);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`sGrammarIdx`	Defines the grammar index (a unique number representing a grammar loaded to the resource).
`*pGrammarData`	Defines the pointer to a data structure (see Section 4.1.7.1) containing grammar data.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- Data in pGrammarData is read from the compiled grammar file in the following way:
  - i. Open the compiled grammar file in binary mode.
  - ii. Read the first long word, which is the Offset indicating the beginning of relevant data to read.
  - iii. Read a second long word that indicates the size (in bytes) of data to read.
  - iv. Perform a seek operation (from beginning of file), using the offset number read in Step (ii).
  - v. Allocate a memory block of Size bytes.
  - vi. Read size bytes into allocated block.
  - vii. Close the file.
- Previously loaded grammars with the same sGrammarIdx must be removed from the resource memory prior to loading a new grammar on this index (see Section 4.2.3.6).

4.2.3.2 NSCASR_Grammar_Activate

Description

This function activates a list of pre-loaded grammars.

Syntax

```
NSC_ERR_TP NSCASR_Grammar_Activate
   (const NSC_HEADER_ST     *pHeader,
   short                    sResourceID,
   NSC_GRAMMAR_LIST_ST      *pGrammarList);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`sGrammarList`	Defines the grammar index (a unique number representing a grammar loaded to the resource).
`*pGrammarData`	Defines the pointer to the list of grammars indexes that should be activated in a NSC_GRAMMAR_LIST_ST structure (see Section 4.2.6.7).

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9).

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- All grammars on the list must be loaded prior to grammar activation.
- All run-time vocabularies (required in the grammar) must be loaded prior to grammar activation.
- This function can be called numerous times; each call can only add grammars to the list (i.e., grammars that are not already activated).
- To perform recognition with the active grammars NSCASR_Recognize should be called (see Section 4.2.2.1) with parameter sGrammarIdx set to NSCASR_ACTIVE_GRAMMAR_INDEX (see Section 4.2.8).

4.2.3.3 NSCASR_Grammar_DeActivate

Description

This function deactivates a list of grammars.

Syntax

```
NSC_ERR_TP NSCASR_Grammar_DeActivate
        (const NSC_HEADER_ST      *pHeader,
        short                     sResourceID,
        NSC_GRAMMAR_LIST_ST       *pGrammarList);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`pGrammarList`	Defines the pointer to the list of the grammars indexes that should be deactivated in a NSC_GRAMMAR_LIST_ST structure (see Section 4.2.6.7).

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9).

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- If one of the grammars indexes on the grammar list is *NSCASR_ACTIVE_GRAMMAR_INDEX* (see Section 4.2.8), then all active grammars will be deactivated.
- Performing *NSCASR_Grammar_Remove()* (see Section 4.2.3.6) will automatically de-activate the removed grammar from the active grammar.

4.2.3.4 NSCASR_Grammar_ActiveGet

Description

This function returns a list of currently activated grammars.

Syntax

```
NSC_ERR_TP NSCASR_Grammar_ActiveGet
        (const NSC_HEADER_ST *pHeader,
        short                sResourceID,
        NSC_GRAMMAR_LIST_ST *pGrammarList);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`*pGrammarList`	Defines a pointer, in synchronous mode, to an allocated NSC_GRAMMAR_LIST_ST structure (see Section 4.2.6.7 and notes below). In asynchronous mode this pointer can be NULL (see notes below).

Output

`*pGrammarList`	In synchronous mode, this will be a pointer to the list of active grammars (see notes below).

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- When calling this function synchronously the grammar list will be returned in the structure pointed to by pGrammarList.
- The application will need to allocate the structure before calling this function and setting the allocated size in pGrammarList>sCount. The error message NSC_MORE_DATA will be returned in case the application did not allocate enough memory. In this case the correct number of active grammars is returned in pGrammarList>sCount. The application can call this function again after re-allocation of the memory required.
- When calling this function asynchronously the data will be returned in the event data (event: NSCASR_EVENT_GRAMMAR_ACTIVEGET) as described in Section 4.1.3.1 for function NSC_Event_Get(). In this case, the pEventData>pData>cData should be sent to NSC_GRAMMAR_LIST_ST* (see Section 4.2.6.7).

4.2.3.5 NSCASR_Grammar_VocabularyLoad

Description

This function loads a runtime vocabulary (see definition in NSC Speecher User's Guide) from the host memory to a grammar with runtime vocabulary definition. The grammar is ready for activation/recognition only after loading all the runtime vocabularies defined in a particular grammar.

Syntax

```
NSC_ERR_TP NSCASR_Grammar_VocabularyLoad
   (const NSC_HEADER_ST      *pHeader,
   short                     sResourceID,
   short                     sGrammarIdx,
   const char                *pstrVocabularyName,
   const NSC_DATA_ST         *pVocabularyData);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`sGrammarIdx`	Defines the grammar index (a unique number representing a grammar loaded on the resource).
`*pstrVocabularyName`	Defines the name of the runtime vocabulary defined in the grammar.
`*pVocabularyData`	Defines the pointer to an NSC_DATA_ST (see Section 4.1.7.1) structure containing the vocabulary data.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

NSC Speecher Only

Notes

- This function is used after loading a grammar that contains one or more runtime vocabularies. All vocabularies must be loaded before the grammar can be used for recognition or activation.
- After loading all runtime vocabularies it is possible to update or change any one of the runtime vocabularies, if necessary, by calling this function again.
- The runtime vocabulary data structure is created after some preprocessing is done using the NSCGrammar API. (Please refer to the *NSCGrammar User's Guide* and the *NSCGrammar API Guide*.)
- In case the same runtime vocabulary appears more than once in a given grammar, one call to this function with the vocabulary name will set all occurrences of this vocabulary in the grammar.
- In order to not cause a dialect mismatch in runtime vocabularies (defined during preparation stage), load vocabularies to grammars with the same dialect code.
- It is possible to load a run-time vocabulary with an empty list. However, in some specific grammars this might result in an invalid grammar (producing the error: NSCASR_VOC_CAUSE_OPEN_LOOP).

4.2.3.6 NSCASR_Grammar_Remove

Description

This function removes a previously loaded grammar from a specific resource memory.

Syntax

```
NSC_ERR_TP NSCASR_Grammar_Remove
   (const NSC_HEADER_ST *pHeader,
   short                sResourceID,
   short                sGrammarIdx);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`sGrammarIdx`	Defines the grammar index (a unique number representing a grammar loaded on the resource).

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9).

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

If an active grammar index is given, the grammar will be first deactivated and then removed.

4.2.4 Parameters Management Functions

4.2.4.1 NSCASR_Parameter_Set

Description

This function sets a list of parameters to a specific resource, overwriting the currently active (default) parameters.

Syntax

```
NSC_ERR_TP NSCASR_Parameter_Set
   (const    NSC_HEADER_ST  *pHeader,
   short                    sResourceID,
   NSC_PARAM_LIST_ST        *pParams);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`*pParams`	Defines the pointer to parameters list structure (see Section 4.1.7.8), which includes the parameters to be set.

Output

None

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous):

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- The parameter values set using this function are "sticky", i.e., they overwrite the currently active parameters for the resource as the resource is allocated.
- When calling NSC_Resource_Get() function, if the 'clear history' flag is set to NSC_NO then the resource is allocated with the same parameters that were last set by the NSCASR_Parameter_Set() function.
- If the 'clear history' flag is set to NSC_YES, the resource is allocated with the default parameters values as set by the server configuration tool (refer to NSCServer configuration manual).
- When calling NSCASR_Recognize() it is possible to send parameters that will be used in the current recognition task only. This will not overwrite the parameters set by the NSCASR_Parameter_Set() function or default parameters (not 'sticky').

4.2.4.2 NSCASR_Parameter_Get

Description

This function gets the currently active list of parameters on a specific resource.

Syntax

```
NSC_ERR_TP NSCASR_Parameter_Get
     (const  NSC_HEADER_ST        *pHeader,
     short                        sResourceID,
     NSC_PARAM_LIST_ST            *pParams);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`*pParams`	In synchronous mode, this refers to a pointer to an allocated NSC_PARAM_LIST_ST structure (see 4.1.7.8 and notes below). In asynchronous mode this pointer can be NULL (see notes below).

Output

`*pParams`	In synchronous mode, this will be a pointer to the list of currently active parameters (see notes below).

Returned Values

NSC_ERR_TP (see Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- When calling this function synchronously, the parameter list will be returned in the structure pointed to by pParams. The application will need to allocate the structure before calling this function and setting the estimated size pParams > usNumParameters. The error message NSC_MORE_DATA will be returned if the application did not allocate enough memory. In this case, the correct number of parameters is returned in pParams □usNumParameters. The application can call this function again after re-allocation of the memory required.
- When calling this function asynchronously, the data will be returned in the event data (see Section 4.1.3.1) for function NSC_Event_Get(). In this case the pEventData>pData>cData should be sent to NSC_PARAM_LIST_ST* (see Section 4.1.7.8).

4.2.5 Logging Functions

4.2.5.1 NSCASR_Log_WaveformPathSet

Description

This function overwrites the default directory in which audio and textual are saved during runtime for a specific resource.

Syntax

```
NSC_ERR_TP NSCASR_Log_WaveformPathSet
   (const NSC_HEADER_ST      *pHeader,
   short                     sResourceID,
   const char                *pstrWaveformPathName);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`*pstrWaveformPathName`	Defines the name of the path in which to save the audio log files.

Output

None

Returned Values

NSC_ERR_TP (see table in Section 4.2.9)

Include Files

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous)

Both

Relevant to (NSC Speecher/NSC Spotter/Both)

Both

Notes

- When this function is called, the log files (if enabled) are saved in the default directory as set during configuration of the server (please refer to NSCServer configuration manual).
- This function can be called once per resource.
- When a resource is allocated using NSC_Resource_Get(), if the 'Clear History' flag is set to NSC_NO the previously set path operation is valid. Otherwise the waveform log path is reset to the default settings.

4.2.5.2 NSCASR_Log_InfoSet

Description

This function sets the name of the audio logged file of the next recognition sessions and also sets application information to be added to the textual log file. It should be called before each recognition operation (before NSCASR_Recognize() is called).

Syntax

```
NSC_ERR_TP NSCASR_Log_InfoSet
   (const NSC_HEADER_ST      *pHeader,
   short                     sResourceID,
   const NSCASR_LOG_INFO_ST *pInfo,
   const char                *pstrWaveformFileName);
```

Input

`*pHeader`	Defines the pointer to the NSC Header Structure (see Section 4.1.7.3).
`sResourceID`	Defines the NSCServer resource ID.
`*pInfo`	Defines a pointer to a NSCASR_LOG_INFO_ST (see Section 4.2.6.6) that contains the information to be written to the textual log file.
`*pstrWaveformFileName`	Defines the name of the audio file in which the next recognition session will be saved.

Output

Null

Returned Values

NSC_ERR_TP (see table in Section 4.2.9)

Include Files:

nscasr.h, nscasr_err.h

Supported Modes (Synchronous/Asynchronous):

Both

Relevant to (NSC Speecher/NSC Spotter/Both):

Both

Notes:

- The function has two optional functionalities (at least one must be operated):
  - Enable extraction of textual log information (per recognition) with given application information.
  - Set the name of the logged audio file.
- If only *pinfo is set to NULL then no textual log file will be created for this recognition session.
- If only *pstrWaveformFileName is set to NULL the engine will save the audio logging of the session using thedefault naming convention, defined by NSC Speecher/Spotter (refer to the *NSC Speecher/Spotter User's Manual*).
- The extension type of the prompt (reference) PCM sample recording will be sru (u-law), sra (a-law), or srl (linear PCM).
- If both *pinfo and *pstrWaveformFileName are NULL the function will return with a NSC_WRONG_USAGE error (see Section 4.1.11).
- The extension type of the main PCM sample recording will be smu (u-law), sma (a-law), or sml (linear PCM).

4.2.6 NSC ASR API Structures

4.2.6.1 NSCASR_ACQ_PARAMS_ST

NSCASR_ ACQ_PARAMS_ST (Acquisition Parameters Structure)		
Member Name	**Type**	**Description**
NewAudioChannel	NSC_FLAG_TP	Indicates if this is a new audio channel (call)
StartTimers	NSC_FLAG_TP	Activation/Deactivation of timers (see Section 4.2.2.2)
SaveWaveform	NSC_FLAG_TP	Activation/Deactivation of audio PCM logs
usNoInputTimeout	unsigned short	'No Input Timeout'
usSpeechIncompleteTimeout	unsigned short	'Speech Incomplete Timeout'
usSpeechCompleteTimeout	unsigned short	'Speech Complete Timeout'
usRecognitionTimeout	unsigned short	'Recognition Timeout'
usECMode	unsigned short	Echo Cancellation operational mode
sSNRThreshold	Short	Signal to Noise Ratio threshold

This structure is used to pass acquisition parameters when calling NSCASR_Recognize().

4.2.6.2 NSCASR_RCG_PARAMS_ST

NSCASR_ RCG_PARAMS_ST (Recognition Parameters Structure)		
Member Name	**Type**	**Description**
AcqParams	NSCASR_ACQ_PARAMS_ST	Acquisition parameters (see Section 4.2.6.1)
usSpeedVsAccuracy	unsigned short	For future use TBD (0- 10)
usSensitivityLevel	unsigned short	Sensitivity level (0-100)
usConfidenceThreshold	unsigned short	Confidence threshold (0-100)
usNumBestAlternatives	unsigned short	Number of alternative phrases

This structure is used to pass recognition parameters when calling *NSCASR_Recognize()*.

4.2.6.3 NSCASR_RCG_ITEM_ST

NSCASR_ RCG_ITEM_ST (Recognized Item Structure)		
Member Name	**Type**	**Description**
pstrSlotName	char	Name of the slot of the recognized item.
pstrReportedText	char	Reported text for the recognized item. Please refer to the NSCGrammar User's Manual.
usConfidence	unsigned short	Confidence level of the recognized item (0- 100).
ulLocation	unsigned long	Location of the recognized item in the recognition input buffer from start of recognition, in units of frames.
ulDuration	unsigned long	Duration of the recognized item, in units of frames.

This structure represents a recognized item.

Note that each string in *pstrReportedText* and *pstrSlotName* is defined as a NULL terminated string.

4.2.6.4 NSCASR_RCG_PHRASE_ST

NSCASR_ RCG_ PHRASE _ST (Recognized Item Structure)		
Member Name	**Type**	**Description**
usConfidence	unsigned short	Confidence level of the whole phrase (0- 100).
usNumItems	unsigned short	Number of recognized items in the phrase.
Item[1]	NSCASR_RCG_ITEM_ST	Array of the recognized words (see Section 4.2.6.3)

This structure represents a recognized phrase consisting of one or more items.

4.2.6.5 NSCASR_RCG_RES_ST

NSCASR_ RCG_RES_ST (Recognition Results Structure)		
Member Name	**Type**	**Description**
ulSizeBytes	unsigned long	The size in bytes of the results data
ulWarnings	unsigned long	Includes the acquisition warnings (see Section 4.2.9)
usNumPhrases	unsigned short	Number of phrases in the recognition results
pstrWaveformURI	char*	The name of the logged waveform file
pPhrase[1]	NSCASR_RCG_PHRASE_ ST	Array of pointers to alternative phrases in descending order

This structure is used to report the recognition results when NSCASR_Recognize_ResultsGet() is called.

4.2.6.6 NSCASR_LOG_INFO_ST

NSCASR_ LOG_INFO_ST (Log Information Structure)		
Member Name	**Type**	**Description**
strInfo	Char[64]	Free text for application information to add to textual log file
sApplScriptId	unsigned short	Application Script ID
sApplStateId	unsigned short	Application State ID

This structure is used by the application to pass information to the textual recognition logs when NSCASR_Log_InfoSet() is called.

4.2.6.7 NSCASR_GRAMMAR_LIST_ST

NSCASR_ LOG_INFO_ST (Log Information Structure)		
Member Name	**Type**	**Description**
sCount	short	Number of grammars in the list
sIndex	short[1]	Array of the grammar indexes

This structure represents a list of grammars (using their unique grammar indexes).

4.2.7 NSC ASR API Event Codes

Name	Value	Description
Event Codes		
Recognition events		
NSCASR_EVENT_RECOGNIZE	401	Event retuned at the end of recognition
NSCASR_EVENT_RECOGNIZE_TIMERSSTART	402	
NSC ASR generated events		
NSCASR_EVENT_SPEECH_DETECTED	1101	Speech was detected on the channel during the recognition
NSCASR_EVENT_CONTENT_SPOTTED	1102	Relevant speech has been detected on the channel
Grammar Events		
NSCASR_EVENT_GRAMMAR_ACTIVATE	403	Event returned after NSCASR_Grammar_Activate() finished running asynchronously
NSCASR_EVENT_GRAMMAR_DEACTIVATE	404	Event returned after NSCASR_Grammar_DeActivate() finished running asynchronously
NSCASR_EVENT_GRAMMAR_ACTIVEGET	405	Event returned after NSCASR_Grammar_ActiveGet() finished running asynchronously
NSCASR_EVENT_GRAMMAR_LOAD	406	Event returned after NSCASR_Grammar_Load() finished running asynchronously

Name	Value	Description
NSCASR_EVENT_GRAMMAR_VOCABULARYLOAD	407	Event returned after NSCASR_Grammar_VocabularyLoad() finished running asynchronously
NSCASR_EVENT_GRAMMAR_REMOVE	408	Event returned after NSCASR_Grammar_Remove() finished running asynchronously
Logging Events		
NSCASR_EVENT_LOG_INFOSET	409	Event returned after NSCASR_Log_InfoSet() finished running asynchronously
NSCASR_EVENT_LOG_WAVEFORMPATHSET	410	Event returned after NSCASR_Log_WaveformPathSet() finished running asynchronously
Parameter Management Events		
NSCASR_EVENT_PARAMETER_SET	411	Event returned after NSCASR_Parameter_Set() finished running asynchronously
NSCASR_EVENT_PARAMETER_GET	412	Event returned after NSCASR_Parameter_Get() finished running asynchronously

4.2.8 NSC ASR API Constants

Name	Value	Description
General Definitions		
NSCASR_ACTIVE_GRAMMAR_INDEX	20100	Index for recognition with activated grammars
Parameters Definitions		
NSCASR_PARAM_NEWAUDIOCHANNEL	401	Indicates a new audio channel (call)
NSCASR_PARAM_STARTTIMERS	402	Allows the activation of timers during recognition. (see Section 4.2.2.2)
NSCASR_PARAM_NOINPUTTIMEOUT	403	No input time out
NSCASR_PARAM_SPEECHINCOMPLETETIMEOUT	404	Speech incomplete time out
NSCASR_PARAM_SPEECHCOMPLETETIMEOUT	405	Speech complete time out
NSCASR_PARAM_RECOGNITIONTIMEOUT	406	Recognition time out
NSCASR_PARAM_ECMODE	407	Echo cancellation activation/deactivation
NSCASR_PARAM_SNRTHRESHOLD	408	Signal to Noise ratio threshold
NSCASR_PARAM_SAVEWAVEFORM	409	Activate/Deactivate audio recording logs
NSCASR_PARAM_SPEEDVSACCURACY	410	Speed Vs. Accuracy – not active in current version
NSCASR_PARAM_SENSITIVITYLEVEL	411	Sensitivity level of the recognition
NSCASR_PARAM_CONFIDENCETHRESHOLD	412	Confidence threshold
NSCASR_PARAM_NUMBESTALTERNATIVES	413	Amount of alternative phrases
NSCASR_PARAM_OPERATIONMODE	413	Special operational modes

4.2.9 NSC ASR API Error Codes

NSC_ERR_TP (Function Error Return Codes – As defined in Nsc_err.h)		
Name	**Value**	**Description**
All values are NSCASR_ERROR_OFFSET(1400) + the value stated		
NSCASR_GRM_LIMIT	1	Number of grammars exceed the maximum allowed
NSCASR_GRM_INVL	2	Invalid grammar
NSCASR_GRM_IDX_INVL	3	Invalid grammar index was provided
NSCASR_GRM_IDX_IN_USE	4	The grammar index provided is already in use
NSCASR_GRM_TOO_LONG	5	Grammar exceeds the maximal speech duration set in the current recognition session. For example: Known length 20-digits grammar has a limitation of 2 seconds.
NSCASR_VOC_INVL	101	Attempt to load a vocabulary of different language than the loaded grammar.
NSCASR_VOC_NOT_SET	102	One or more vocabulary lists are not set
NSCASR_VOC_NAME_MISMATCH	103	Vocabulary name does not match any name in the grammar
NSCASR_VOC_CAUSE_OPEN_LOOP	104	Vocabulary with no words (causes an open loop in the grammar)
NSC Acquisition warnings (bit location)		
NSCASR_LOW_SNR	0x1L	SNR is below SNR threshold
NSCASR_NO_SPEECH	0x2L	No speech was detected
NSCASR_TOO_LOUD	0x4L	Detected speech too loud at acquisition
NSCASR_TOO_SOFT	0x8L	Detected speech too soft at acquisition
NSCASR_TOO_LONG	0x10L	Detected speech too long

NSC_ERR_TP (Function Error Return Codes – As defined in Nsc_err.h)		
Name	**Value**	**Description**
NSCASR_TOO_EARLY	0x20L	Detected speech starts too early
NSCASR_TOO_SHORT	0x40L	Detected speech too short for recognition using the selected grammar
NSCASR_NO_MATCH	0x80L	Detected speech too short for recognition using the selected grammar
Log waveform warnings (bit location)		
NSCASR_LOG_WAVEFORM_CORRUPT	0x100L	Waveform data has been overwritten

This page is intentionally left blank.

A References

- *NSC Speecher/Spotter User’s Guide* - Overview of the NSC engines: environment, concepts and usage.
- *NSCGrammar API Manual* - Definition of the Application Programming Interface to the NSCGrammar (Refer to this manual for details of run-time vocabulary functions).

International Headquarters
1 Hayarden Street,
Airport City
Lod 7019900, Israel
Tel: +972-3-976-4000
Fax: +972-3-976-4040

AudioCodes Inc.
27 World's Fair Drive,
Somerset, NJ 08873
Tel: +1-732-469-0880
Fax: +1-732-469-2298

Contact us: https://www.audiocodes.com/corporate/offices-worldwide
Website: https://www.audiocodes.com

Document #: LTRT-13180