



ASR1600/C V2 Low Level API Specification for Sega's DreamCast platform

1 Table of Contents

ASR1600/C Low Level API Specification for Sega's DreamCast platform.....	1	5.2.2 Context related.....	54
1 Table of Contents.....	2	CasrExportContext	54
2 Introduction	4	5.2.3 Classes related.....	55
2.1 Basic Functionality	4	CasrCloseClasses	55
2.2 Basic Requirements.....	4	CasrImportClasses	56
3 Paradigm	5	5.2.4 Merging of contexts and classes.....	57
3.1 Definitions	5	CasrMergeContextsAndClasses	57
3.2 API Internals	6	6 Callbacks	59
3.3 Initializing the engine.....	6	CBABNORM	59
3.4 Context manipulation	7	CBAGC	60
3.5 Classes	7	CBASKCURRENTGAIN	61
3.6 Recognition	7	CBRESULT	62
3.7 Inheritance of adaptive data	9	CBSTATE	63
3.8 Automatic Gain Control.....	9	CBTRAIN	64
3.9 Memory	9	7 OS Callbacks.....	65
3.10 User word training.....	9	CBCREATECRITICALSECTION	65
3.11 Spelling.....	10	CBDELETECRITICALSECTION	66
3.12 Adding and deleting of words	10	CBENTERCRITICALSECTION	67
3.13 Adding and deleting of user words	10	CBFREE	68
3.14 Exported Symbol activation.....	11	CBGETCURTASK.....	69
3.15 Activation of more than 1 context at the same time.....	11	CBGETCURTHREAD.....	70
3.16 Recognition Operating Mode and State diagram 11		CBLEAVECRITICALSECTION.....	71
4 ASR1600 Low Level API function calls ...	13	CBMALLOC	72
4.1 Engine related functions.....	13	CBREALLOC	73
4.2 Context related functions	14	8 Types, structures and defines.....	74
4.3 Classes related functions	14	8.1 Types	74
4.4 Training related functions	15	BOOL	74
4.5 Engine state related functions.....	15	CASRLEVEL	74
4.6 Miscellaneous functions	15	CPARAMID	74
5 ASR1600 Function Specification.....	16	CPARAMVALUE	76
5.1 Basic functionality	16	CWORDID	76
CasrAcquisition.....	16	DWORD	76
CasrActivateContext	18	ERRORID.....	76
CasrActivateData	20	HASR	77
CasrAPIClose	22	HCONT	77
CasrAPIInit	23	HCLASSES	77
CasrClose	24	HCRITSECTION	77
CasrCloseContext.....	25	HDATA	77
CasrCloseData	26	PALTERNATIVE	78
CasrExportData	27	PCASRLEVEL.....	78
CasrGainSet	28	PCASRRESULT	78
CasrGetActivatedContext	29	PCASRSTATE	78
CasrGetActivatedData	30	PDATA	78
CasrGetActiveWords	31	PCLASSES	78
CasrGetAPIVersion	32	PCOSCALLBACKS	78
CasrGetParam	33	PCPARAMLIST	78
CasrGetParamList	34	PCPARAMVALUE	78
CasrGetSignalLevel.....	35	PCRECOGCALLBACKS	78
CasrGetSNR.....	36	PCONT	78
CasrGetSPIVersion	37	PCWORDID	78
CasrGetState.....	38	PDWORD	78
CasrImportContext.....	39	PHASR.....	78
CasrImportData	40	PHCONT	78
CasrOpen	41	PHCLASSES.....	78
CasrSetActiveWords	42	PHDATA	79
CasrSetParam	44	PRECWORD	79
CasrSetParamList	45	PSSENTENCE	79
CasrStart	46	PSPEECHUNITBUF.....	79
CasrStop	47	PTIMEINFO.....	79
5.2 Added functionality.....	48	PUSERWORDBUF	79
5.2.1 User word training related	48	PVOID	79
CasrAcknowledge	48	PWORD	79
CasrAddContextUserWord	49	PWORDBUF	79
CasrDeleteContextUserWord.....	51	PUSERDATA	79
CasrStartUserWordTraining.....	52	USERWORDBUF.....	79
		WORD.....	79
		8.2 Structures	80
		ALTERNATIVE.....	80

CASRRESULT	80	TIMEINFO	83
COSCALLBACKS	80	USERDATA	84
CPARAMLIST	81	WORDBUF	84
CRECOGCALLBACKS	82	8.3 Enumeration types	85
DATA	82	ABNORMCOND	85
RECWORD	83	CASRSTATE	85
SENTENCE	83	CSAMPLEFORMAT	86
SPEECHUNITBUF	83	PROMPTYPE	86

2 Introduction

The ASR1600/C API is designed to be a portable speech API for consumer products that need a phonetic engine. A port of the ASR1600 engine to a consumer product can be started from this specification. The API encapsulates the SPI (Service Provider Interface) and adds some functionality to it. The SPI is an interface, which contains only the basic engine functionality needed. The API adds to this functionality the addition and deletion of user words, the possibility to export contexts and the concept of classes¹.

The following list enumerates the functionality that is implemented, and the minimum requirements of the device and operating system that must be available.

2.1 Basic Functionality

- Only 1 context can be active at the same time on an engine. If a higher level API or an application wants to activate more than 1 context at the same time, those contexts must be merged to 1 context before sending it to the engine.
- >1 engine can be started
- Language switching is available
- 1 task or thread controls 1 engine.
- Fast activation of words is available. If an application or a higher level API wants symbol¹ activation, this API has to do this with word activation.
- User Word training at runtime is possible.
- Trained user words can be shared among different contexts and can be stored by the user of the API to be able to add them to different contexts.
- N-Best alternative recognition results will be returned.
- Confidence levels are returned with each recognition result alternative.
- Spelling is not available.

2.2 Basic Requirements

- No operating system calls are called directly in this API. All the OS functions needed are encapsulated in a set of callback functions the user of the API has to supply.
- The API does not do signal acquisition. The API is not concerned with input devices, the samples are supplied by the user of the API.
- The implementation of the API doesn't use global variables that can change so that the linker-generated file can be put in ROM.
- All callback functions are passing application-defined user data. This allows applications to do their own sub-allocation in specific heap or to keep data attached to the API and/or engine sessions.

¹ For more information about concepts of ASR, please refer to *The Speech Application Development Guide*.

3 Paradigm

3.1 Definitions

API

Application Programmer Interface. Layer of function calls that interface between an application or a higher level API and a recognition engine. It provides services that can be used by a client. In this document API refers to the ASR1600 Low Level consumer API also written as ASR1600/C.

SPI

Service provider interface. Layer of function calls that expose the basic recognition functionality. It provides the service (a recognizer) that can be used by a client (application or a higher level API).

Callback function

Function of the application that is called by the API. In an initializing the API, a pointer to this application function is passed to the API. The API can then call this function when it needs, e.g. to request more memory, or to inform that a new recognition result is available.

Grammar

Textual description of all sentences (syntax + vocabulary) that need to be recognized.

Context

Compiled grammar (in an engine-useable format).

Engine

Code that actually does the recognition.

User Word

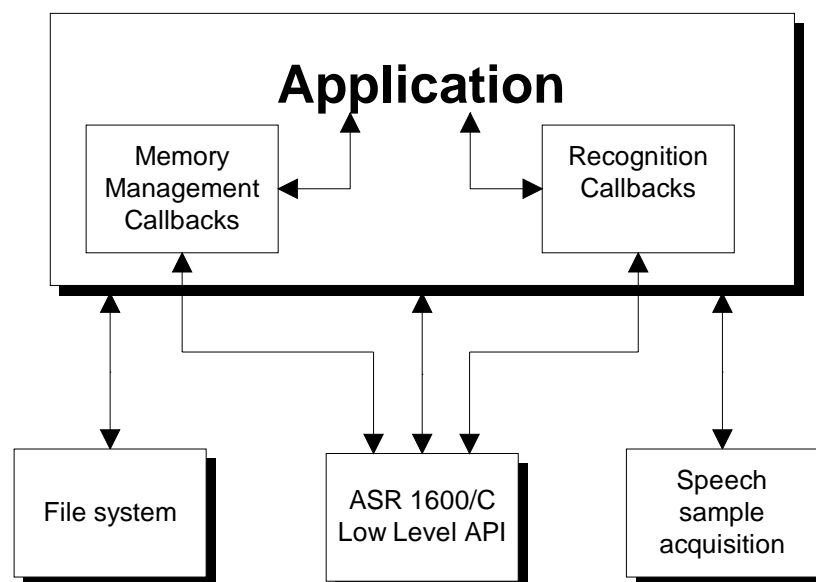
Word that can be added to a class, of which the pronunciation is obtained by having the user speak the word a number of times (the so-called "training procedure")

3.2 API Internals

The ASR 1600/C Low Level API is designed to be a portable speech API for consumer products that needs a phonetic recognizer. The API does not have support for advanced context and application management. This means in practice that the API is not able to associate contexts with different applications and that recognition results will be sent only to one application.

Portability of the ASR1600 Low Level API is ensured due to the fact that the API

- does not make use of a file system: all data (contexts, language data and user models) are received and returned as data-buffers. This way all data is independent of the storage format, plus the controlling application does not need to know more about the data than its size, and location in memory.
- does not do its own memory management (does not directly allocate and de-allocate memory).
- calls the operating system using a limited set of basic (encapsulated) functions, which should be available on any operating system. All multitasking and multithreading functions defined can be replaced by dummy functions in operating systems that do not support multitasking and/or multithreading. These functions are only used to make the API task and thread safe.
- doesn't use global variables that can change (only constant ones).



3.3 Initializing the engine

This API does not have the concepts user and language. All data the recognizer needs can be imported with `CasrImportData` and activated on an engine with `CasrActivateData`.

The data buffers given to `CasrImportData` can be permanent (for example in ROM), non-permanent or changeable by the engine. In the case of permanent non-changeable data the engine makes no copy but only a reference to the data is kept. In the case of changeable data, this data can be saved with the `CasrExportData` function.

3.4 Context manipulation

Each context is stored outside the API in a separate buffer. This buffer can be imported with the `CasrImportContext` function. Before recognition can be started, the context has to be activated on the engine (`CasrActivateContext`). The same context can be activated on different engines.

Note that context buffers used to store contexts are address independent, and thus can be moved around in memory, saved, reloaded etc.

The context can be saved for later reuse with the `CasrExportContext` function.

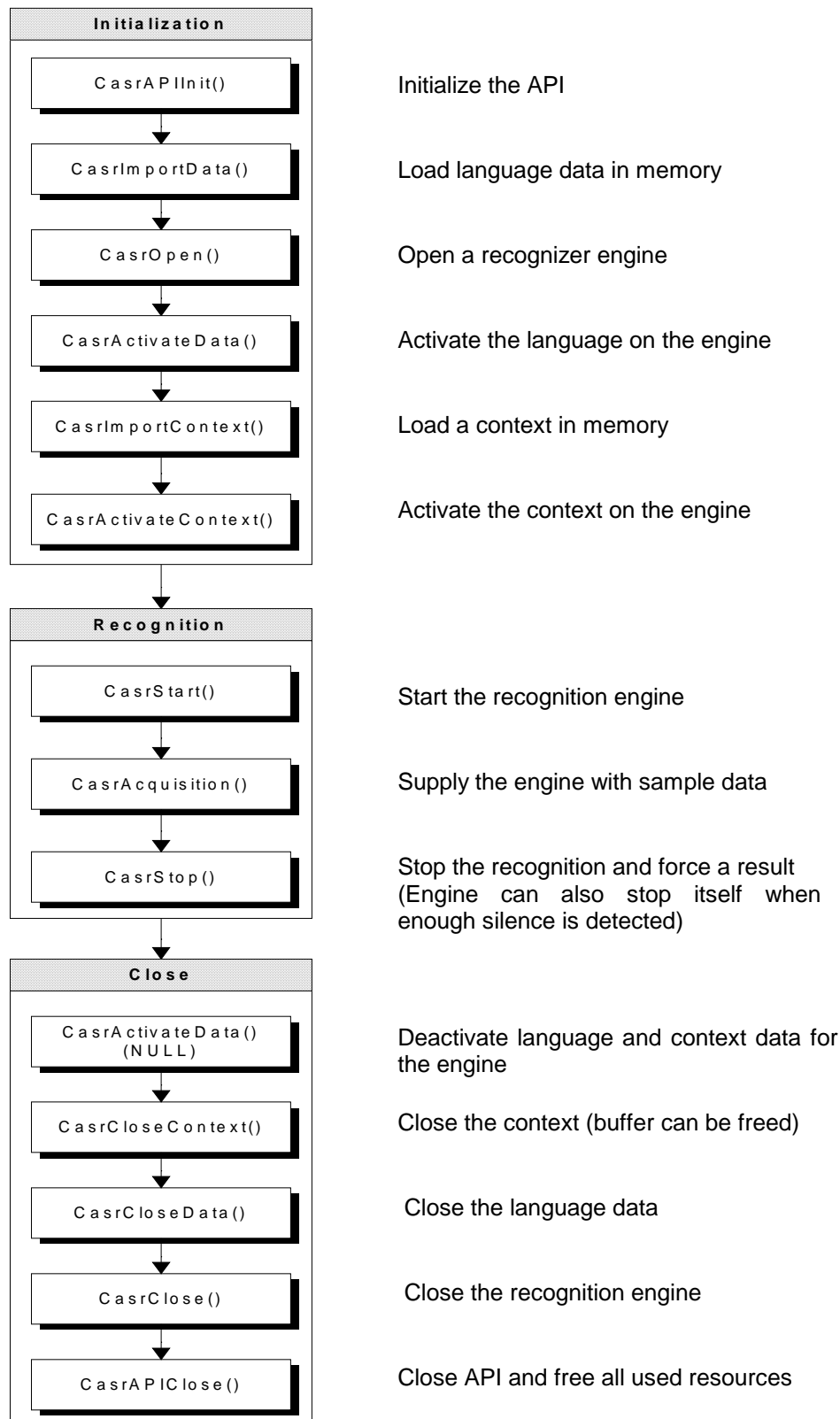
3.5 Classes

The class names data buffer is stored outside the API. This buffer can be obtained from the PC tools. This buffer can be imported with the `CasrImportClasses` context.

3.6 Recognition

After the recognition is started up, the user of the API supplies the API with buffers containing wave data. These buffers are supplied by calling the `CasrAcquisition` function. When the engine has a recognition result ready, it calls the `CBRESULT` callback function. Recognition is started with the `CasrStart` function. This function returns directly without waiting for a recognition result.

Basic Sequence of Function Calls for Recognition



3.7 Inheritance of adaptive data

Adaptive data refers to engine settings based on line characteristics and speaker characteristics. These characteristics have a default value when a recognizer is started and are automatically recalculated (and adapted) on the fly based on the engines input signal.

There is a possibility to initialize adaptive data of the ASR1600 data. This can be accomplished by specifying the initialization data with the CasrActivateData function. The following sources can be used:

1. Platform specific. The data has been optimized to a certain hardware platform.
2. Data generated with a microphone adaptation procedure if supported for this platform. This could be a dummy recognition session where the user is asked to count. At the end the CasrExportData function gives the initialization data.
3. Data from the previous recognition session (Data that has been saved with the CasrExportData function).
4. Default (Factory setting). This is done if no CasrActivateData is called before CasrStart.

The data used for CasrActivateData should preferably be the first in this list that is available.

3.8 Automatic Gain Control

The ASR1600 engine contains an Automatic Gain Control that can be used to automatically adjust the analog gain. This feature will only work if the platform supports changing of the analog gain. This feature can be enabled with the CPARAM_AGCON parameter.

The engine will call the CBAGC callback function each time it wants to do an adjustment of the gain supplying a new value for the gain. It is up to the application if it wants to adjust the gain to this value. The new value of the gain will be notified to the API with the CasrGainSet function.

At the start of the engine the CBASKCURRENTGAIN callback is used to initialize the gain value on the engine.

3.9 Memory

The API does not have internal memory management. This implies that all memory for use of the API must be given to the API by an application. This does not only include buffers containing contexts and sample buffers, but even buffers containing the API workspace and intermediate results.

The callback functions CBMALLOC and CBFREE are used for allocating and freeing memory.

3.10 User word training

A user word is trained independently of the contexts. When the user word training is done, the result is a buffer. This buffer can then be used for adding the word to a context (CasrAddContextUserWord) or saved by the application for later use. Note that this buffer can be used many times, or can even be stored and reloaded later.

Training starts by calling CasrStartUserWordTraining. The engine calls the CBTRAIN (PROMPTTYPE_START) callback function. Upon acknowledge (CasrAcknowledge) of this callback function, the engine expects a training utterance. At the end of the utterance (TS detected or CasrStop called) the CBTRAIN (PROMPTTYPE_ACCEPT) callback function is called to ask for an acceptance of this utterance. On acceptance of the utterance the next utterance is asked or the CBTRAIN (PROMPTTYPE_CONFIRM) callback function is called if it was the last utterance that is being accepted. When the acknowledgment of this callback function is PROMPT_OK, the user word model is calculated. At the end the CBTRAIN (PROMPTTYPE_TRAINEND) callback function is called. After this event the user word buffer returned, is valid.

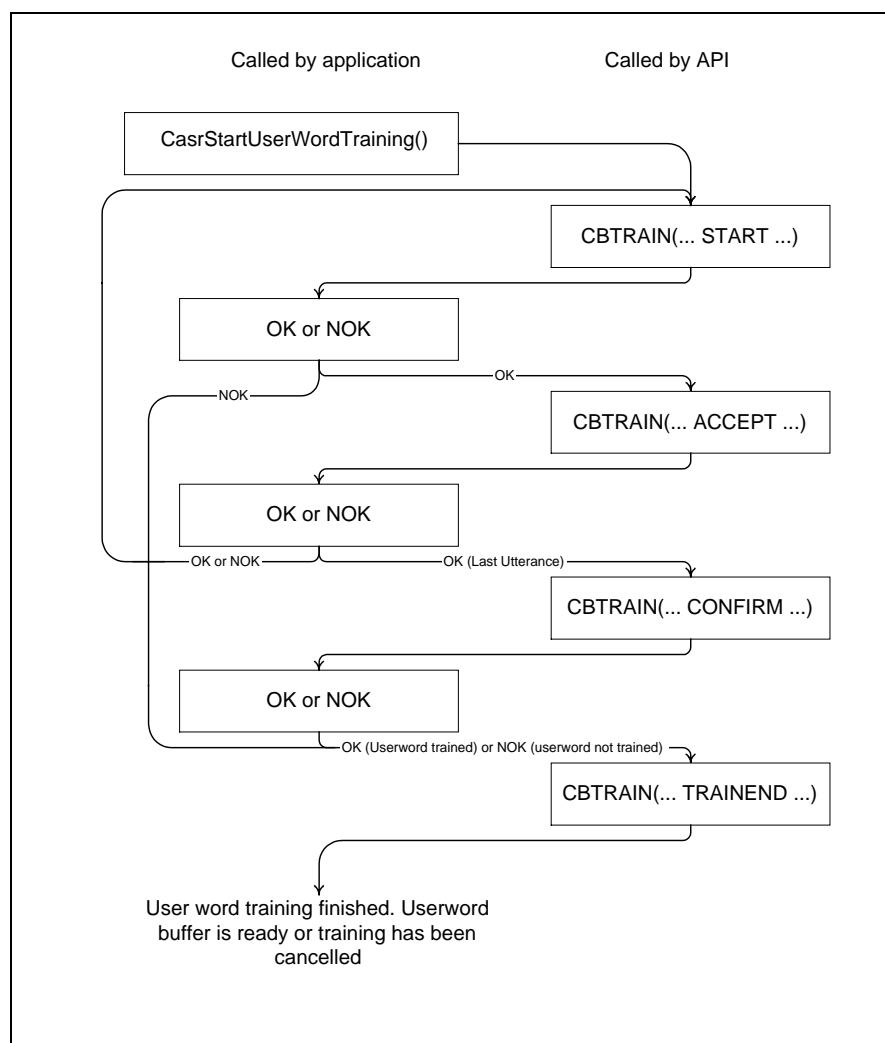


Figure 1: user word training call flow

3.11 Spelling

Spelling is not included.

3.12 Adding and deleting of words

This API does not support adding and deleting of words from a context. This means that only userwords can be added dynamically, all speaker independent words need to be known beforehand.

3.13 Adding and deleting of user words

This API supports adding and deleting of user words from a context. This can be done with `CasrAddContextUserWord` and `CasrDeleteContextUserWord`. The new context can be exported with `CasrExportContext`.

3.14 Exported Symbol activation

This API does not support activation and deactivation of symbols that are exported in a BNF grammar. This functionality can be implemented on top of the API with the `CasrSetActiveWords` and `CasrGetActiveWords` functions.

3.15 Activation of more than 1 context at the same time

Only one context can be activated on the engine itself. If a higher level API or an application wants to activate more than 1 context at the same time, the contexts must be merged to one context with the `CasrMergeContextsAndClasses` function.

3.16 Recognition Operating Mode and State diagram

The ASR1600 engine has one operating mode. An operating mode can be represented by a finite state diagram describing the possible states and state transitions. State transitions in this diagram can happen due to API function calls or internal events in the recognition engine.

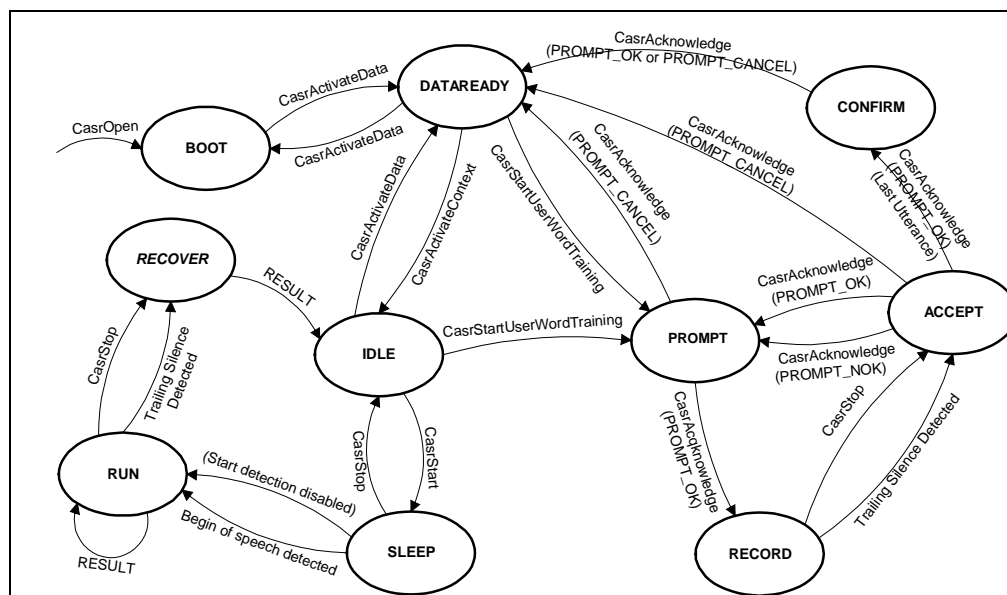


Figure 2: Engine state diagram

The operating mode can be represented by a total of 10 states:

BOOT : Initial state.

DATAREADY : All necessary data has been activated on the engine so that context can be activated or user word training can start.

IDLE : All necessary data and context have been activated so that recognition can start.

SLEEP : Low processing begin of speech detection active. When start detection is disabled the state automatically changes to the RUN state.

RUN : Recognition is active, full processing power is used.

RECOVER : Recognition not active, recovering recognition result.

PROMPT : Asking for a confirmation to start the next utterance.

RECORD : Acquisition active.

ACCEPT : Asking for an acceptance of an utterance.

CONFIRM : Asking for a confirmation to accept the 3 utterances.

The engine is opened in the boot state. After all necessary data has been activated (CasrActivateData) on the engine, a transition to the *DATAREADY* state is made. From

this state a context can be activated (to the *IDLE* state) to do recognition or user word training can be started (to the *PROMPT* state).

The CasrStart function will force a state transition from the *IDLE* state to the *SLEEP* state. Depending on the CPARAM_START_ENABLE parameter the engine stays in the *SLEEP* state for detecting the beginning of speech or goes directly to the *RUN* state. On CasrStop or on detection of trailing silence (if enabled) a transition is made to the *RECOVER* state. In this state the recognition result is determined and the result is notified to the application. Afterwards a transition is made to the *IDLE* state.

The user word procedure is started from the *DATAREADY* or *IDLE* state with the CasrStartUserwordTraining API function. A transition is made to the *PROMPT* state. In this state the application is notified when the user can start speaking. The application has to acknowledge (CasrAcknowledge) this notification to proceed with the training. When the application acknowledges a *PROMPT* notification with *PROMPT_OK* this indicates that the training procedure can continue and a transition is made to the *RECORD* state. At this point the acquisition process is started and the user is supposed to say the user word. Detection of trailing silence and/or CasrStop will cause a transition to the *ACCEPT* state that will wait for confirmation to continue. At this point the user can accept or reject the utterance that was spoken during the *RUN* state. After acceptance or rejection a transition is made to the *PROMPT* state. If the user has rejected the utterance, he will be prompted to pronounce the user word again. The speech information will be used to train the user word when he accepts the utterance and he will be prompted for the next utterance. When this was the last utterance, a transition is made to the *CONFIRM* state and the user will be asked if he wants to accept the utterances and create the user word. A transition is made to the *DATAREADY* state which finishes the training procedure. The end of the training is notified with the *PROMPTTYPE_TRAINEND* notification. From now on the user word buffer is valid.

The CPARAM_TS parameter indicates to the recognizer how much trailing silence has to be heard before it can be decided that the utterance is complete. This is not an internal constant in the engine because this value depends on the specific application. Typical command and control applications making use of isolated word recognition or keyword-spotting syntax requires typically 300 ms of trailing silence. Applications using continuous digit syntax or general continuous speech grammars will need 500-1000 ms of trailing silence before the endpoint criterion fires. This is due to the fact that in continuous speech small pauses and hesitations occur in the middle of a sentence.

The CPARAM_TIMEOUT parameter indicates the maximum time the recognizer will search for trailing silence. Bad environmental conditions, excessive background noise, competing signals can be the cause that the stop criterion never fires. In those cases one needs an emergency brake to stop the recognizer after a reasonable amount of time.

The CPARAM_ENABLEPREMRES parameter is used to enable the generation of premature result notifications. This is useful in a keyword spotting or dictation like syntax. Once the engine is started a result is notified when there is one available (e.g.: a keyword detected) and the engine stays in the *RUN* state. This process is stopped when CasrStop is called. When this feature is enabled, the CPARAM_START_ENABLE parameter indicates if during silence the low CPU speech detection system is used. When this parameter is enabled the engine is only recognizing during periods of speech. In this case the engine state will be switching between *SLEEP* and *RUN*.

4 ASR1600 Low Level API function calls

4.1 Engine related functions

CasrAcquisition

Supplies the engine with sample data. This function should be called periodically.

CasrActivateData

Activates certain data on an engine.

CasrClose

Closes an engine.

CasrCloseData

Closes data.

CasrExportData

Exports data to a buffer in case the engine has changed the data.

CasrGainSet

Passes the analog gain setting to the API.

CasrGetActivatedData

Returns the data that has been activated on an engine.

CasrGetSignalLevel

Gets the signal level from the last sample buffer.

CasrGetSNR

Gets the current Signal to Noise Rate of the signal seen so far.

CasrImportData

Loads data in memory passes a memory pointer to the permanent data.

CasrOpen

Opens an engine (also initializes certain callback functions).

CasrStart

Starts a recognition engine.

CasrStop

Stops a recognition and forces a result.

4.2 Context related functions

CasrActivateContext

Activates a context on an engine disabling the previous context.

CasrAddContextUserWord

Adds a user word to a context.

CasrCloseContext

Closes a context.

CasrDeleteContextUserWord

Removes a user word from a context.

CasrExportContext

Exports a context to a buffer.

CasrGetActivatedContext

Gets the activated context.

CasrGetActiveWords

Gets the enabled words in 1 class in a context.

CasrImportContext

Loads a context in memory and associates a handle with it.

CasrMergeContextsAndClasses

This function merges two context buffers and their corresponding class buffers.

CasrSetActiveWords

Enables certain words in 1 class in a context.

4.3 Classes related functions

CasrCloseClasses

Closes a class names instance.

CasrImportClasses

Imports a class names data buffer.

4.4 Training related functions

CasrAcknowledge

Sends an acknowledge signal back to the engine when a user word utterance is accepted.

CasrStartUserWordTraining

Initializes and starts user words training.

4.5 Engine state related functions

CasrGetParam

Gets the value of a parameter.

CasrGetParamList

Gets the value of all the engine specific parameters.

CasrGetState

Gets the current state of the engine.

CasrSetParam

Sets the value of a parameter.

CasrSetParamList

Sets the value of all the engine specific parameters.

4.6 Miscellaneous functions

CasrAPIClose

Frees all used resources.

CasrAPIInit

Initializes the API.

CasrGetAPIVersion

Gets the API version.

CasrGetSPIVersion

Gets the SPI version.

5 ASR1600 Function Specification

5.1 Basic functionality

CasrAcquisition

ERRORID CasrAcquisition(HASR *hAsr*, CSAMPLEFORMAT *SampleFormat*, PVOID *pSampleBuf*, DWORD *dwBufSize*)

Receives a sample buffer from the application and recognition processing is done. Notification callback functions (state changes, recognition results) are likely to be called within this function call.

Parameters

hAsr

Handle to an engine installed opened with **CasrOpen** [INPUT].

SampleFormat

Format of the samples in *pSampleBuf* [INPUT].

The only current valid format is PCM_16_11KHZ.

pSampleBuf

Buffer containing the samples [INPUT].

dwBufSize

Length (in bytes) of the sample buffer [INPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function is called on an engine that is in the wrong state. Valid states are CASR_RUN, CASR_RECORD, CASR_SLEEP.
ERR_BADSAMFORMAT	The format of the samples is not valid.
ERR_NULL_HANDLE	A NULL handle was passed
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to an engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

Callback functions within this function call

All recognition callback functions can be called from within this function depending on the engine state.

See Also

CasrOpen(),CasrStart()

CasrActivateContext

ERRORID CasrActivateContext(HASR *hAsr*, HCONT *hCont*)

Activates a context on an engine, enabling all the words in the context. After this function call, recognition can be started.

Parameters

hAsr

Handle to the engine instance [INPUT].

hCont

Context handle returned by the **CasrImportContext** function. If *hCont* equals NULL the active context is de-activated [INPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function is called on an engine that is in the wrong state. Valid states are CASR_IDLE, CASR_DATAREADY.
ERR_NULL_HANDLE	A NULL handle was passed
ERR_INVALID_CONTEXT	Handle to the imported context is not valid, probably because it has been closed.
ERR_TASK_THREAD_CONTEXT	Handle to the context was created in another task or thread by CasrImportContext .
ERR_MALLOC	Memory allocation failed. The callback functions CBMALLOC and CBFREE are called for storing and release the engine's dynamic data. Probably not enough heap memory.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to an engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

Callback functions within this function call

CBSTATE

Callback function used to notify state changes.

See Also

CasrImportContext(), CasrGetActivatedContext(), CasrSetActiveWords(),
CasrGetActiveWords()

CasrActivateData

ERRORID CasrActivateData(HASR *hAsr*, HDATA *hData*)

Activates certain data to be used by an engine. For the 1600 this is either a language buffer or an session data buffer. If a context was already activated on the engine, an activation of a language buffer automatically deactivates that context! If *hData* equals 0, the current language, if present, is deactivated, together with its context, if present. The engine is put back in the BOOT state.

Parameters

hAsr

Handle to the engine instance [INPUT].

hData

Handle of the data returned by the **CasrImportData** function [INPUT].

Return Value

Error code.

ERR_WRONG_STATE	The engine function is called on an engine that is in the wrong state. Valid states are CASR_IDLE, CASR_DATAREADY, CASR_BOOT.
ERR_NULL_HANDLE	A NULL handle was passed
ERR_INVALID_CONTEXT	Handle to the imported context is not valid, probably because it has been closed.
ERR_TASK_THREAD_CONTEXT	Handle to a context is used in another task or thread than the one that did the CasrImportContext . The callback function CBGETCURTASK is called to examine this.
ERR_MALLOC	Memory allocation failed. The callback functions CBMALLOC and CBFREE are called for storing and release the engine's dynamic data. Probably not enough heap memory.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

Callback functions within this function call

CBSTATE

Callback function used to notify state changes.

See Also

CasrImportData(), CasrGetActivatedData(), CasrCloseData(), CasrExportData()

CasrAPIClose

ERRORID CasrAPIClose(HAPI *hApi*)

Frees all resources used by the API. The *hApi* handle is invalidated by this function. When this function has been called, no other API functions may be called anymore except the CasrAPIInit function.

Parameters

hApi
Handle to the API instance [INPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed already.
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrAPIInit()

CasrAPIInit

ERRORID CasrAPIInit(PCOSCALLBACKS *pOsCallbacks*, DWORD *dwApiUserData*, PHAPI *phApi*)

Initializes the OS callbacks used by the API. These are the only operating system calls called in the API. The API handle is the only handle that can be used across different threads.

Parameters

pOsCallbacks

Pointer to the COSCALLBACKS structure containing all the OS callbacks. Structure will be copied and can therefor reside on the stack [INPUT].

dwApiUserData

DWORD representing a value that will be passed by all callback functions called by the API [INPUT].

phApi

Address where the handle of an API instance will be written [OUTPUT].

Return Value

Error code.

ERR_NULL_HANDLE

A NULL handle was passed.

ERR_MALLOC

Memory allocation failed.

The callback functions CBALLOC and CBFREE are called for storing and release userdata into memory.

Probably not enough heap memory.

See Also

CasrAPIClose(), CasrOpen()

CasrClose

ERRORID CasrClose(HASR *hAsr*)

Closes an engine, invalidates the *hAsr*. Frees all used resources belonging to that engine. The engine must be in the BOOT, IDLE or DATAREADY state. This function invalidates the *hAsr* handle.

Parameters

hAsr
Handle to the engine instance [INPUT].

Return Value

Error code.

ERR_WRONG_STATE	An engine function is called on an engine that was in the wrong state. Valid states are CASR_IDLE, CASR_DATAREADY, CASR_BOOT.
ERR_NULL_HANDLE	A NULL handle was passed
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrOpen()

CasrCloseContext

ERRORID CasrCloseContext(HCONT *hCont*)

Closes a context, invalidates the *hCont* handle. Frees all used resources. The hCtx may not be active on any engine!

Parameters

hCont
Handle to the context [INPUT].

Return Value

Error code.

ERR_CONTEXT_IN_USE	A CasrCloseContext was attempted on a context that is still in use on some engine.
ERR_NULL_HANDLE	A NULL handle was passed to the API function.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrImportContext(), CasrCloseData(), CasrActivateContext()

CasrCloseData

ERRORID CasrCloseData(HDATA *hDATA*)

Closes the data, invalidates the *hData*. Frees all used resources. The *hData* must not be active on any engine!

Parameters

hDATA

Handle to the data [INPUT].

Return Value

Error code.

ERR_DATA_IN_USE	A CasrCloseData was attempted on imported data that is still in use on some engine.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrCloseContext(), CasrActivateData(), CasrImportData(), CasrExportData()

CasrExportData

ERRORID CasrExportData(HASR *hAsr*, DWORD *DataType*, PVOID **ppBuf*, PDWORD *pdwSize*)

Exports the data associated of type *DataType* to a buffer of memory that will be allocated by means of the CBMALLOC callback function. It is the responsibility of the calling application to free this buffer To ensure that the correct free function is called it is safest to free with the CBFREE function.

Parameters

hAsr

Handle to the engine instance [INPUT].

DataType

Type of data to export. For now only DATA_SESSIONDATA is supported [INPUT].

ppBuf

Address of a variable where the pointer to the memory buffer will be written [OUTPUT].

pdwSize

Address of a variable where the length of the memory buffer will be written [OUTPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function was called on an engine that was in the wrong state. Valid states are DATAREADY, IDLE, SLEEP, RUN.
ERR_WRONG_DATA	An import of a data buffer with a wrong <i>DataType</i> field was attempted. Valid Data Types are DATA_SESSIONDATA.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrImportData(), CasrExportData(), CasrActivateData(), CasrGetActivatedData()

CasrGainSet

ERRORID CasrGainSet(HASR *hAsr*, WORD *Gain*)

Passes the analog gain setting to the API. This function has to be called in response to the **CBAGC** and the **CBASKCURRENTGAIN** callback function calls. It is the responsibility of the user application to actually set the gain of the acquisition hardware. This function only reports the gain information back to the engine.

Parameters

hAsr

Handle to the engine instance [INPUT].

Gain

The value of the current analog gain. This has to be a value between 1 (minimum) and 65535 (maximum). 0 should not be passed because it means that there will be no signal [INPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to an engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CBAGC(), CBASKCURRENTGAIN()

CasrGetActivatedContext

ERRORID CasrGetActivatedContext(HASR *hAsr*, PHCONT *phCont*)

Returns the handle of the context activated on an engine.

If no context has been activated then the function returns a handle 0 in **phCont*.

Parameters

hAsr

Handle to the engine instance [INPUT].

phCont

Pointer to the context handle [OUTPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_WRONG_STATE	This function was called with no context active in the engine.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrActivateContext(), CasrGetActivatedData(), CasrImportContext(),
CasrCloseContext()

CasrGetActivatedData

ERRORID CasrGetActivatedData(HASR *hAsr*, PHDATA **ppData*, PDWORD *pdwSize*)

Returns an array of the activated data handles on an engine. This function will allocate memory for 2 HDATA handles and return the memory address in *ppData*. (**ppData*)[0] will contain the active language and (**ppData*)[1] will contain the handle to the session data. It is up to the application to free this memory. To ensure that the correct free function is called it is safest to free with the CBFREE function.

Parameters

hAsr

Handle to the engine instance [INPUT].

ppData

Address of a variable that will receive the pointer to the allocated buffer [OUTPUT].

pdwSize

Address of a variable where the size of the array will be written [OUTPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.
ERR_MALLOC	Memory allocation failed. The callback functions CBMALLOC and CBFREE are called for storing and release the engine's dynamic data. Probably not enough heap memory.

See Also

CasrGetActivatedContext(), CasrImportData(), CasrActivateData(), CasrCloseData()

CasrGetActiveWords

ERRORID CasrGetActiveWords(HASR *hAsr*, PCWORDID **ppActiveWords*, PDWORD *pdwSize*)

Returns an array containing the words that are active on the engine. The engine allocates a buffer of word ids through the CBMALLOC callback function. It is up to the user to free this memory again. To ensure that the correct free function is called it is safest to free with the CBFREE function.

Parameters

hAsr

Handle to the engine instance [INPUT].

ppActiveWords

Address of a variable that points to the array containing the ID's of the active words.
[OUTPUT].

pdwSize

Address of a variable where the size of the array will be written [OUTPUT].

Return Value

Error code.

ERR_WRONG_STATE	Engine was in wrong state. Valid states are IDLE, SLEEP, RUN.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrSetActiveWords(), CasrGetActivatedContext(), CasrActivateContext()

CasrGetAPIVersion

ERRORID CasrGetAPIVersion(HAPI *hApi*, PDWORD *pdwAPIVersion*)

Returns the version of the current API implementation.

Parameters

hApi

Handle to the API instance [INPUT].

pdwAPIVersion

Pointer to the DWORD where the API version information will be written.

The DWORD has the following format:

0xMMmmbbbb

-> MM Major version number

-> mm Minor version number

-> bbbb Build number

[OUTPUT].

Return Value

This function always returns ERR_SUCCES.

See Also

CasrGetSpiVersion()

CasrGetParam

ERRORID CasrGetParam(HASR *hAsr*, CPARAMID *ParamId*, PCPARAMVALUE *pParamValue*)

Gets the current value of an engine parameter.

Parameters

hAsr

Handle to the engine instance [INPUT].

ParamId

ID of the parameter to get [INPUT].

pParamValue

Pointer to the variable where the engine parameter will be written [OUTPUT].

Return Value

Error code.

ERR_UNKNOW_PARAM	Parameter used in CasrSetParam or CasrGetParam is out of range.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrSetParam(), CasrGetParamList(), CasrSetParamList()

CasrGetParamList

ERRORID CasrGetParamList(HASR *hAsr*, PCPARAMLIST *pParamList*)

Gets the list of all the parameter settings of an engine.

Parameters

hAsr

Handle to the engine instance [INPUT].

pParamList

Pointer to the structure where the engine parameters will be written [OUTPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

Comments

pParamList should be allocated before calling this function.

See Also

CasrSetParam(), CasrGetParam(), CasrSetParamList()

CasrGetSignalLevel

ERRORID CasrGetSignalLevel(HASR *hAsr*, PCASRLEVEL *pCasrLevel*)

Gets the current signal level. This level is a value between -7200 (-72 dB) and 1800 (18 dB).

Parameters

hAsr

Handle to the engine instance [INPUT].

pCasrLevel

Pointer to the variable where the current signal level will be written [OUPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrGetSNR()

CasrGetSNR

BOOL casrGetSNR(HASR *hAsr*, CASRLEVEL **pSNR*)

Gets the current SNR of the signal seen so far. This level is a value between 0 (0 dB) and 5000 (50 dB).

Parameters:

hAsr

Handle to the engine instance [INPUT].

pSNR

Pointer to the variable where the SNR will be written [OUTPUT].

Return value:

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

GetSignalLevel()

CasrGetSPIVersion

ERRORID CasrGetSPIVersion(HAPI hApi, PDWORD *pdwSPIVersion*)

Returns the version of the current SPI implementation.

Parameters

hApi

Handle to the API instance [INPUT].

pdwSPIVersion

Pointer to the DWORD where the SPI version information will be written to.

The DWORD has the following format :

0xMMmmbbbb

-> MM Major version number

-> mm Minor version number

-> bbbb Build number

[OUTPUT].

Return Value

Error code.

See Also

CasrGetAPIVersion()

CasrGetState

ERRORID CasrGetState(HASR *hAsr*, PCASRSTATE *pState*)

Gets the current engine state.

Parameters

hAsr

Handle to the engine instance [INPUT].

pState

Pointer to a variable where the current engine state will be written [OUTPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrOpen(), CasrStart(), CBSTATE()

CasrImportContext

ERRORID CasrImportContext(HAPI *hApi*, PCONT *pCont*, BOOL *IsPermanent*, PHCONT *phCont*)

Imports a context and associates a handle with it. This context can then be activated on different engine instances with a low memory overhead. If the same context buffer is imported twice, the data is only copied once. When an imported context is not needed anymore, the resources should be freed with the CasrCloseContext() function.

Parameters

hApi

Handle to the API instance [INPUT].

pCont

Read only buffer containing a context. This data can be shared between different engine instances. [INPUT]

IsPermanent

TRUE if the buffer pointed to is permanent (stays in memory) and can be used as such as long as HCONT is valid. When this parameter is set to false the entire databuffer is copied by the API to a buffer allocated by CBMALLOC. [INPUT].

phCont

Address of a context handle. This context can then be used on different engines [OUTPUT].

Return Value

Error code.

ERR_WRONG_DATA	The data buffer that was passed was not a context data buffer.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrImportData(), CasrExportContext(), CasrActivateContext()

CasrImportData

ERRORID CasrImportData(HAPI *hApi*, PDATA *pData*, BOOL *IsPermanent*, PHDATA *phData*)

Loads data in memory and associates a handle with it. This data can then be activated on different engine instances with a low memory overhead. If the same data is imported twice, the data is only copied once. When imported data is not needed anymore, the resources should be freed with the CasrCloseData() function.

Parameters

hApi

Handle to the API instance [INPUT].

pData

Buffer containing the data. This data can be shared between different engine instances depending on the type of the data [INPUT].

IsPermanent

TRUE if the buffer pointed to is permanent (stays in memory) and can be used as such as long as HDATA is valid. If IsPermanent is FALSE memory will be allocated and the data will be copied. Data that can be changed by the engine (user specific models, inherits data) must be non-permanent if the engine is allowed to change it.

CasrExportData can be used to save the changed data [INPUT].

phData

Pointer to the handle of the data associated with pData. This language can then be used on different engines [OUTPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrImportContext(), CasrExportData(), CasrActivateData()

CasrOpen

ERRORID CasrOpen(HAPI *hApi*, PCRECOGCALLBACKS *pRecogCallbacks*,
DWORD *dwUserData*, PHASR *phAsr*)

Opens an engine and initializes it.

Parameters

hApi

Handle to the API instance [INPUT].

pRecogCallbacks

Pointer to a structure containing a set of callback functions that can be used by the engine. The structure is copied by the function and therefor can be non-permanent [INPUT].

dwUserData

DWORD that will be returned with every callback used to notify some information to the application. This can be used to assign user-data to an engine instance [INPUT].

phAsr

Pointer where the handle of the new engine instance will be written to [OUTPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

Callback functions within this function call

CBSTATE

Callback function used to notify state changes.

CBASKCURRENTGAIN

Callback function used to ask for the current setting of the analog gain.

See Also

CasrClose(), CasrAPIInit(), CasrAPIClose()

CasrSetActiveWords

ERRORID CasrSetActiveWords(HASR *hAsr*, PCWORDID *pidWords*, DWORD *NbrWords*)

Enables or disables certain words in 1 context. Activation is a fast way for switching active vocabularies.

Parameters

hAsr

Handle to the engine instance [INPUT].

pidWords

Array containing the ID's of all words that must be enabled. All the words that are not in this array will be disabled [INPUT].

NbrWords

Number of word ID's in *pidWords* [INPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function is called on an engine that was in the wrong state. Valid states are CASR_IDLE, CASR_DATAREADY.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_CONTEXT	Handle to the imported context is not valid, probably because it has been closed.
ERR_TASK_THREAD_CONTEXT	This function is called in another task or thread than the one that did the CasrImportContext of the active context.
ERR_MALLOC	Memory allocation failed. The callback functions CBMALLOC and CBFREE are called for storing and release the engine's dynamic data. Probably too less memory.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrActivateContext(), CasrImportContext(), CasrGetActiveWords()

CasrSetParam

ERRORID CasrSetParam(HASR *hAsr*, CPARAMID *ParamId*, CPARAMVALUE *ParamValue*)

Sets a certain engine parameter.

Parameters

hAsr

Handle to the engine instance [INPUT].

ParamId

ID of the parameter to set [INPUT].

ParamValue

New value of the parameter [INPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function is called on an engine that was in the wrong state. Valid states are CASR_IDLE, CASR_DATAREADY, CASR_BOOT, CASR_RECOVER .
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.
ERR_UNKNOW_PARAM	Parameter used in CasrSetparam is out of range.

See Also

CasrOpen(), CasrSetParamList(), CasrGetParam(), CasrGetParamList()

CasrSetParamList

ERRORID CasrSetParamList(HASR *hAsr*, PCPARAMLIST *pParamList*)

Initializes all the parameters. The buffer supplied can be retrieved with the **CasrGetParamList** function in a previous session.

Parameters

hAsr

Handle to the engine instance [INPUT].

pParamList

Pointer to the structure containing the new values of all the parameters [INPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function is called on an engine that was in the wrong state. Valid states are CASR_IDLE, CASR_DATAREADY, CASR_BOOT.
ERR_NULL_HANDLE	A NULL handle is passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

See Also

CasrGetParamList(), CasrGetParam(), CasrSetParam()

CasrStart

ERRORID CasrStart(HASR *hAsr*)

Start the recognition engine. From now on the recognition is active and notifications can be sent. The engine is put either in the SLEEP state or in the RUN state, depending on the setting of the START_ENABLE parameter.

Parameters

hAsr
Handle to the engine instance [INPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function is called on an engine that was in the wrong state. Valid states are CASR_IDLE.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

Callback functions within this function call

CBSTATE
Callback function used to notify state changes.

See Also

CasrOpen(), CasrStop()

CasrStop

ERRORID CasrStop(HASR *hAsr*)

Forces the recognition engine to stop. Whatever is heard up to this point, is turned into a recognition result.

Parameters

hAsr

Handle to the engine instance [INPUT].

Return Value

Error code.

ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

Callback functions within this function call

CBRESULT

Callback function used to indicate that a recognition result is available.

CBSTATE

Callback function used to notify state changes.

See Also

CasrOpen(), CasrStart(), CasrAcquisition()

5.2 Added functionality

5.2.1 User word training related

CasrAcknowledge

ERRORID CasrAcknowledge(HASR *hAsr*, DWORD *Decision*)

Gives an acknowledge (and a decision) to the engine to continue the user word training. This function can be called in the CBSTATE callback.

Parameters

hAsr

Handle to an engine installed opened with **CasrOpen** [INPUT].

Decision

Decision of the application (PROMPT_OK, PROMPT_NOK or PROMPT_CANCEL)
[INPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function is called on an engine that was in the wrong state. Valid states are CASR_PROMPT, CASR_ACCEPT, CASR_CONFIRM.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to the engine is used in another task or thread than the one that did the CasrOpen .
ERR_INVALID_SPI	Handle to SPI data is invalid or is used in another task than the task that created the handle.

Callback functions within this function call

CBSTATE

Callback function used to notify state changes.

CBTRAIN

Callback function used during user word training to notify user interactions.

See Also

CBTRAIN()

CasrAddContextUserWord

ERRORID CasrAddContextUserWord(HCONT *hCont*, HCLASSES *hClasses*, char **pszClass*, USERWORDBUF *userWordBuf*, DWORD *dwUserWordSize*, PCWORDID *pnFirstUserWordId*, PDWORD *pcUserWordIds*)

This function adds a user word to a context. Userwords are added to a class in a certain context. Since a class can occur more than once in a context, more than one id is generated and returned.

Parameters

hCont

Context instance handle [INPUT].

hClasses

Class names instance handle [INPUT].

pszClass

Class name string [INPUT].

userWordBuf

User word data buffer [INPUT].

dwUserWordSize

User word data buffer byte size [INPUT].

pnFirstUserWordId

Pointer where first user word ID assigned will be written [OUTPUT].

pcUserWordIds

Pointer where the number of assigned user word ID's will be stored [OUTPUT].

Return Value

Error code.

ERR_CONTEXT_IN_USE

Some engine is busy with this context.

ERR_MALLOC

Memory allocation failed.

The callback functions CBMALLOC and CBFREE are called for storing and release engine's dynamic data.

Probably not enough heap memory.

Comments

The parameter `userWordBuf` can be filled in with the buffer pointer returned via the API function `CasrStartUserWordTraining`, after the training process has been completed. When calling this function a next available user word ID range is assigned to the user word. The range is returned via the first ID and the number of IDs. It can be a range because the specified class can occur several times in the grammar. For each occurrence a new user word ID is generated. If the `userWordBuf` obtained by `CasrStartUserWordTraining` is saved, this function can be called with the same data at a later time. Saving this buffer to store userwords and adding them to a context when needed is the least memory consuming.

See Also

`CasrImportContext()`, `CasrCloseContext()`, `CasrStartUserWordTraining()`

CasrDeleteContextUserWord

ERRORID CasrDeleteContextUserWord(HCONT *hCont*, HCLASSES *hClasses*, char * *pszClass*, CWORDID *nFirstUserWordId*, DWORD *cUserWordIds*)

This function deletes a user word from a context.

Parameters

hCont
Context instance handle [INPUT].

hClasses
Class names instance handle [INPUT].

pszClass
Class name string [INPUT].

nFirstUserWordId
First user word ID [INPUT].

cUserWordIds
Number of user word ID's [INPUT].

Return Value

Error code.

ERR_CONTEXT_IN_USE	Some engine is still working with the context.
ERR_MALLOC	Memory allocation failed. The callback functions CBMALLOC and CBFREE are called for storing and release engine's dynamic data. Probably not enough heap memory.
ERR_INVALID_RANGE	Invalid range specified.

Comments

When calling this function all user word ID's bigger than *nFirstUserWordId* are decremented by *cUserWordIds*. User word ID's start from 0x80000000. This function deletes the range of user word ID's, as returned by the function CasrAddContextUserWord. Remark that only user words that have been added since the last context import can be deleted with this function.

See Also

CasrImportContext(), CasrCloseContext(), CasrAddContextUserWord()

CasrStartUserWordTraining

ERRORID CasrStartUserWordTraining(HASR *hAsr*, PUSERWORDBUF *pUserWordBuf*, PDWORD *pdwSize*)

Starts the user word training. The UserWordBuf and dwSize will not be written immediately, the addresses that are passed need to be valid until the userword training has ended. The START_ENABLE engine parameter needs to be disabled (set to FALSE) before calling this function. The userword training process does not have an equivalent of the sleep state, the engine always immediately jumps from *start* to *record*. If the userword is fully trained (A PROMPT_OK has been answered to the CONFIRM request) then the engine will allocate memory for the userword and pUserWordBuf will contain a pointer to this memory. This memory has been allocated with the CBMALLOC callback by the engine and it is up to the application to free the memory when the userword is not needed anymore. The application can use the CBFREE callback to free this memory.

Parameters

hAsr

Handle to the engine instance [INPUT].

pUserWordBuf

Address of a variable that will receive a pointer to a buffer containing the user word data. The user of the API can use this data to add the word to a context. This buffer will only be valid upon a successful receipt of a **PROMPTTYPE_TRAINEND** notification [OUTPUT]

pdwSize

Address of variable that receives the length of the buffer [OUTPUT].

Return Value

Error code.

ERR_WRONG_STATE	The function is called on an engine that was in the wrong state. Valid states are CASR_IDLE, CASR_DATAREADY.
ERR_NULL_HANDLE	A NULL handle was passed.
ERR_INVALID_CONTEXT	Handle to the imported context is not valid, probably because it has been closed.
ERR_TASK_THREAD_CONTEXT	Handle to a context is used in another task or thread than the one that did the CasrImportContext .
ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
ERR_TASK_THREAD	Handle to an engine is used in another task or thread than the one that did the CasrOpen .

ERR_MALLOC

Memory allocation failed.
The callback functions CBMALLOC and CBFREE
are called for storing and engine's dynamic data.
Probably not enough heap memory.

ERR_INVALID_SPI

Handle to the SPI is invalid or is used in another task
than the task that created the handle.

Callback functions within this function call

CBSTATE

Callback function used to notify state changes.

See Also

CasrAddContextUserWord(), CasrDeleteContextUserWord()

5.2.2 Context related

CasrExportContext

ERRORID CasrExportContext(HCONT *hCont*, PCONT *pCont*, PDWORD *pdwLength*)

This function exports a context to a data buffer. This function can be called with *pCont* equal to NULL in which case the *pdwLength* variable will be filled with the required buffer size. With this information the application can allocate that amount of memory and assign *pCont* with this. After this you call *CasrExportContext* again which will actually fill the buffer with the context data.

Parameters

hCont

Context instance handle [INPUT].

pCont

Context data buffer. Application should allocate enough memory before calling this function and put the buffersize in *dwLength* . [OUTPUT].

pdwLength

Pointer where length in bytes of context data buffer should be passed /will be written [INPUT/OUTPUT].

Return Value

Error code.

ERR_BUF_TOO_SMALL

Output buffer too small.

Comments

The context data buffer actual length is always returned, even if NULL is passed via the parameter *pCont*.

See Also

CasrImportContext()

5.2.3 Classes related

CasrCloseClasses

ERRORID CasrCloseClasses(HCLASSES *hClasses*)

This function closes a class names instance.

Parameters

hClasses
Class names instance handle [INPUT].

Return Value

In the current implementation this function always returns ERR_SUCCESS.

Comments

This function frees all used resources and invalidates the class names instance handle.

See Also

CasrImportClasses()

CasrImportClasses

ERRORID CasrImportClasses(HAPI *hApi*, PCLASSES *pClasses*, PHCLASSES *phClasses*)

This function imports a class names data buffer.

Parameters

hApi

Handle to the API instance [INPUT].

pClasses

Class names data buffer [INPUT].

phClasses

Class names instance handle [OUTPUT].

Return Value

Error code.

ERR_MALLOC

Memory allocation failed.

The callback functions CBMALLOC and CBFREE are called for storing and release engine's dynamic data.

Probably not enough memory.

Comments

The class names data buffer contains among other class names and class ID cross-references. This data buffer can be obtained from the PC tools. A class names instance handle is associated with the imported data.

See Also

CasrCloseClasses()

5.2.4 Merging of contexts and classes

CasrMergeContextsAndClasses

ERRORID CasrMergeContextsAndClasses(HCONT *hCont1*, HCONT *hCont2*, PHCLASSES *phClasses1*, PHCLASSES *phClasses2*, PHCONT *phContResult*, PHCLASSES *phClassesResult*)

This function merges two context buffers and their corresponding class buffers. This function creates a new context and new classes buffer. These buffers are already imported in the API and are identified with a handle (stored in *phContResult* and *phClassesResult*). If the merged context is no longer needed, the resources of these two handles should be freed with the CasrCloseContext() and CasrCloseClasses() functions.

Parameters

hCont1

Context instance handle [INPUT].

hCont2

Context instance handle [INPUT].

phClasses1

Class names instance handle [INPUT].

phClasses2

Class names instance handle [INPUT].

phContResult

Pointer to the context handle [OUTPUT].

phClassesResult

Class names instance handle [OUTPUT].

Return Value

Error code.

ERR_CONTEXT_IN_USE	Some engine was still using one of the engines.
ERR_MALLOC	Memory allocation failed. The callback functions CBMALLOC and CBFREE are called for storing and release engine's dynamic data. Probably not enough memory.
ERR_INCOMPATIBLE_CONTEXTS	Contexts are of a different type.
ERR_DOUBLE_CLASSNAMES	Contexts that contain equal classnames can not be merged.

Comments

The only purpose of this function is to make it possible to use two contexts at the same time. It is however not possible to merge two contexts which have classes with the same name. The handle for the merged context returned by this function can be used in the `CasrActivateContext()` function.

See Also

`CasrCloseContext()`, `CasrImportClasses()`, `CasrImportContext()`

6 Callbacks

CBABNORM

unsigned char (*CBABNORM)(ABNORMCOND AbnormalCondition, PUSERDATA pUserData)

Callback function used to notify abnormal conditions.

Parameters

AbnormalCondition: Type of abnormal condition.

Possible conditions:

BADSNR

The signal to noise ratio is too low.

OVERLOAD

The input signal is too loud.

TOOQUIET

The input signal is too soft.

NOSIGNAL

No signal has been seen for a time after the start of the agc.

GARBLED SOUND

A very short non-speech sound has been seen.

POORMIC

A poor microphone has been detected.

pUserData: User data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

Non-zero if there has been an error.

CBAGC

unsigned char (*CBAGC)(WORD NewGain, PUSERDATA pUserData)

Callback function used to request an analog gain change. In response of this callback the application has to pass the setting of the new gain to the API with the **CasrGainSet** function. The **CasrGainSet** function can be called within this callback or at a later time. Never calling the **CasrGainSet** function (e.g. If you use a dummy CBAGC callback) has the same effect as disabling the AGC functionality.

Parameters

NewGain: value for the new analog gain. This is a value between 1 (minimum) and 65535 (maximum). 0 means that the applications does not support this feature. The new value of the analog gain the application has changed to or 0 if the application does not support this feature.

pUserData: user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

Non-zero if there has been an error.

CBASKCURRENTGAIN

unsigned char (*CBASKCURRENTGAIN)(PUSERDATA pUserData)

Callback function used to ask for the current setting of the analog gain. In response of this callback the application has to pass the setting of the gain to the API with the **CasrGainSet** function. The **CasrGainSet** function can be called within this callback or at a later time.

Parameters

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

Non-zero if there has been an error.

CBRESULT

unsigned char (*CBRESULT)(PCASRRESULT pResult, PUSERDATA pUserData)

Callback function used to indicate that a recognition result is available.

Parameters

pResult : pointer to the result structure. This pointer is only valid as long as the program is inside the callback function. The result structure is destroyed when the callback function returns.

Members of the result structure are:

iNbr

Number of recognition alternatives.

pSentences

Array containing all sentences of the N-best result. They are ordered according to their confidence value (probability). Per sentence you have an iConf member which indicates how confident the recognizer is that this sentence is the correct one.

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

Non-zero if there has been an error.

CBSTATE

unsigned char (*CBSTATE)(CASRSTATE State, PUSERDATA pUserData)

Callback function used to notify state changes.

Parameters

State: new State.

Possible states are:

CASR_BOOT

No contexts, language or user activated yet.

CASR_DATAREADY

All necessary data is activated.

CASR_IDLE

Context, language and user are activated.

CASR_SLEEP

Low CPU start detection.

CASR_RUN

Recognizing.

CASR_RECOVER

TS detected or **CasrStop** called. Notifying the result.

CASR_PROMPT

Waiting for user feedback during training to start the utterance.

CASR_RECORD

Recording a user word utterance.

CASR_ACCEPT

Waiting for user feedback during training to accept an utterance.

CASR_CONFIRM

Waiting for user feedback during training to accept all the training data.

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

Non-zero if there has been an error.

CBTRAIN

unsigned char (*CBTRAIN)(PROMPTTYPE PromptType, PUSERDATA pUserData)

Callback function used during user word training to notify user interactions.

Parameters

PromptType: type of prompt notification. See **PROMPTTYPE** for possible values.

Possible prompt types are:

PROMPTTYPE_START

Send before the start of an utterance. Upon return of this function the user can start speaking. Possible acknowledges:

- PROMPT_OK : The speaker may start to speak.
- PROMPT_CANCEL : The training will be canceled and a **PROMPTTYPE_TRAINEND** will be send.

PROMPTTYPE_ACCEPT

Send to ask the user if he wants to accept the spoken utterance. Possible acknowledges:

- PROMPT_OK : The speaker accepts this utterance.
- PROMPT_NOK : The speaker rejects this utterance.
- PROMPT_CANCEL : The training will be canceled and a **PROMPTTYPE_TRAINEND** will be send.

PROMPTTYPE_CONFIRM

Send when the 3 utterances are spoken, to ask for the user a confirmation if he wants to create the user word using the 3 utterances. Possible acknowledges:

- PROMPT_OK : OK, create.
- PROMPT_CANCEL : The user word will not be created and a **PROMPTTYPE_TRAINEND** will be send.

PROMPTTYPE_TRAINED

The training has finished. No acknowledge necessary.

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

Non-zero if there has been an error.

7 OS Callbacks

This is the list of callbacks that encapsulates the needed operating system functionality used in the ASR1600\C API. For a non-multitasking operating system, the 3 callback functions that are really needed are CBMALLOC, CBFREE and CBREALLOC, all the other functions can be implemented as dummy functions. The other ones are only needed to implement a multitasking and multithreading save API.

CBCREATECRITICALSECTION

HCRITSECTION (*CBCREATECRITICALSECTION)(PUSERDATA pUserData)

Callback function for creation and initialization of a critical section. Callbacks for operating systems that do not support multithreading can be implement as a dummy function. The dummy function can simply return 1. A critical section can be used to synchronize resources that are accessed in different threads.

Return Value

Handle (type **HCRITSECTION**) associated with the critical section. NULL if there has been an error.

Parameters

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** or **CspellOpen** and **CspellAPIInit** function.

CBDELETECRITICALSECTION

BOOL (*CBDELETECRITICALSECTION)(HCRITSECTION hCritSection, PUSERDATA pUserData)

Callback function for deleting a critical section. Callbacks for operating systems that do not support multithreading can be implemented as a dummy function. The dummy function should simply return **TRUE**. A critical section can be used to synchronize resources that are accessed in different threads. This function frees all the resources used by the critical section.

Parameters

hCritSection : handle of the critical section to delete.

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** or **CspellOpen** and **CspellAPIInit** function.

Return Value

TRUE if there has been an error.

CBENTERCRITICALSECTION

BOOL (*CBENTERCRITICALSECTION)(HCRITSECTION hCritSection, PUSERDATA pUserData)

Callback function for entering a critical section. Callbacks for operating systems that do not support multithreading can be implemented as a dummy function. The dummy function should simply return **TRUE**. A critical section can be used to synchronize resources that are accessed in different threads. Upon return of this callback the calling thread owns this critical section. Another thread locks if it calls this function until the first thread calls

CBLEAVECRITICALSECTION.

Parameters

hCritSection : handle of the critical section to enter.

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** or **CspellOpen** and **CspellAPIInit** function.

Return Value

TRUE if there has been an error.

CBFREE

void (*CBFREE)(PVOID pMem, PUSERDATA pUserData)

Callback function used to release memory obtained with function of type **CBMALLOC**.

Parameters

pMem : pointer to the memory to free.

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

No return value.

CBGETCURTASK

int (*CBGETCURTASK)(PUSERDATA pUserData)

Callback function used to ask for the task ID Callbacks for operating systems that do not support multitasking can always return 0. This function is used to check if handles are not used between tasks.

Return Value

Current task's ID.

Parameters

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** or **CspellOpen** and **CspellAPIInit** function.

CBGETCURTHREAD

int (*CBGETCURTHREAD)(PUSERDATA pUserData)

Callback function used to ask for the thread ID. Callbacks for operating systems that do not support multithreading can always return 0. This function is used to check if handles are not used between threads.

Return Value

ID of the current thread.

Parameters

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** function.

CBLEAVECRITICALSECTION

BOOL (*CBLEAVECRITICALSECTION)(HCRITSECTION hCritSection, PUSERDATA pUserData)

Callback function for leaving a critical section. Callbacks for operating systems that do not support multithreading can be implemented as a dummy function. The dummy function should simply return **TRUE**. A critical section can be used to synchronize resources that are accessed in different threads. This function releases the ownership of a critical section and unlocks another thread that is waiting for the ownership (in the **CBENTERCRITICALSECTION** function).

Parameters

hCritSection : handle of the critical section to leave.

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

TRUE if there has been an error.

CBMALLOC

void * (*CBMALLOC)(DWORD dwSize, PUSERDATA pUserData)

Callback function to a memory allocation function.

Parameters

dwSize : size of the memory buffer to allocate.

pUserData : user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return Value

Pointer to the newly allocated block of memory or NULL if not enough memory available.

CBREALLOC

```
void* (* CBREALLOC)(void *Ptr,DWORD Size, PUSERDATA pUserData);
```

Callback function to a function used to adjust the size of a memory buffer previously allocated with CBMALLOC. This callback is currently not used in this API. It is specified in the callback structure because it is needed when spelling is supported.

Parameters

Ptr: Pointer to the memory buffer to resize.

Size: New Size for the memory buffer.

pUserData: user data passed with the **CasrOpen** and **CasrAPIInit** function.

Return value:

Pointer to the newly re-allocated block of memory or NULL if not enough memory available. The returned pointer is not necessarily the same as *Ptr*.

8 Types, structures and defines

8.1 Types

BOOL

Boolean variable, can be TRUE (1) or FALSE (0).

CASRLEVEL

Short, representing the current signal level of the engine. Valid values are between -7200 (-72 dB) and 1800 (+18 dB).

CPARAMID

DWORD representing the ID of a certain engine parameter. Following parameters are available :

a. acoustic engine (handle type HASR)

CPARAM_TS_ENABLE

To enable or disable trailing silence detection.

Range: TRUE (1) or FALSE (0)

Default: TRUE (1)

CPARAM_START_ENABLE

To enable or disable low CPU start detection.

Range: TRUE (1) or FALSE (0)

Default: TRUE (1)

CPARAM_ENABLEPREMRES

Enable the generation of premature results. This is useful in keyword spotting mode where the engine is running continuously and results are notified each time a keyword is recognized.

Range: TRUE (1) or FALSE (0)

Default: FALSE (0)

CPARAM_TS

Minimum amount of trailing silence before the end of an utterance is detected.

Range: 100 ms - 2000 ms

Default: 300 ms

CPARAM_TIMEOUT

Maximum amount of time the engine can recognize (SLEEP, RUN or RECORD state).

Range: 0 sec (no Time Out) - 30 sec

Default: 0

CPARAM_ACCURACY

Trade off between CPU-load, memory requirements and the obtained accuracy of the recognizer. The lower this values the less CPU-time and memory are needed, but the performance will also degrade. The higher this value, the more CPU-time and memory, but the performance will increase.

Range: 100 - 10000

Default: 300

CPARAM_REJECTION

With this parameter you can change the confidence level values. The higher this value the more rejections (the lower the confidence levels).

Range: 0 - 100

Default: 50

CPARAM_GARBAGE

With this parameter you can change the model for the anyspeech <...>. The higher this value the more speech will match the anyspeech model in favor of the words of the context.

Range: 0 - 100
Default: 50

CPARAM_SENSITIVITY

Sensitivity of the low CPU speech detection system (if enabled). The lower this value the easier begin of speech will be detected.

Range: 0 (0 dB) - 4000 (40 dB)
Default: 2000 (20 dB)

CPARAM_MINSPEECH

Minimum duration of speech the low CPU speech detection system has to see before begin of speech is detected.

Range: 10 msec - 400 msec
Default: 60 msec

CPARAM_AGCON

Switches AGC on or off.

Range: TRUE (1) or FALSE (0)
Default: FALSE (0)

CPARAM_MAXNBEST

Maximum number of sentences in the result.

Range: 1 - 1000
Default: 10

CPARAM_FARTALK

Informs the AGC if a far talk or a close talk microphone is used.

Range: TRUE (1) or FALSE (0)
Default: TRUE (1)

CPARAM_SELECTGENDER

Selects the genders that are being used by the engine. If more then 1 gender is selected, the engine automatically defines the gender of the speaker. If the gender of the speaker is known in advance it is better to set this gender on the engine. This saves memory and CPU and very little in recognition performance. This parameter is a combination (or-ing) of the following defines:

GENDER_MALE
GENDER_FEMALE
GENDER_CHILD
GENDER_BOY
GENDER_GIRL
GENDER_UNKNOWN

Which genders can be activated depends on the activate language. GENDER_UNKNOWN means all genders that are in the language. Most languages contain at least the GENDER_MALE and GENDER_FEMALE gender.

Range: Bitfield made by or-ing one of more of the following defines: GENDER_MALE, GENDER_FEMALE, GENDER_CHILD, GENDER_BOY or GENDER_GIRL.
Default: GENDER_UNKNOWN

CPARAMVALUE

DWORD representing the value of a certain engine parameter. See **CPARAMID** as well.

CWORDID

DWORD representing a word in a context.

DWORD

Unsigned long (32 bit).

ERRORID

DWORD containing an error code. Following error codes are available.

Basic error codes:

0	ERR_SUCCESS	Function executed successfully.
1	ERR_MALLOC	Memory allocation failed.
2	ERR_TASK_THREAD	Handle to an engine was used in another task or thread than the one that did the CasrOpen .
3	ERR_APIINIT	CasrAPIInit called more than once.
4	ERR_CONTEXT_WRONG_HANDLE	Context handle is not present in the global list of imported contexts.
5	ERR_CONTEXT_IN_USE	A CasrCloseContext was attempted on a context that is still in use on some engine.
6	ERR_NULL_HANDLE	A NULL handle was passed to some API function.
7	ERR_WRONG_STATE	An engine function was called on an engine that was in the Wrong state.
8	ERR_DATA_WRONG_HANDLE	Data handle is not present in the global list of imported data.
9	ERR_DATA_IN_USE	CasrCloseData was attempted on a data import that is still in use on some engine.
10	ERR_UNKNOWN_PARAM	Parameter used in CasrSetParam or CasrGetParam is out of range.
11	ERR_INVALID_ENGINE	Handle to the engine is not valid, possibly because the engine has been closed.
12	ERR_LANGUAGE_MISMATCH	attempt to activate context or session data that does not match the activated language
13	ERR_WRONG_DATA	An import of some data buffer with a wrong DataType field was attempted
14	ERR_INVALID_DATA	Handle to the imported data not valid, probably because it has been closed
15	ERR_TASK_THREAD_DATA	Handle to an imported data was used in another task or thread than the one that did the CasrImportData

16	ERR_INVALID_CONTEXT	handle to the imported context not valid, probably because it has been closed
17	ERR_TASK_THREAD_CONTEXT	handle to a context was used in another task or thread than the one that did the CasrImportContext
18	ERR_INVALID_SPI	handle to a SPI was invalid or was used in another task then the task the created the handle
19	ERR_NULL_POINTER	a NULL pointer was passed to some SPI function
20	ERR_INVALID_GENDER	illegal value for CPARAM_SELECTGENDER (might also occur during CasrActivate!)
20	ERR_NO_USERWORD_TRAINING	the currently active language is unable to do userword training
22	ERR_NO_START_DETECTION	attempted CasrStartUserWordTraining with start detection enabled
23	ERR_WRONG_SIZE	A buffer with the wrong size has been passed
24	ERR_UTTFORMAT_UNKNOWN	Unknown utterance format
25	ERR_INVALID_WORDID	at least one wordID is out of range in CasrSetActiveWords
26	ERR_TIMEOUT	Timeout occurred

Added error codes:

100	ERR_INCOMPATIBLE_HANDLES	handles are not compatible
110	ERR_BUF_TOO_SMALL	output buffer too small
120	ERR_INVALID_CLASSNAME	classname is not valid
130	ERR_INCOMPATIBLE_CONTEXTS	contexts are of a different type.
131	ERR_DOUBLE_CLASSNAMES	contexts which contain equal classnames can not be merged.
140	ERR_INVALID_RANGE	invalid range specified.

HASR

Pointer representing a handle to an engine instance.

HCONT

Pointer representing a handle to context data that can be loaded, activated and enabled on an engine.

HCLASSES

Pointer representing a class names instance handle.

HCRITSECTION

Pointer representing a handle associated with a critical section.

HDATA

Pointer representing a handle associated with loaded data in memory.

PALTERNATIVE

pointer to **ALTERNATIVE**.

PCASRLEVEL

pointer to **CASRLEVEL**.

PCASRRESULT

pointer to **CASRRESULT**.

PCASRSTATE

pointer to **CASRSTATE**.

PDATA

pointer to **DATA**.

PCLASSES

Class names data buffer.

PCOSCALLBACKS

pointer to **COSCALLBACKS**.

PCPARAMLIST

pointer to **CPARAMLIST**.

PCPARAMVALUE

pointer to **CPARAMVALUE**.

PCRECOGCALLBACKS

pointer to **CRECOGCALLBACKS**.

PCONT

PVOID representing a pointer to a context buffer.

PCWORDID

pointer to **CWORDID**.

PDWORD

pointer to **DWORD**.

PHASR

pointer to **HASR**.

PHCONT

Pointer to a **HCONT**.

PHCLASSES

Pointer to a **HCLASSES**.

PHDATA

pointer to **HDATA**.

PRECWORD

pointer to **RECWORD**.

PSENTENCE

pointer to **SENTENCE**.

PSPEECHUNITBUF

pointer to **SPEECHUNITBUF**.

PTIMEINFO

pointer to **TIMEINFO**.

PUSERWORDBUF

pointer to **USERWORDBUF**.

PVOID

void pointer.

PWORD

pointer to **WORD**.

PWORDBUF

pointer to **WORDBUF**.

PUSERDATA

pointer to **USERDATA**.

USERWORDBUF

PVOID representing a pointer to a user word buffer.

WORD

unsigned short (16 bit).

8.2 Structures

ALTERNATIVE

```
typedef struct {  
    int iConf;  
    long lScore;  
    CWORDID idWord;  
} ALTERNATIVE;
```

Formatted part of a recognition result.

Members

iConf

The confidence level of this alternative.

lScore

Score for this alternative, the lower the better.

idWord

id of this alternative (User word ids have an offset **OFFSET_USERWORDID**).

CASRRESULT

```
typedef struct {  
    int iNbr;  
    PSENTENCE pSentences;  
} CASRRESULT;
```

Represents a formatted recognition result.

Members

iNbr

Number of recognition alternatives.

pSentences

Array containing all sentences of the N-best result.

COSCALLBACKS

```
typedef struct {  
    CBMALLOC cbMalloc;  
    CBFREE cbFree;  
    CBREALLOC cbRealloc;  
    CBGETCURTASK cbGetCurThread;  
    CBGETCURTHREAD cbGetCurThread;  
    CBCREATECRITICALSECTION cbCreateCriticalSection;  
    CBDELETECRITICALSECTION cbDeleteCriticalSection;  
    CBENTERCRITICALSECTION cbEnterCriticalSection;  
    CBLEAVECRITICALSECTION cbLeaveCriticalSection;  
} COSCALLBACKS;
```

Structure representing a set of callback functions that encapsulates operating system calls needed by the API. This structure is passed with the `casrAPIInit` function.

Members

cbMalloc

Memory allocation function.

cbFree

Memory freeing function.

cbRealloc

Re-allocating function.

cbGetCurThread

Get id of current task (only useful in multitasking environments).

cbGetCurThread

Get id of current thread (only useful in multithreading environments).

cbCreateCriticalSection

Function for creation and initialization of a critical section (only useful in multithreading environments).

cbDeleteCriticalSection

Function for deleting a critical section (only useful in multithreading environments).

cbEnterCriticalSection

Function for entering a critical section (only useful in multithreading environments).

cbLeaveCriticalSection

Function for leaving a critical section (only useful in multithreading environments).

CPARAMLIST

```
typedef struct {  
    DWORD cparam_ts_enable;  
    DWORD cparam_start_enable;  
    DWORD cparam_ts;  
    DWORD cparam_timeout;  
    DWORD cparam_accuracy;  
    DWORD cparam_rejection;  
    DWORD cparam_garbage;  
    DWORD cparam_sensitivity;  
    DWORD cparam_minspeech;  
    DWORD cparam_agcon;  
    DWORD cparam_maxnbest;  
    DWORD cparam_enablepremres;  
    DWORD cparam_fartalk;  
    DWORD cparam_selectgender;  
} CPARAMLIST;
```

Structure representing a list of parameters. See also **CPARAMID** and **CPARAMVALUE**.

Members

cparam_ts_enable

Trailing silence detection on or of.

cparam_start_enable

Low CPU start detection on or of.

cparam_ts

Minimum amount of trailing silence.

cparam_timeout

Maximum amount of time the engine can recognize.

cparam_accuracy

Accuracy of the recognizer.

cparam_rejection

Penalty for the rejection word.

cparam_garbage

Penalty for the garbage model.

cparam_sensitivity

Sensitivity of the low CPU speech detection.

cparam_minspeech

Minimum amount of speech the low CPU detection system has to see before speech is detected.

cparam_agcon

AGC on or of.

cparam_maxnbest

Maximum number of sentences in the result.

cparam_enablepremres

Generation of premature results enabled or disabled.

cparam_fartalk

Close talk or far talk microphone.

cparam_selectgender

Sets the level of automatic gender detection.

CRECOGCALLBACKS

```
typedef struct {  
    CBRESULT cbResult;  
    CBSTATE cbState;  
    CBTRAIN cbTrain;  
    CBABNORM cbAbnorm;  
    CBAGC cbAgc;  
    CBASKCURRENTGAIN cbAskCurrentGain;  
} CRECOGCALLBACKS;
```

Represents a set of callback functions the engine can use. This structure is passed with the **CasrOpen** function.

Members

cbResult

Result notification.

cbState

State notification.

cbTrain

User word training notification.

cbAbnorm

Abnormal condition notification.

cbAgc

Request to change the analog gain.

cbAskCurrentGain

Returns the current gain setting.

DATA

```
typedef struct {  
    DWORD dwDataType;  
    DWORD dwSize;  
} DATA;
```

Structure representing data buffer information.

Members

dwDataType

Type of data in the buffer. Can be **DATA_LANG**, **DATA_LANG_FIXED**, **DATA_CONTEXT**, **DATA_SESSIONDATA** or **DATA_NAMETREE**.

dwSize

Size of the data buffer.

RECWORD

```
typedef struct {  
    int iNbrAlternatives;  
    TIMEINFO TimeInfo;  
    ALTERNATIVE* pAlternatives;  
} RECWORD;
```

Formatted part of a recognition result.

Members

iNbrAlternatives

Number of alternatives for this word.

TimeInfo

Contains the times of the begin and the end of this word.

pAlternatives

Array containing all the alternatives.

SENTENCE

```
typedef struct {  
    int iNbrWords;  
    int iConf;  
    long lScore;  
    TIMEINFO TimeInfo;  
    RECWORD pWords;  
} SENTENCE;
```

Formatted part of a recognition result.

Members

iNbrWords

Number of words in this sentence.

iConf

Confidence level of this sentence.

lScore

Score for this sentence, the lower the better. This value needn't be used, the confidence value is a better choice.

TimeInfo

Contains the times of the begin and the end of this sentence.

pWords

Array containing the words of this sentence.

SPEECHUNITBUF

```
typedef struct {  
    WORD NbrSpeechUnits;  
    PWORD pSpeechUnits;  
} SPEECHUNITBUF;
```

Structure representing the speech units of a pronunciation of a word.

Members

NbrSpeechUnits

Number of speech units in the array.

pSpeechUnits

Array of speech units.

TIMEINFO

```
typedef struct {
```

```

    DWORD TimeBegin;
    DWORD TimeEnd;
} TIMEINFO;

```

Structure returning time information about the utterance that was said. All returned times are in number of samples received counting from the time that CastStart has been called. A value of -1 (0xffffffff) means that no timing information is returned.

Members

TimeBegin

Begin time of the corresponding utterance.

TimeEnd

End time of the corresponding utterance.

USERDATA

```

typedef struct {
    DWORD dwApiUserData;
    DWORD dwEngineUserData;
} USERDATA;

```

Structure representing the different pronunciations of a word.

Members

dwApiUserData

User data passed with the CasrAPIInit or CspellAPIInit function.

dwEngineUserData

User data passed with the CasrOpen or CspellOpen function.

WORDBUF

```

typedef struct {
    WORD NbrProns;
    PSPEECHUNITBUF pSpeechUnitBufs;
} WORDBUF;

```

Structure representing the different pronunciations of a word.

Members

NbrProns

Number of pronunciations.

pSpeechUnits

Array of structures representing the speech units of the different pronunciations.

8.3 Enumeration types

ABNORMCOND

```
enum ABNORMCOND {  
    BADSNR,  
    OVERLOAD,  
    TOOQUIET,  
    NOSIGNAL,  
    GARBLED_SOUND,  
    POORMIC  
};
```

Type of the abnormal condition that occurred.

Members

BADSNR

The signal to noise ratio is too low.

OVERLOAD

The input signal is too loud.

TOOQUIET

The input signal is too soft.

NOSIGNAL

No signal has been seen for a time after the start of the AGC.

GARBLED_SOUND

A very short non-speech sound has been seen.

POORMIC

A poor microphone has been detected.

CASRSTATE

```
enum CASRSTATE {  
    CASR_BOOT,  
    CASR_DATA_READY,  
    CASR_IDLE,  
    CASR_SLEEP,  
    CASR_RUN,  
    CASR_RECOVER,  
    CASR_PROMPT,  
    CASR_RECORD,  
    CASR_ACCEPT,  
    CASR_CONFIRM  
};
```

Represents the current state of an engine.

Members

CASR_BOOT

No contexts, language or user activated yet.

CASR_DATA_READY

All necessary data is activated.

CASR_IDLE

Context, language and user are activated.

CASR_SLEEP

Low CPU start detection.

CASR_RUN

Recognizing.

CASR_RECOVER

TS detected or **CasrStop** called. Notifying the result.

CASR_PROMPT

Waiting for user feedback during training to start the utterance.

CASR_RECORD

Recording a user word utterance.

CASR_ACCEPT

Waiting for user feedback during training to accept an utterance.

CASR_CONFIRM

Waiting for user feedback during training to accept all the training data.

CSAMPLEFORMAT

```
enum CSAMPLEFORMAT {  
    PCM_16_11KHZ  
};
```

Specifies the format of the samples passed in the **CasrAcquisition** function.

Members

PCM_16_11KHZ

16-bit samples PCM at 11 kHz.

PROMPTTYPE

```
enum PROMPTTYPE {  
    PROMPTTYPE_START,  
    PROMPTTYPE_ACCEPT,  
    PROMPTTYPE_CONFIRM,  
    PROMPTTYPE_TRAINED  
};
```

Represents the different types of prompt notifications send during user word training. The decision has to be notified to the engine with the **CasrAcknowledge** function call.

Members

PROMPTTYPE_START

Send before the start of an utterance. Upon return of this function the user can start speaking. Possible acknowledges:

- PROMPT_OK : The speaker may start to speak.
- PROMPT_CANCEL : The training will be canceled and a **PROMPTTYPE_TRAINED** will be send.

PROMPTTYPE_ACCEPT

Send to ask the user if he wants to accept the spoken utterance. Possible acknowledges:

- PROMPT_OK : The speaker accepts this utterance.
- PROMPT_NOK : The speaker rejects this utterance.
- PROMPT_CANCEL : The training will be canceled and a **PROMPTTYPE_TRAINED** will be send.

PROMPTTYPE_CONFIRM

Send when the 3 utterances are spoken, to ask for the user a confirmation if he wants to create the user word using the 3 utterances. Possible acknowledges:

- PROMPT_OK : OK, create.
- PROMPT_CANCEL : The user word will not be created and a **PROMPTTYPE_TRAINEND** will be send.

PROMPTTYPE_TRAINED

The training has finished. No acknowledge necessary.