Dialogic
Making Innovation Thrive

Dialogic® NaturalAccess™ Voice Control Element API Developer's Manual

October 2009 64-0505-01

www.dialogic.com

Copyright and legal notices

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

Dialogic, Dialogic Pro, Brooktrout, Diva, Cantata, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, Diva ISDN, TruFax, Exnet, EXS, SwitchKit, N20, Making Innovation Thrive, Connecting to Growth, Video is the New Voice, Fusion, Vision, PacketMedia, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation or its subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. The names of actual companies and product mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Revision history

Revision	Release date	Notes
9000-6422-10	April 1997	SRG, CT Access 1.0, portions of 6304-10 and 6305-10
9000-6422-10	January 1998	SRG, CT Access 2.0
9000-6422-11	September 1998	MVH
9000-6422-12	February 1999	NBS
9000-6422-13	December 1999	NBS, CT Access 3.0
9000-6422-14	July 2000	LBG, Platform Support for Fusion 4.0
9000-6422-15	September 2000	LBG, CT Access 3.0/4.0
9000-6422-16	April 2001	MCM, Natural Access 2000-2
9000-6422-17	November 2001	MCM, Natural Access 2002-1 Beta
9000-6422-18	May 2002	SRG, Natural Access 2002-1
9000-6422-19	April 2003	SRG, Natural Access 2003-1
9000-6422-20	December 2003	MVH, Natural Access 2004-1
9000-6422-21	November 2004	SRR, Natural Access 2005-1 Beta
9000-6422-22	March 2005	SRR, Natural Access 2005-1
9000-6422-23	October 2005	SRG, Natural Access 2005-1, SP 1
9000-6422-24	July 2006	SRG, Natural Access 2005-1, SP 2
9000-6422-25	February 2009	DEH, Natural Access R8.1
64-0505-01	October 2009	LBG, NaturalAccess R9.0
Last modified: September 4, 2009		

Refer to www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

Table Of Contents

1 Introduction

The *Dialogic® NaturalAccess™ Voice Control Element API Developer's Manual* provides:

- An overview of the Voice Message service
- A reference to functions, errors, events, and reasons

This manual defines telephony terms where applicable, but assumes that you are familiar with telephony concepts. It also assumes that you are familiar with the C programming language.

Read the *Dialogic® NaturalAccess™ Software Developer's Manual* before using this manual. The *Dialogic® NaturalAccess™ Software Developer's Manual* contains detailed information on Natural Access concepts, architecture, and application development. This information must be fully understood before you develop a voice application using Natural Access.

2 Terminology

Note: The product to which this document pertains is part of the NMS Communications Platforms business that was sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") on December 8, 2008. Accordingly, certain terminology relating to the product has been changed. Below is a table indicating both terminology that was formerly associated with the product, as well as the new terminology by which the product is now known. This document is being published during a transition period; therefore, it may be that some of the former terminology will appear within the document, in which case the former terminology should be equated to the new terminology, and vice versa.

Former terminology	Dialogic terminology
CG 6060 Board	Dialogic® CG 6060 PCI Media Board
CG 6060C Board	Dialogic® CG 6060C CompactPCI Media Board
CG 6565 Board	Dialogic® CG 6565 PCI Media Board
CG 6565C Board	Dialogic® CG 6565C CompactPCI Media Board
CG 6565e Board	Dialogic® CG 6565E PCI Express Media Board
CX 2000 Board	Dialogic® CX 2000 PCI Station Interface Board
CX 2000C Board	Dialogic® CX 2000C CompactPCI Station Interface Board
AG 2000 Board	Dialogic® AG 2000 PCI Media Board
AG 2000C Board	Dialogic® AG 2000C CompactPCI Media Board
AG 2000-BRI Board	Dialogic® AG 2000-BRI Media Board
NMS OAM Service	Dialogic® NaturalAccess™ OAM API
NMS OAM System	Dialogic® NaturalAccess™ OAM System
NMS SNMP	Dialogic® NaturalAccess™ SNMP API
Natural Access	Dialogic® NaturalAccess™ Software
Natural Access Service	Dialogic® NaturalAccess™ Service
Fusion	Dialogic® NaturalAccess™ Fusion™ VoIP API
ADI Service	Dialogic® NaturalAccess™ Alliance Device Interface API
CDI Service	Dialogic® NaturalAccess™ CX Device Interface API
Digital Trunk Monitor Service	Dialogic® NaturalAccess™ Digital Trunk Monitoring API
MSPP Service	Dialogic® NaturalAccess™ Media Stream Protocol Processing API
Natural Call Control Service	Dialogic® NaturalAccess™ NaturalCallControl™ API
NMS GR303 and V5 Libraries	Dialogic® NaturalAccess™ GR303 and V5 Libraries

Former terminology	Dialogic terminology
Point-to-Point Switching Service	Dialogic® NaturalAccess™ Point-to-Point Switching API
Switching Service	Dialogic® NaturalAccess™ Switching Interface API
Voice Message Service	Dialogic® NaturalAccess™ Voice Control Element API
NMS CAS for Natural Call Control	Dialogic® NaturalAccess™ CAS API
NMS ISDN	Dialogic® NaturalAccess™ ISDN API
NMS ISDN for Natural Call Control	Dialogic® NaturalAccess™ ISDN API
NMS ISDN Messaging API	Dialogic® NaturalAccess™ ISDN Messaging API
NMS ISDN Supplementary Services	Dialogic® NaturalAccess™ ISDN API Supplementary Services
NMS ISDN Management API	Dialogic® NaturalAccess™ ISDN Management API
NaturalConference Service	Dialogic® NaturalAccess™ NaturalConference™ API
NaturalFax	Dialogic® NaturalAccess™ NaturalFax™ API
SAI Service	Dialogic® NaturalAccess™ Universal Speech Access API
NMS SIP for Natural Call Control	Dialogic® NaturalAccess™ SIP API
NMS RJ-45 interface	Dialogic® MD1 RJ-45 interface
NMS RJ-21 interface	Dialogic® MD1 RJ-21 interface
NMS Mini RJ-21 interface	Dialogic® MD1 Mini RJ-21 interface
NMS Mini RJ-21 to NMS RJ-21 cable	Dialogic® MD1 Mini RJ-21 to MD1 RJ-21 cable
NMS RJ-45 to two 75 ohm BNC splitter cable	Dialogic® MD1 RJ-45 to two 75 ohm BNC splitter cable
NMS signal entry panel	Dialogic® Signal Entry Panel

3 Overview of the Voice Message service

Voice Message service definition

The Voice Message service is a Natural Access service for developing voice applications. You must have Natural Access installed on your system to build applications using the Voice Message service. For detailed information about Natural Access, refer to the *Natural Access Developer's Reference Manual*.

The Voice Message service:

- Provides a set of functions for playing, recording, and editing voice messages in files or memory.
- Supports NMS VOX files, WAVE files, and flat (unformatted) files.
- Provides advanced functions for playing a list of concatenated messages and assigning message numbers to segments of an open file or a memory block.
- Uses the ADI service play and record functions.

To use the Voice Message service with the ADI service, open both services on the same context. Refer to the *ADI Service Developer's Reference Manual* for more information about the ADI service play and record functions.

Refer to the demonstration programs that use the Voice Message service play and record functions.

Voice file types

The Voice Message service supports the following voice file types:

- NMS VOX file format. This file format stores multiple messages in one file and enables messages to be deleted, added, edited, and annotated. Refer to *Overview of VOX file format* on page 137 and *VOX file information: voxinfo* on page 125 for more information.
- Waveform audio file format (WAVE) files. Refer to the *Microsoft Windows MultiMedia Programmer's Reference Manual* for information about the WAVE file format.
- Flat files that contain no header or format.

Voice encoding formats

When recording speech files, you must select an encoding format. The primary issue to consider when selecting a format is the compression ratio and fidelity. More aggressive compression requires less disk space and reduces host-to-board loading, but uses more DSP resources.

On heavily loaded systems, the throughput requirements between the host and the board can cause gaps, called an underrun condition, in the voice record or playback. Failure to maintain pace with the board can also cause underruns in the voice record or playback. Greater compression may be necessary in this case to eliminate the problem.

The ADI service opened on the same context determines what type of voice encoding is supported. For information on attaching the ADI service, refer to the *ADI Service Developer's Reference Manual*.

Each encoding format has a minimum data block size that is called a frame. Frames vary in size and duration depending on the encoding format. For NMS boards, a frame corresponds to 10 or 20 milliseconds of speech, depending on the encoding.

Note: If you are using an AG or CG board, refer to the **adiPlayAsync** function in the *ADI Service Developer's Reference Manual.* A table in the topic lists the DSP files that must be loaded on the AG and CG boards depending on the encoding type. It also lists the valid encoding types that QX boards and PacketMedia HMP processes support.

The encodings refer to the data going to and from the host, typically stored in a voice file. Except for VCE_ENCODE_NMS_64, this host encoding is independent of the line encoding, which is always either mu-law or A-law, depending on how the board is configured when it is initialized.

Note: The ADI service has equivalent encoding formats that begin with ADI_.

EDTX encoding types

EDTX encoding types specify the NMS proprietary EDTX storage format for playing and recording audio to and from packet networks using the native RTP packet payload format. The EDTX encoding types support only files of type VCE_FILETYPE_FLAT or VCE_FILETYPE_VOX. For more information about native play and record, refer to the *ADI Service Developer's Reference Manual*.

EDTX encoding uses frames that include individual headers. The frames can have variable frame sizes and represent variable times. The following functions assume a fixed frame size and are currently not supported with EDTX encoding:

- **vceRead**
- **vceWrite**
- **vceRecord**
- **vcePlay**
- **vceErase**
- **vceConvertMessage**

Refer to *Encoding descriptions* on page 149 and to *Encoding and WAVE information* on page 150.

Messages

A voice file or memory block can contain zero, one, or multiple messages, depending on the file type. A message is a logical, contiguous block of speech identified by an unsigned number. Messages can be created in an open file or memory block by recording or copying to it. Unstructured formats (flat files, memory) contain only message 0 (zero) when first opened. Other messages can be defined by using **vceDefineMessages**.

The file type determines the range of valid message numbers:

File type	Range of valid message numbers
VOX	0 to 32,767
Flat	0 to 65,535
WAVE	0 only
Memory	0 to 65,535

All unused (or deleted) message numbers in the valid range refer to zero-length messages and do not result in an error when referenced.

The special message number VCE_ALL_MESSAGES enables you to operate on all messages in a file or memory block as one message. Use this message number to copy all messages, play all messages, or erase all messages. You cannot record or write to this message number.

Setting up the Natural Access environment

Before calling functions from the Voice Message library, the application must set up the Natural Access environment by performing the following steps:

1. Initialize the Natural Access application.
2. Create event queues and contexts and attach the contexts to an event queue.
3. Open services on each context.

To set up a second Natural Access application that shares a context with the first application:

1. Initialize the Natural Access application.
2. Create event queues.
3. Attach the application to the existing context.

Initializing Natural Access

To register services in the call to **ctaInitialize**, specify the service (VCE) and service manager (VCEMGR). The application can open only the services initialized in the call to **ctaInitialize**. Service managers are dynamic link libraries (DLL) in Windows and shared libraries in UNIX that are linked to the application.

Creating event queues and contexts

After initializing Natural Access, create the event queues and the contexts. To create one or more event queues, call **ctaCreateQueue** and specify the service manager to attach to each queue. When you attach or bind a service manager to an event queue, you make that service manager available to the event queue.

To create a context, call **ctaCreateContext** and provide the queue handle (***ctaqueuehd***) returned from **ctaCreateQueue**. All events for services on the context are received in the specified queue.

ctaCreateContext returns a context handle (***ctahd***). The application supplies the context handle when running Voice Message service functions. Events communicated back to the application are also associated with the context.

Refer to the *Natural Access Developer's Reference Manual* for details on the programming models created by the use of contexts and event queues.

Opening services

To open services on a context, call **ctaOpenServices** and pass a context handle and a list of service descriptors. The service descriptor specifies the name of the service, the service manager, and any service-specific arguments. The Voice Message service has no service-specific arguments.

Linking with the Voice Message service

The Voice Message service contains two components, the Voice Message service interface (*vceapi*) and the Voice Message service implementation (*vcemgr*). When building a new Natural Access application that uses the Voice Message service, link to *vceapi.lib* (under UNIX, *libvceapi.so*). The *vcemgr.lib* (under UNIX, *libvcemgr.so*) is dynamically loaded at runtime.

Earlier releases of Natural Access shipped both the interface and implementation components in a single library, the *vcemgr* library. It is strongly recommended that you modify source files of existing applications to link with *vceapi.lib* (*libvceapi.so*).

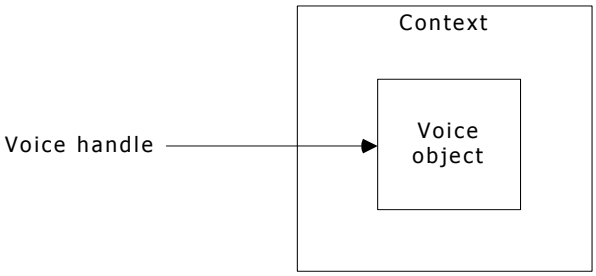
Refer to the *Natural Access Service Writer's Manual* for information about service implementation.

4 Developing applications

Managing voice objects

A voice object is a file or a memory block. To use a voice object, you need a voice handle. A voice handle identifies an open voice object. Use a voice handle with a message number to identify a segment of speech in the object for playing, recording, or editing.

A voice handle is always associated with a context. When you specify a voice handle, the context is implied, as illustrated:



To access the same voice object (for example, the same prompt file),

- Open the voice object on separate contexts, *or*
- Share the open voice object.

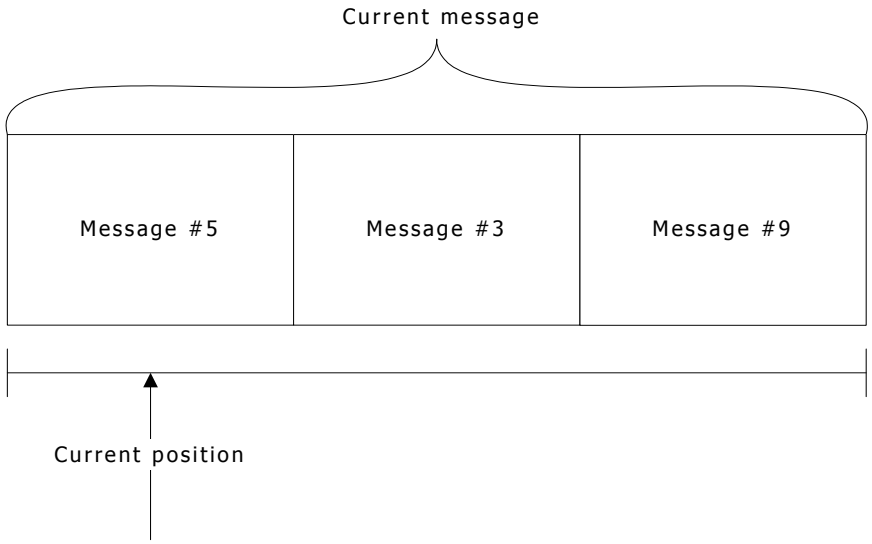
To share an open voice object, use **ctaAttachObject** on a shared context. An action that one application performs on a shared voice object can affect other applications using the voice object. For example, a call to **vceClose** closes the voice object to all applications with an open handle to the voice object. Subsequent calls to the closed voice object return an error. For more information on shared contexts and service objects, refer to the *Natural Access Developer's Reference Manual.*

Voice object functions

To...	Use...
Access an existing file for play or record	**vceOpenFile**
Create a new voice file	**vceCreateFile**
Access a range of memory	**vceOpenMemory** **vceCreateMemory**
Access a file (or device) that is already opened	**vceAssignHandle**
Release a voice handle, close the associated file if **vceOpenFile** or **vceCreateFile** obtained the handle, and free the memory that **vceCreateMemory** allocated.	**vceClose**

Defining current message

When you play a message or a list of messages, the message or message list becomes the current message for pausing, resuming, and repositioning. The current message is associated with the context handle. The current message has a current position, as illustrated:



Using current position

To...	Use...
Remove a specified number of milliseconds of data at the current position in the current message	**vceErase**
Start playing from the current position in the current message	**vcePlay**
Read data at the current position in the current message	**vceRead**
Start recording at the current position in the current message	**vceRecord**
Set the current position in the current message	**vceSetPosition**
Write data at the current position in the current message	**vceWrite**

Using current message

To...	Use...
Copy a message from one voice object to another	**vceConvertMessage**
Create or replace a message in a destination object with a copy of a message in a source object	**vceCopyMessage**
Remove an entire message from a specified voice object	**vceEraseMessage**
Play a sequence of messages with no delay between messages	**vcePlayList**
Play one message from a specified voice object	**vcePlayMessage**
Copy a message from one voice object to another	**vceRecordMessage**
Define the current message to be a list of messages	**vceSetCurrentList**
Define the current message	**vceSetCurrentMessage**

If a voice object containing a message in the current message (list) is closed, a current message no longer exists. Play or record is stopped, if it is active.

Note: The current message is not affected when a call hangs up.

Playing

The Voice Message service provides two types of playing functions:

- The message-oriented functions **vcePlayMessage** and **vcePlayList** operate on a specific message or a list of messages. They take a voice handle and message number, or a list of message numbers. Playing starts from the beginning of the first message.
- The position-oriented function **vcePlay** operates on the current message in the specified context. Playing starts from the current position in the most recently accessed message or list.

Use the ADI service function **adiPlayAsync** or **adiStartPlaying** for direct access to buffers. Refer to the *ADI Service Developer's Reference Manual* for more information on ADI functions.

The *vceplay* demonstration program demonstrates playing messages from one or more voice files.

This topic presents an overview of:

- Play functions
- Playing a sequence of messages
- Play completion reasons
- Adjusting volume and speed
- Play parameters

Play functions

To...	Use...
Start playing from the current position in the current message	**vcePlay**
Start playing from the beginning of the first message in a list of messages	**vcePlayList**
Play one message from a specified voice object	**vcePlayMessage**
Change the current play speed to a specified rate	**vceSetPlaySpeed**
Change the volume (loudness) of message playback by adjusting the amplification, or gain, applied to the message being played	**vceSetPlayGain**

Playing a sequence of messages

A list is one or more messages. Use **vcePlayList** to play a sequence of messages with no delays between each message. **vcePlayList** enables you to play a prompt string that **vceBuildPromptList** converted to a sequence of message numbers. All messages in the list must have the same encoding. If the messages are in different files, use **vceSetCurrentList** followed by **vcePlay**.

Play completion reasons

The following table lists the Voice Message service reason codes returned when play terminates. The value field of the VCEEVN_PLAY_DONE event states the reason.

Note: If play fails, the value field contains an error code.

If...	The DONE event contains...
The end of the current message or list is reached	CTA_REASON_FINISHED
The time limit in **vcePlay** is reached	CTA_REASON_TIMEOUT
Play is stopped by calling **vceStop**	CTA_REASON_STOPPED
A touch tone digit is received and the corresponding bit in the DTMF abort parameter is set	CTA_REASON_DIGIT
The call is ended	CTA_REASON_RELEASED
The attached device service does not support the encoding format of the playing object	CTAERR_FUNCTION_NOT_AVAIL

Adjusting volume and speed

You can set the volume (**vceSetPlayGain**) and speed (**vceSetPlaySpeed**) of play at any time. The new volume (gain or amplification) and the new speed are stored in the current context. If play is currently active, the change takes effect immediately. If you want a gain or speed change to carry over to the next play function, set the play gain or speed parameter on the next play to VCE_CURRENT_VALUE. The PacketMedia HMP process does not support play speed control.

You can also adjust the playing speed for some encoding formats. Speed control is available for the NMS ADPCM (VCE_ENCODE_NMS_xx) and the OKI ADPCM encoding formats (VCE_ENCODE_OKI_xx).

To increase the playing speed, increase the value of the ***maxspeed*** play parameter from a default value of 100. When play is started with a higher value of ***maxspeed***, the necessary DSP resources are allocated to support increased speed. You can start play with a fast speed (up to the value of the ***maxspeed*** play parameter) by changing the value of the speed parameter.

Note: Starting play with ***maxspeed*** greater than 100 requires more DSP resources than required for playing at normal speed. To determine whether the AG boards and the configuration can support the increased speed, refer to the *NMS OAM System User's Manual* or the board-specific installation and developer's manual.

Use **vceGetContextInfo** to get the current gain and speed values.

Play parameters

The play functions take an optional pointer to a parameter structure. If you pass NULL, the parameters take default values. Refer to *VCE_PLAY_PARMS* on page 134 for more information about the play parameters.

Recording

The Voice Message service provides two types of recording functions:

- The message-oriented function **vceRecordMessage** operates on a specific message number. This function creates a new message, replacing the existing message, if any.
- The position-oriented function **vceRecord** operates on the current message in the specified context. Recording starts from the current position in the most recently accessed message.

Use the ADI functions **adiRecordAsync** or **adiStartRecording** for direct access to buffers. Refer to the *ADI Service Developer's Reference Manual* for more information on the ADI service.

The *vcerec* demonstration program demonstrates copying messages from one voice file to another.

Note: Playing and recording are mutually exclusive on a given context. Use the ADI service to perform simultaneous play and record.

This topic presents the following information:

- Record functions
- Record completion reasons
- Message resizing
- Record parameters

Record functions

Recording optionally starts with a beep tone. The recording functions have a time limit. Silence detected during recording can optionally abort the recording.

To...	Use...
Start recording at the current position in the current message	**vceRecord**
Record a message into a specified voice object	**vceRecordMessage**

Record completion reasons

The following table lists the Voice Message service reason codes returned when record terminates. The value field of the VCEEVN_RECORD_DONE event states the reason.

Note: If record fails, the value field contains an error code.

If...	The DONE event contains...
The time limit in **vceRecord** is reached	CTA_REASON_TIMEOUT
No more space is available in the voice object, or recording in overwrite mode reached the end of the existing message	CTA_REASON_FINISHED
Silence is detected at the beginning of recording	CTA_REASON_NO_VOICE
Silence occurs after some energy is recorded	CTA_REASON_VOICE_END
Record is stopped at any time by calling **vceStop**	CTA_REASON_STOPPED
A touch tone digit is received and the corresponding bit in the DTMF abort parameter is set	CTA_REASON_DIGIT
The call is ended	CTA_REASON_RELEASED
The attached device service does not support the encoding format of a recording object	CTAERR_FUNCTION_NOT_AVAIL

Message resizing

When **vceRecordMessage** replaces a message, the new message can be larger or smaller than the previous one. The ability to resize a message depends on the object and the position of the message in the object. A message in a VOX file or a message that is located at the end of a flat (unformatted) file can grow without limit until the disk is full. A file can physically shrink if a smaller message replaces the message at the end of the file (the shrinkage occurs when the file is closed). A message in a memory block is limited to the size of the memory specified with **vceOpenMemory** or **vceCreateMemory**.

Usually only one message exists in a flat file or a memory block. If more than one message exists in a flat file or memory block and a message in the middle is replaced, the new message cannot be larger than the existing message. **vceRecordMessage** ends with CTA_REASON_FINISHED when the limit is reached.

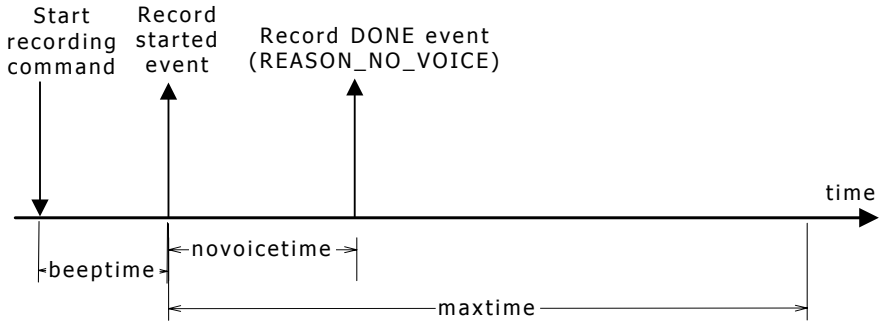
Record parameters

The record functions take an optional pointer to a parameter structure for controlling beep tone, silence detection, record gain (including optional automatic gain control), and DTMF abort. If you pass NULL, the parameters take default values. Refer to *VCE_RECORD_PARMS* on page 135 for more information about the record parameters.

Note: Automatic gain control is an algorithm applied to incoming speech (recording) before compression and storage that is designed to keep the amplitude of the stored speech at a target level.

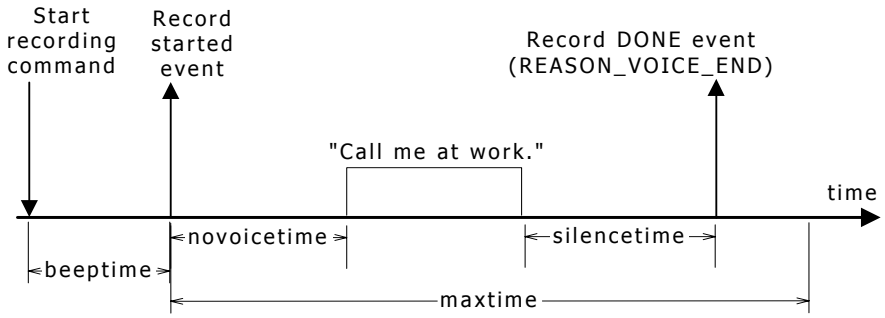
Record termination - no voice

The following illustration shows record termination - no voice:



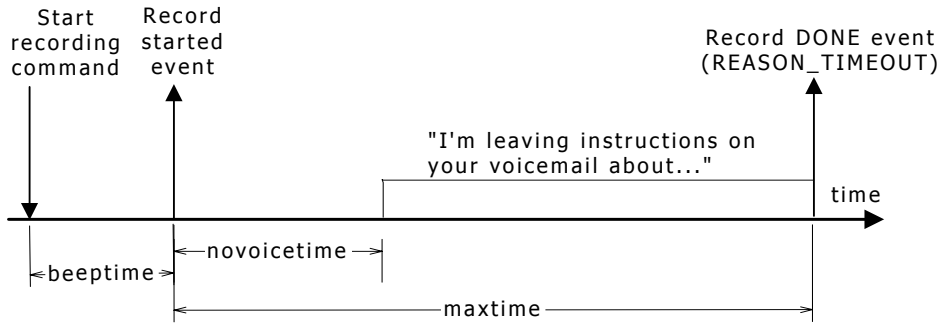
Record termination - voice end

The following illustration shows record termination - voice end:



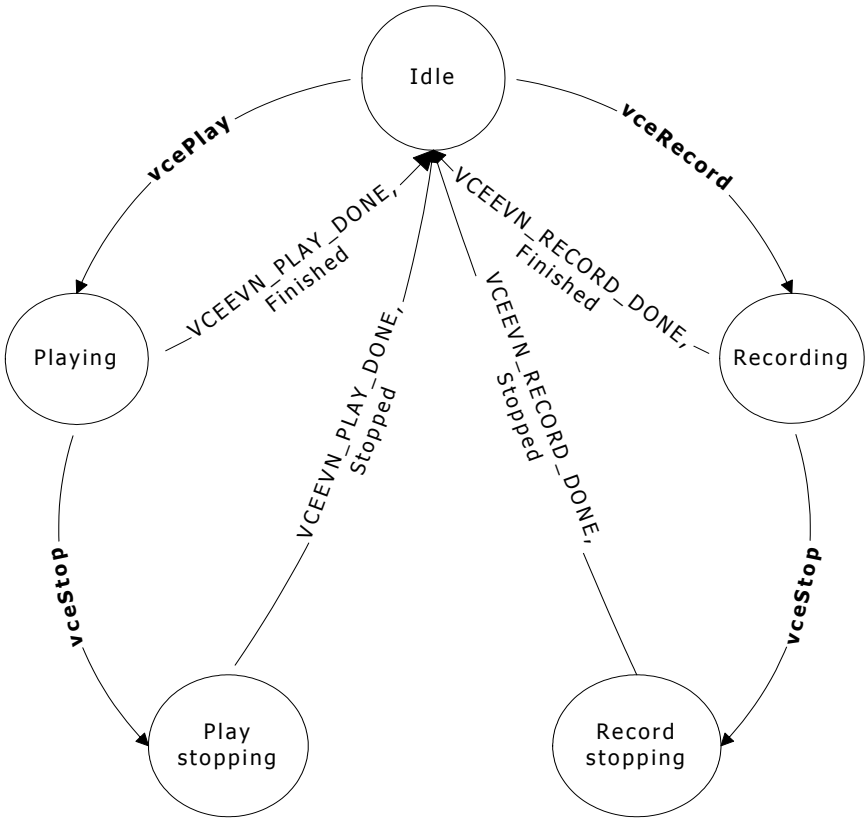
Record termination - timeout

The following illustration shows record termination - timeout:



Play and record states

The following illustration shows the play and record states:



The following table describes the play and record states:

State	Description
Idle	The play and record functions are not active.
Playing	When the application initiates playing by launching a play function (**vcePlay**, **vcePlayList**, or **vcePlayMessage**), the Voice Message service is in the playing state.
Recording	When the application initiates recording by launching a record function (**vceRecord** or **vceRecordMessage**), the Voice Message service is in the recording state.
Play stopping	The application initiates stopping the play by launching **vceStop**. The Voice Message service generates a VCEEVN_PLAY_DONE event and returns to the idle state.
Record stopping	The application initiates stopping the record by launching **vceStop**. The Voice Message service generates a VCEEVN_RECORD_DONE event and returns to the idle state.

The play or record state remains active until either:

- A VCEEVN_PLAY_DONE event or a VCEEVN_RECORD_DONE event is received.
- **vceStop** is called.

Stopping

Call **vceStop** to stop play or record. Stopping play or record results in a DONE event. The value field of the DONE event contains CTA_REASON_STOPPED.

Note: CTA_REASON_STOPPED is also returned if play or record is stopped when the voice object being played or recorded is closed with **vceClose**. If you are playing a list, play is stopped if an object containing any message in the list is closed.

Using DTMF to terminate play or record

By default, play or record terminates when any DTMF digit is detected. The play parameter structure and the record parameter structure provide a selective DTMF abort mask specifying which DTMF digits terminate the function.

The DTMF abort mask is a 16-bit entity where each bit corresponds to a specific key on the telephone keypad. Setting a bit in the mask causes the voice function to terminate if that particular key is entered. The DTMF abort mask corresponds to the DTMF telephone keys as shown:

	Most significant bit											**Least significant bit**				
Bit position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DTMF key	D	C	B	A	#	*	9	8	7	6	5	4	3	2	1	0

For example, if the abort mask is set to 0x03FF, the playing and recording functions terminate if the remote party enters any digit from 0 through 9. The *vcedef.h* include file contains #defines (VCE_DTMF_xxx) for each digit and for certain digit groups.

Retrieving context status

vceGetContextInfo returns the current Voice Message service status of the specified context, including information about the current, or most recent, play or record operation.

The status block includes the following information:

- Current position in milliseconds relative to the beginning of the current message.
- Reason last play or record ended (0 initially and while function is active).
- Current or last function (play, record, or none).
- Underruns (cumulative for the lifetime of the context).
- Auxiliary error code (for example, an OS error from file access).
- Current value of play gain. This value is used for the next playing function if the play gain parameter is set to VCE_CURRENT_VALUE. Otherwise, this value is set to the value of the play gain parameter when playing starts. Use **vceSetPlayGain** to change the value.
- Current value of play speed. This value is used for the next playing function if the play speed parameter is set to VCE_CURRENT_VALUE. Otherwise, this value is set to the value of the play speed parameter when playing starts. Use **vceSetPlaySpeed** to change the value.
- Number of messages in the current message. Use **vceGetCurrentList** to retrieve the actual message list.
- Encoding of the current message.
- Frame size of the encoding.
- Frame duration of the encoding.

The millisecond and position values are updated as each buffer is submitted internally for play or completed for record.

Note: When play ends, the values can be adjusted downward if not all data submitted to the device was actually played.

Retrieving voice object information

Use the following functions to retrieve voice object information:

To get the...	Use...
File type, open mode, or associated context of an open voice object	**vceGetOpenInfo**
Size of a specified message in milliseconds	**vceGetMessageSize**
Highest message number currently in use in the voice object	**vceGetHighMessageNumber**
Lowest unused message number for the voice object	**vceGetUniqueMessageNumber**

The *vceinfo* demonstration program displays information about a voice file.

Editing voice messages

The editing functions enable you to access and modify data in a voice object without actually playing or recording. You can open a file and edit it without needing a voice board.

The Voice Message service provides functions for the following editing tasks:

- Copying messages and converting encoding or amplitude
- Erasing messages
- Reading and writing messages

Copying and converting messages

Use the following functions to copy and convert messages:

To...	Use...
Create or replace a message in a destination object with a copy of a message in a source object when the encoding is the same	**vceCopyMessage**
Create or replace a message in a destination object with a copy of a message in a source object when the encoding is different or you want to change the amplitude	**vceConvertMessage**

The *vcecopy* demonstration program demonstrates copying messages from one voice file to another.

Erasing messages

Use the following functions to erase messages:

To...	Use...
Remove an entire message from a specified voice object by changing the message size to zero (0)	**vceEraseMessage**
Remove a specified number of milliseconds of data at the current position in the current message	**vceErase**

Reading and writing messages

Use the following functions to read and write messages:

To...	Use...
Read data at the current position in the current message	**vceRead**
Write data at the current position in the current message	**vceWrite**

Converting text strings to message lists

The Voice Message service prompt builder contains functions for converting text strings to message lists. The prompt builder enables an application to announce dates, times, numbers, and monetary amounts.

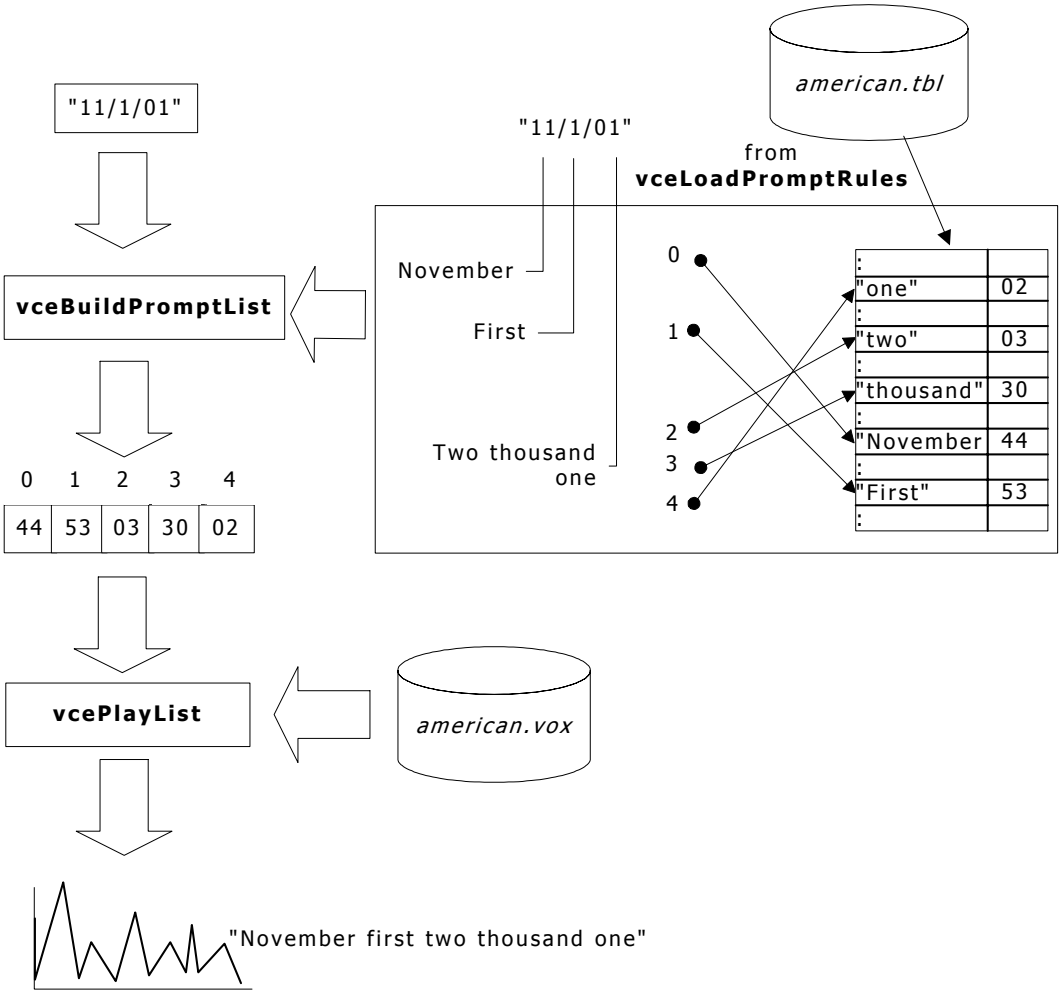
Use the following prompt builder functions to convert text strings to message lists:

To...	Use...
Build a list of message numbers from specified text	**vceBuildPromptList**
Load a prompt rules table into memory	**vceLoadPromptRules**
Release resources associated with a prompt rules table	**vceUnloadPromptRules**

How the prompt builder works

The prompt builder uses a prompt rules table to determine how text strings translate into voice messages. An associated voice file contains the messages corresponding to these rules. For example, the text string 10 is translated to the voice message ten.

The following illustration shows how the prompt builder works. A text string containing data, such as 11/1/01, is passed to **vceBuildPromptList** which then builds a list of messages to be played from a standard message library. The output list of message numbers is passed to **vcePlayList**.



The standard prompt rules table (*american.tbl*) supplied with the prompt builder can play dates, times, numbers, and dollar amounts. The compiled table and associated voice message file is different for each spoken language supported. Additional phrase types can be supported by adding to the rules provided, or by developing a new set of rules for different spoken languages. Refer to *Creating a prompt rules table* on page 142 for information on adding or changing prompt rules.

5 Function summary

Voice object functions

Function	Synchronous/ Asynchronous	Description
vceOpenFile	Synchronous	Opens a voice file and returns a voice handle. Use this function to open the file for play only or for both play and record.
vceCreateFile	Synchronous	Creates a new file of a specified type.
vceOpenMemory	Synchronous	Assigns a voice handle to a range of memory.
vceCreateMemory	Synchronous	Assigns a voice handle to a memory block.
vceAssignHandle	Synchronous	Assigns a voice handle to an open file.
vceClose	Synchronous	Closes a voice file or other voice object.

Play functions

Function	Synchronous/ Asynchronous	Description
vcePlay	Asynchronous	Starts playing from the current position in the current message. Use this function to resume playing after stopping a message in the middle.
vcePlayList	Asynchronous	Starts playing from the beginning of the first message in a list of messages. This function builds a message from a library of words or phrases contained in a voice file. This function enables you to play a sequence of messages with no delay between messages.
vcePlayMessage	Asynchronous	Plays one message from a specified voice object.
vceSetPlayGain	Synchronous	Changes the volume (loudness) of message playback by adjusting the amplification, or gain, applied to the message being played.
vceSetPlaySpeed	Synchronous	Changes the current play speed to a specified rate.

vceplay demonstrates playing voice files.

Record functions

Function	Synchronous/ Asynchronous	Description
vceRecord	Asynchronous	Starts recording at the current position in the current message.
vceRecordMessage	Asynchronous	Records a message in an open voice object.

vcerec demonstrates recording voice files.

Stop function

vceStop stops the currently active play or record. This function is asynchronous.

Query functions

Function	Synchronous/ Asynchronous	Description
vceGetCurrentList	Synchronous	Returns the list of messages contained in the current message.
vceGetCurrentSize	Synchronous	Retrieves the size in milliseconds of the current message.
vceGetEncodingInfo	Synchronous	Returns information about a voice encoding.
vceGetHighMessageNumber	Synchronous	Returns the highest message number in use in the voice object.
vceGetMessageSize	Synchronous	Returns the size in milliseconds of the specified message.
vceGetOpenInfo	Synchronous	Returns information about an open voice object.
vceGetContextInfo	Synchronous	Returns the current Voice Message service status of the specified context.
vceGetUniqueMessageNumber	Synchronous	Returns the lowest available (empty) message in the file.
vceGetWaveInfo	Synchronous	Returns wave header information for a given encoding value.

vceinfo demonstrates displaying information about a voice file.

Current message functions

Function	Synchronous/ Asynchronous	Description
vceSetCurrentList	Synchronous	Defines the current message to be a list of messages.
vceSetCurrentMessage	Synchronous	Defines the current message.
vceSetPosition	Synchronous	Sets the current position in the current message.

Voice message edit functions

Function	Synchronous/ Asynchronous	Description
vceConvertMessage	Synchronous	Copies a message from one voice object to another. This function translates the encoding, the amplitude, or both the encoding and the amplitude of the message.
vceCopyMessage	Synchronous	Creates or replaces a message in a destination object with a copy of a message in a source object. The source and destination objects must have the same encoding.
vceErase	Synchronous	Removes a specified number of milliseconds of data at the current position in the current message.
vceEraseMessage	Synchronous	Removes an entire message from a specified voice object.
vceRead	Synchronous	Reads data at the current position in the current message.
vceWrite	Synchronous	Writes data at the current position in the current message.

vcecopy demonstrates copying voice messages from one file to another.

Text string to message lists conversion functions

Function	Synchronous/ Asynchronous	Description
vceBuildPromptList	Synchronous	Builds a list of message numbers based on the specified prompt rule table entry point and variable text.
vceLoadPromptRules	Synchronous	Reads the specified prompt rules table into memory.
vceUnloadPromptRules	Synchronous	Releases resources associated with the specified prompt rules.

Message text functions

Message text is a descriptive string or other data that can be attached to a message in a NMS VOX file. Use the following functions to write and copy message text:

Function	Synchronous/ Asynchronous	Description
vceCopyMessageText	Synchronous	Copies message text from one voice message to another voice message.
vceReadMessageText	Synchronous	Reads the message text for a specified voice message in an NMS VOX file.
vceWriteMessageText	Synchronous	Writes message text for a specified voice message in an NMS VOX file.

Miscellaneous voice functions

Function	Synchronous Asynchronous	Description
vceDefineMessages	Synchronous	Defines message boundaries in a flat (unformatted) file or memory block.
vceSetWaveInfo	Synchronous	Loads a new entry in the WAVE encoding table.

6 Function reference

Using the function reference

This section provides an alphabetical reference to the Voice Message service functions. A typical function includes:

Prototype	The prototype is followed by a list of the function's arguments. Data types include: • WORD (16-bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is shown.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation. Refer to the *Alphabetical error summary* on page 127 for a list of errors that the Voice Message service functions return.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. Additional information such as reason codes and return values appears in the value field of the event. If there are no events listed, the function is synchronous.

vceAssignHandle

Assigns a voice handle to an open file.

Prototype

DWORD **vceAssignHandle** (CTAHD ***ctahd***, int ***filedes***, unsigned ***encoding***, VCEHD ****vh***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext**.
filedes	File descriptor returned by an operating system open function.
encoding	Encoding of all messages in the file.
vh	Pointer to a returned voice handle.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_INVALID_HANDLE	***filedes*** is invalid.
CTAERR_NOT_IMPLEMENTED	This error is returned when an application that is not using the inproc default server launches **vceAssignHandle**.
VCEERR_UNSUPPORTED_ENCODING	Neither the Voice Message service nor the associated play or record service, if any, support the specified encoding.

Details

vceAssignHandle assigns a voice handle to an open file or to some other data stream such as stdin.

When **vceClose** closes the voice handle, ***filedes*** is not automatically closed.

The Voice Message service marks the voice handle that **vceAssignHandle** returns as thread-safe so that multiple voice handles can be assigned to the same open file. For example, multiple contexts running in separate threads can safely access a common prompt file without incurring the overhead of multiple file opens.

The record or edit function returns a write failed error code if the file has not been opened for writing.

Note: In UNIX, you can use the file descriptor that the C **open** function or the C **fileno** function returns. In Windows, because some compilers do not return the operating system file descriptor, you can use file descriptors returned only by system calls (for example, **CreateFile**).

Use **vceAssignHandle** only if the application is using the inproc default server.

See also

vceCreateFile, **vceOpenFile**, **vceOpenMemory**

Example

```
/* Play from standard input */
#if defined _WIN32 && defined _MSC_VER
    #include <io.h>
    #define STDIN_FILENO _get_osfhandle (0)
#endif

extern CTAHD      Ctahd;
extern CTAQUEUEHD CtaQueueHd;
void myplaystdin( unsigned encoding )
{
    VCEHD     vh;
    CTA_EVENT event;

    vceAssignHandle (Ctahd, STDIN_FILENO, encoding, &vh );
    vcePlayMessage (vh, 0, NULL);
    do
    {
        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
    } while (event.id != VCEEVN_PLAY_DONE); /* Ignore other events */
}
```

vceBuildPromptList

Builds a list of message numbers from a text string.

Prototype

DWORD **vceBuildPromptList** (VCEPROMPTHD ***prompthandle***, unsigned ***method***, char ****text***, unsigned ***list[]***, unsigned ***maxcount***, unsigned ****actualcount***)

Argument	Description
prompthandle	Handle of the prompt table obtained through **vceLoadPromptRules**.
method	Table-specific method used for deciphering the specified text string.
text	Pointer to the string of text to translate.
list	Array to receive message numbers to use with **vcePlayList**.
maxcount	Maximum number of message numbers that can be put in ***list***.
actualcount	Pointer to the returned number of messages. This number can be passed to **vcePlayList**.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_SIZE	***maxcount*** is too small to contain the output list.
CTAERR_INVALID_HANDLE	***prompthandle*** is not a valid handle to an open prompt rules table.
CTAERR_SVR_COMM	Server communication error.
VCEERR_BAD_PROMPT_COMMAND	Invalid ***method*** is entered (a value other than 0, 1, 2, 3, or 4) or an error exists in the rules table to which ***prompthandle*** refers.
VCEERR_PROMPT_BUILD_FAIL	Error exists in the prompt rules table to which ***prompthandle*** refers.

Details

vceBuildPromptList builds a list of message numbers based on the specified prompt rule table entry point and variable text. **vceBuildPromptList** only creates a message list. The application must open the corresponding voice file separately.

list is built by translating the text string into specific messages in the associated VOX file. The array must be large enough to contain the maximum number of message numbers in the list. For example, $1.25 is translated into six different messages (one, dollar, and, twenty, five, cents) using the American rules.

The standard rules provided in *american.tbl* are translated using the following methods:

Method	Description
0	Based on the characters in the string, **vceBuildPromptList** decides if it is for date, time, number, monetary, day-of-week, or month utterances.
1	Any number up to 999 trillion with or without commas. Can also handle numbers with decimal fractions.
2	Date in mm/dd, mm/dd/yy, or mm/dd/yyyy formats.
3	Time in hh:mm or hh:mm:ss formats. Hours must be in 24 hour time, for example, from 0 to 23.
4	Monetary amounts with or without commas every three digits.

See also

vceUnloadPromptRules

Example

```
/* Demonstrates playing a text string */

char *text = "$1,234,567";
extern CTAHD CtaHd;
extern CTAQUEUEHD CtaQueueHd;

void mytestprompt ()
{
    VCEHD    vh;
    VCEPROMPTHD prompthandle;
    unsigned msglist[50];
    unsigned count;

    /* Open the associated voice file */
    vceOpenFile (CtaHd, "american.vox", VCE_FILETYPE_VOX,
                                        VCE_PLAY_ONLY, 0, &vh);

    /* Load the American rules */
    vceLoadPromptRules (CtaHd, "american.tbl", &prompthandle );

    /* Translate the string into a list of messages in american.vox */
    vceBuildPromptList (prompthandle, 0, text, msglist,
                        sizeof msglist/sizeof msglist[0], &count);

    /* Play the messages */
    vcePlayList (vh, msglist, count, NULL);
    do
    {

        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
    } while (event.id != VCEEVN_PLAY_DONE); /* Ignore other events */

    /* Close the rules table */
    vceUnloadPromptRules (prompthandle);

    /* Close the associated voice file */
    vceClose (vh);
}
```

vceClose

Closes a voice object.

Prototype

DWORD **vceClose** (VCEHD ***vh***)

Argument	Description
vh	Handle of an open voice object.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_SVR_COMM	Server communication error.

Details

vceClose closes a voice file or a memory block. If the object is a memory block, the memory is not freed. If **vceAssignHandle** creates the object, the application must close the associated file descriptor.

Note: If play or record is currently active on a message in the voice object, the play or record function is terminated. The reason in the DONE event is CTA_REASON_STOPPED.

Refer to *Managing voice objects* on page 15 for more information.

Shared voice objects

A call to **vceClose** closes the voice object to all applications with an open handle to the voice object. Subsequent calls to the voice object return an error. For example, an attempt to play a message using the handle to the voice object results in VCEERR_NO_MESSAGE being returned.

The application launching **vceClose** must notify the other applications that the voice object is closing. Upon notification or upon receiving an error message, applications sharing the voice object must call **ctaDetachObject** to deallocate local resources associated with the shared voice object.

Note: When an invalid ***vh*** calls **vceClose**, the function automatically destroys the local handle. In this case, the application does not need to launch **ctaDetachObject**.

For more information on shared service objects, refer to the *Natural Access Developer's Reference Manual.*

See also

vceCreateFile, **vceOpenFile**, **vceOpenMemory**

Example

```
/* Close an open memory block and free the memory */

void myclosememory (VCEHD vh, BYTE *address)
{
    free (address);          /* OK to free first if not playing.*/
    vceClose (vh);
}
```

vceConvertMessage

Copies a message and translates its encoding.

Prototype

DWORD **vceConvertMessage** (VCEHD ***srcvh***, unsigned ***srcmsg***, VCEHD ***destvh***, unsigned ***destmsg***, int ***gain***)

Argument	Description
srcvh	Voice handle of source file or memory block.
srcmsg	Source message number.
destvh	Voice handle of destination file or memory block.
destmsg	Destination message number.
gain	Amplification if positive or attenuation if negative.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	***gain*** is less than -24 or greater than 24.
CTAERR_DISK_FULL	There is not enough room on the disk to complete the write operation. No data was written.
CTAERR_FILE_READ_FAILED	File system error.
CTAERR_FILE_WRITE_FAILED	File system error.
CTAERR_FUNCTION_ACTIVE	Playing or recording is active on the context.
CTAERR_INVALID_HANDLE	A voice handle is invalid, or the source and destination message numbers belong to different contexts.
CTAERR_SVR_COMM	Server communication error.
VCEERR_CONVERSION_FAILED	An internal error occurred in the conversion library.
VCEERR_INVALID_MESSAGE	Source message number or destination message number is invalid for the file type.
VCEERR_NO_SPACE	Insufficient room in the destination message.
VCEERR_OUT_OF_INDICES	No free header entries in the destination VOX file.
VCEERR_PLAY_ONLY	Destination voice file is not open for record.
VCEERR_UNSUPPORTED_ENCODING	Source or destination encoding format is not supported by the conversion function.

Details

vceConvertMessage copies a message from one voice object to another, or to another message in the same voice object. The encoding of the message is translated as needed from the source to the destination. If the ***gain*** argument is not zero (0), attenuation or amplification is applied, depending on whether ***gain*** is negative or positive.

Source and destination must belong to the same context. The destination message becomes the current message in the context.

EDTX encoding types are not supported.

To copy and translate all messages from the source to the destination, set the source message number (***srcmsg***) and the destination message number (***destmsg***) to VCE_ALL_MESSAGES. This setting replaces all messages in the destination, not just messages that exist in the source.

The ***gain*** must be in the range of -24 to +24. Otherwise, CTAERR_BAD_ARGUMENT is returned.

Depending on the type of conversion required, a long message can take many seconds to convert. During this time, the context and the queue associated with the message is locked. You can open a separate queue for performing conversions when other Natural Access functions are used in the same process.

Converting from one mu-law, A-law, or 8 kss PCM format to another is relatively fast. Converting between PCM and ADPCM, or converting from one sample rate to another, takes more time.

The following example shows the possible conversions and indicates the relative amount of processing required:

	a	b	c	d	e	f	g	h	i	j	k	l
a) NMS 16/24/32	2											
b) NMS 64	1	0										
c) mu/A-law	1	0	0									
d) PCM 8 kss	1	0	0	0								
e) PCM 11 kss	2	1	1	1	0							
f) PCM 22 kss	2	1	1	1	1	0						
g) PCM 44 kss	2	1	1	1	1	1	0					
h) OKI 24	x	x	x	x	x	x	x	x				
i) OKI 32	2	1	1	1	2	2	2	x	0			
j) G.726	2	1	1	1	2	2	2	x	2	0		
k) IMA 24	x	x	x	x	x	x	x	x	x	x	x	
l) IMA 32	2	1	1	1	2	2	2	x	2	2	x	0

Note: kss = kilo-samples per second (mono or stereo, 8-bit or 16-bit).

Key:

Value	Description
0	No conversion, or conversion between A-law, mu-law, and 8 kHz PCM.
1	Either ADPCM <--> PCM conversion or sample rate conversion.
2	Either two ADPCM <--> PCM conversions, or ADPCM <--> PCM plus sample rate conversion.
x	Not supported.

See also

vceCopyMessage

Example

```
/* Convert a Wave file of any encoding to a VCE file with NMS encoding */
void myWaveToVce (CTAHD ctahd, char *wavefile, char *vcefile)
{
VCEHD srcvh, destvh;

vceOpenFile (ctahd, wavefile, VCE_FILETYPE_WAVE, VCE_PLAY_ONLY, 0, &srcvh);

vceCreateFile (ctahd, vcefile, VCE_FILETYPE_FLAT,
               VCE_ENCODE_NMS_24, NULL, &destvh);

vceConvertMessage (srcvh, 0, destvh, 0, 0);

    vceClose (srcvh);
    vceClose (destvh);
}
```

vceCopyMessage

Copies a message from one voice object to another voice object.

Prototype

DWORD **vceCopyMessage** (VCEHD ***srcvh***, unsigned ***srcmsg***, VCEHD ***destvh***, unsigned ***destmsg***)

Argument	Description
srcvh	Voice handle of source file or memory block.
srcmsg	Source message number.
destvh	Voice handle of destination file or memory block.
destmsg	Destination message number.

Return values

Return value	Description
SUCCESS	
CTAERR_DISK_FULL	There is not enough room on the disk to complete the write operation. No data was written.
CTAERR_FILE_READ_FAILED	File system error.
CTAERR_FILE_WRITE_FAILED	File system error.
CTAERR_FUNCTION_ACTIVE	Playing or recording is active on the context.
CTAERR_INVALID_HANDLE	A voice handle is invalid, or source and destination message numbers belong to different contexts.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Source or destination message number is invalid for file format type.
VCEERR_NO_SPACE	Insufficient room in the destination message.
VCEERR_OUT_OF_INDICES	No free header entries in the destination VOX file.
VCEERR_PLAY_ONLY	Destination voice file is not open for record.

Details

vceCopyMessage copies the ***srcmsg*** in the voice object denoted by ***srcvh*** to the destination message in the voice object denoted by ***destvh***. It creates or replaces the message in a destination voice object with a copy of the message in the source voice object. If ***destmsg*** already exists, it is replaced.

Source and destination voice objects must belong to the same context. The destination voice object must be open for record. The destination message becomes the current message in the context. ***srcvh*** and ***destvh*** can be the same.

Source and destination message numbers must be valid for their respective file types.

Note: If the source message has zero (0) size, the destination size is set to zero (0). This is equivalent to erasing the destination message.

To copy all messages from the source to the destination, set source and destination message numbers to VCE_ALL_MESSAGES. This setting replaces all messages in the destination voice object, not just messages that exist in the source voice object.

To copy message text associated with a voice message, use **vceCopyMessageText**.

See also

vceConvertMessage

Example

```
/*
* Copy VOX file to a new one. The copy operation removes any fragmentation that
* might exist as a result of editing.
*/

void myCopyVoxFile (CTAHD ctahd, char *destfile, char *srcfile)
{
    VCEHD           srcvh, destvh;
    VCE_OPEN_INFO   openinfo;
    VCE_CREATE_VOX  voxcreate;
    unsigned        highmsg;

     /* Open file, get encoding */
    vceOpenFile (ctahd, srcfile, VCE_FILETYPE_VOX, VCE_PLAY_ONLY, 0, srcvh);
    vceGetOpenInfo (srcvh, &openinfo, sizeof openinfo);

    /* Create a file with enough indices to accommodate growth */
    vceGetHighMessageNumber (srcvh, &highmsg);
    voxcreate.maxindex = 2 * highmsg + 100;
    voxcreate.size     = sizeof voxcreate;
    vceCreateFile (ctahd, destfile, VCE_FILETYPE_VOX,
                   openinfo.encoding, &voxcreate, &destvh);

    /* Copy all messages */
    vceCopyMessage(srcvh, VCE_ALL_MESSAGES, destvh, VCE_ALL_MESSAGES);

    /* Close files */
    vceClose(destvh);
    vceClose(srcvh);

}
```

vceCopyMessageText

Copies the associated message text data from one voice message to another voice message.

Prototype

DWORD **vceCopyMessageText** (VCEHD ***srcvh***, unsigned ***srcmsg***, VCEHD ***destvh***, unsigned ***destmsg***)

Argument	Description
srcvh	Voice handle of the source VOX file.
srcmsg	Source message number.
destvh	Voice handle of the destination VOX file.
destmsg	Destination message number.

Return values

Return value	Description
SUCCESS	
CTAERR_DISK_FULL	There is not enough room on the disk to complete the write operation. No data was written.
CTAERR_FILE_READ_FAILED	File system error.
CTAERR_FILE_WRITE_FAILED	File system error.
CTAERR_INVALID_HANDLE	A voice handle is invalid, or source and destination message numbers belong to different contexts.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Source message number or destination message number is not in the valid range for VOX files (0 to 32767).
VCEERR_OUT_OF_INDICES	No free header entries in the destination VOX file.
VCEERR_PLAY_ONLY	Destination voice file is not open for record.
VCEERR_WRONG_FILE_TYPE	Source or destination is not a VOX file.

Details

vceCopyMessageText copies the message text for message ***srcmsg*** in the VOX file denoted by ***srcvh*** to message number ***destmsg*** in the VOX file denoted by ***destvh***. If message text for ***destmsg*** already exists, it is replaced.

Source and destination voice handles must belong to the same context. They can be the same handle.

To copy all message text from the source file to the destination file, set the source message and the destination message to VCE_ALL_MESSAGES. This setting replaces all message text in the destination file.

vceCopyMessageText does not affect the current message for the context.

To copy a voice message, use **vceCopyMessage**.

See also

vceReadMessageText, **vceWriteMessageText**

Example

```
/*
* Copy a message and its associated text from one .VOX file to another */
void myCopyPrompt (VCEHD destvh, VCEHD srcvh, unsigned message)
{
    vceCopyMessage    (srcvh, message, destvh, message);
    vceCopyMessageText(srcvh, message, destvh, message);
}
```

vceCreateFile

Creates a new voice file.

Prototype

DWORD **vceCreateFile** (CTAHD ***ctahd***, char ****filename***, unsigned ***filetype***, unsigned ***encoding***, void ****fileinfo***, VCEHD ****vh***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
filename	Pointer to the name of the file to create. If the name does not include a complete path, the file is created in the current working directory of either the application or the Natural Access Server (*ctdaemon*), depending on whether the application is using the inproc default server or the localhost default server.
filetype	Acceptable values are: VCE_FILETYPE_VOX VCE_FILETYPE_WAVE VCE_FILETYPE_FLAT VCE_FILETYPE_AUTO Refer to the Details section for more information.
encoding	Encoding used for all messages in the file.

fileinfo	Pointer to a structure that contains parameters specific to the file type. For unformatted files, this argument must be NULL. For NMS VOX files, ***fileinfo*** can point to the VCE_CREATE_VOX structure (if ***fileinfo*** is NULL, default values are used).
	The VCE_CREATE_VOX structure is:
	`typedef struct`
	`{`
	`  DWORD size;  /* size of this structure */`
	`  DWORD maxindex;  /* number of index entries in header (default 250) */`
	`} VCE_CREATE_VOX;`
	For WAVE files, ***fileinfo*** can point to a VCE_WAVE_INFO structure. (Refer to **vceSetWaveInfo** for a description of this structure.) This argument enables WAVE files to be created with formats for which there is no predefined encoding value (for example, 22 kHz stereo). Set ***encoding*** to 0 in this case.

vh	Pointer to a returned voice handle.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	***filetype*** is invalid.
CTAERR_FILE_CREATE_FAILED	File system error.
CTAERR_FILE_EXISTS	You attempted to create a file that already exists.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_PATH_NOT_FOUND	An invalid path is specified when creating a WAVE file.
CTAERR_SVR_COMM	Server communication error.
VCEERR_UNSUPPORTED_ENCODING	Neither the Voice Message service nor the associated play or record service, if any, support the specified encoding.

Details

vceCreateFile creates a new voice file. If the file already exists, an error is returned.

The following file types are supported:

- NMS VOX
- WAVE
- Flat (unformatted)

To infer the file type from the ***filename***, set ***filetype*** to VCE_FILETYPE_AUTO.

If the file type is VCE_FILETYPE_AUTO and the file ends in...	The file type is...
.VOX	VCE_FILETYPE_VOX
.WAV or *.WAVE*	VCE_FILETYPE_WAVE
Any other	VCE_FILETYPE_FLAT

Note: The file extensions can also be lowercase.

The default number of index entries in a VOX file is 250. The minimum number of index entries is 48. If you specify less than 48 entries in the call to **vceCreateFile**, the number of index entries will be 48. The maximum number of index entries is 6500.

To create a WAVE file, **vceCreateFile** maps the encoding to the appropriate WAVE header values based on an internal table. **vceSetWaveInfo** enables you to add entries for encodings not already in the table.

When using the localhost default server, the Natural Access Server (*ctdaemon*) creates the file. To ensure the proper location of the file, specify a fully qualified path for the ***filename*** argument.

Refer to *Encoding descriptions* on page 149 and *Encoding and WAVE information* on page 150 for details about the encoding information that is compiled into the Voice Message service.

See also

vceAssignHandle, **vceOpenFile**, **vceOpenMemory**

Example

```
/* mycreatefile - create a new file for recording to */

void mycreatefile (CTAHD ctahd, char *filename, VCEHD *vh)
{
    unsigned        ret;

    ret = vceCreateFile (ctahd, filename, VCE_FILETYPE_VOX,
                                        VCE_ENCODE_NMS_24, NULL, vh);
    if (ret == CTAERR_FILE_EXISTS)
        printf("Create failed: File %s already exists\n", filename);
    else if (ret != SUCCESS)
    {
        char textbuf[80];
        ctaGetText (ctahd, ret, textbuf, sizeof textbuf);
        printf("Error creating %s: error %s from vceCreateFile\n", filename,
            textbuf);
    }
    else
        printf("File %s created\n", filename);
    return;
}
```

vceCreateMemory

Allocates a memory block and assigns a voice handle to it.

Prototype

DWORD **vceCreateMemory** (CTAHD ***ctahd***, unsigned ***bytes***, unsigned ***encoding***, VCEHD ****vh***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
bytes	Size of the memory block, in bytes.
encoding	Type of voice encoding of all data in the memory block.
vh	Pointer to a returned voice handle.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Server communication error.
VCEERR_UNSUPPORTED_ENCODING	Neither the Voice Message service or the associated play or record service, if any, support the specified encoding.

Details

vceCreateMemory allocates a range of memory and assigns a voice handle to it. The size is specified in bytes.

An open memory object initially has only one non-empty message, which is assigned message zero (0). The size of this message is equal to the size of the memory block. To record to a message other than message zero (0), either erase message zero (0) or resize it by recording or writing a message that is smaller than the entire memory block.

If your memory block contains multiple messages, use **vceDefineMessages** to tell the Voice Message service how the memory is partitioned.

Memory is always opened in PLAY_RECORD mode. There is no protection against opening overlapping memory regions. There is no interlock to prevent multiple threads recording to the same area from different contexts.

Unlike voice files, memory objects cannot grow.

Note: Recording to a memory object ends with CTA_REASON_FINISHED if the message is bounded by the next message or if the end of the memory block is reached.

Use **vceClose** to free memory and release the handle and internal resources.

Use **vceRead** to access data within the memory block.

Launch **vceCreateMemory** when using either the inproc default server or the localhost default server.

See also

vceOpenMemory

Example

```
/* myCreateMemory - Allocate a block of memory and get a voice handle to it. */

void myCreateMemory (CTAHD ctahd, unsigned maxtime, unsigned encoding,
VCEHD *vh)
{
    unsigned framesize;
    unsigned frametime;
    unsigned bytes;
    unsigned        ret;

    vceGetEncodingInfo  (ctahd, encoding, &framesize, &frametime);
    bytes = maxtime/frametime * framesize;
    ret =  vceCreateMemory (ctahd, bytes, encoding, vh);
    if (ret != SUCCESS)
    {
       char textbuf[80];
       ctaGetText (ctahd, ret, textbuf, sizeof(textbuf));
       printf("Error creating memory: error %s from vceCreateMemory\n",
               textbuf);
    }
}
```

vceDefineMessages

Defines messages in a flat (unformatted) file or memory block.

Prototype

DWORD **vceDefineMessages** (VCEHD ***vh***, VCE_SEGMENT ****segments[]***, unsigned ***msgcount***)

Argument	Description
vh	Handle of an open voice object.

segments	Pointer to an array of VCE_SEGMENT structures. Each element defines the corresponding message number. The VCE_SEGMENT structure is:
	`typedef struct`
	`{`
	`  unsigned offset;`
	`  unsigned size;`
	`} VCE_SEGMENT;`
	Refer to the Details section for a description of these fields.

msgcount	Number of elements in the ***segments*** array.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	***segments*** is NULL or ***msgcount*** is zero (0).
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Messages overlap or extend beyond the end of the file or memory block.

Details

vceDefineMessages defines multiple messages in the file or memory block that ***vh*** specified. This function tells the Voice Message service how the file or memory is partitioned.

The VCE_SEGMENT structure contains the following fields:

Field	Description
offset	Starting offset in bytes from the beginning of the file or memory block.
size	Number of bytes in the message.

To skip a message number, set both offset and size to zero (0) in the element of the ***segments*** array that the message number indexed.

Messages must not overlap one another or extend beyond the size of the file or memory block.

See also

vceCreateMemory, **vceOpenFile**, **vceOpenMemory**

Example

```
/*
* Open a file that contains a simple header, then assign message numbers to
 * segments of the file.
 *
 * Assume the file's header structure is:
 *     unsigned count          = number of messages
 *     unsigned encoding       = voice encoding of all messages
 *     VCE_SEGMENT msg[count]  = array of "count" structures
 *                               containing offset and size
 */

extern CTAHD  CtaHd;

void myOpen (char *filename, VCEHD *returnedvh)
{
    FILE        *filep;
    unsigned     msgcount;
    unsigned     encoding;
    VCE_SEGMENT *segments;
    VCEHD        vh;

    /* Read the count, allocate a buffer, read the msg list */
    filep = fopen (filename, "rb");
    fread (&msgcount, sizeof msgcount, 1, filep);
    fread (&encoding, sizeof encoding, 1, filep);
    segments = malloc (msgcount * sizeof (VCE_SEGMENT));
    fread (segments,  sizeof (VCE_SEGMENT), msgcount, filep);
    fclose (filep);

    /* Re-open the file with VCE and define the messages */
    vceOpenFile (CtaHd, filename, VCE_FILETYPE_FLAT, VCE_PLAY_ONLY,
                                              encoding, &vh);
    vceDefineMessages (vh, segments, msgcount);
    *returnedvh = vh;
}
```

vceErase

Removes data at the current position in the current message.

Prototype

DWORD **vceErase** (CTAHD ***ctahd***, unsigned ***msec***, unsigned ****actual***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
msec	Amount of data to erase, in milliseconds.
actual	Pointer to returned number of milliseconds erased.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is active on the context.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_OPERATION	Either the current file type supports erasing only to the end of the message, or the current message is a list.
VCEERR_NO_MESSAGE	No current message is in the context.
VCEERR_OUT_OF_INDICES	No free header entries are in the destination VOX file.
VCEERR_PLAY_ONLY	The voice object containing the current message is not open for record.

Details

At the current position in the current message, **vceErase** removes the specified number of milliseconds of data in the context that ***ctahd*** specifies. It returns the number of milliseconds deleted in ***actual***, unless ***actual*** is NULL.

The current position is unchanged. The current message size is reduced by the amount deleted.

You cannot erase while play or record is active.

If you attempt to erase beyond the end of the current message, SUCCESS is returned and the current position becomes the end of the message.

Erasing data to anywhere except the end of a message is allowed only on file types that fully support editing (currently only VOX files).

EDTX encoding types are not supported.

See also

vceEraseMessage

Example

```
/* Trim specified number of seconds off end of the current message */

void myTrim (CTAHD ctahd, unsigned seconds)
{
    unsigned msec = seconds*1000 ;

    vceSetPosition (ctahd, msec, VCE_SEEK_END, NULL) ;
    vceErase (ctahd, msec, NULL);
}
```

vceEraseMessage

Deletes the specified message from the specified voice object.

Prototype

DWORD **vceEraseMessage** (VCEHD ***vh***, unsigned ***message***)

Argument	Description
vh	Handle of an open voice object.
message	Message number.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is active.
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Invalid message number for the file type.
VCEERR_PLAY_ONLY	Voice object is not open for record.

Details

vceEraseMessage deletes the ***message*** in the voice object that ***vh*** specified by changing its size to zero (0). The specified message becomes the current message in the context associated with ***vh***. The voice object must be open for record.

To erase all messages in ***vh***, specify VCE_ALL_MESSAGES as the message number.

vceEraseMessage returns SUCCESS for any message number that is valid for the file type of ***vh***. Otherwise, the return code is VCEERR_INVALID_MESSAGE.

See also

vceErase

Example

```
/* Replace or create a message from a data buffer */

void myWriteMessage (CTAHD ctahd, VCEHD vh, unsigned message,
                                 char *buffer, unsigned bytes)
{
    vceEraseMessage (vh, message);
    vceWrite(ctahd, buffer, bytes, VCE_INSERT, NULL);
}
```

vceGetContextInfo

Returns the current Voice Message service status of the specified context.

Prototype

DWORD **vceGetContextInfo** (CTAHD ***ctahd***, VCE_CONTEXT_INFO ****status***, unsigned ***size***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.

status	Pointer to a buffer to receive the VCE_CONTEXT_INFO structure:
	`typedef struct`
	`{`
	`  DWORD  size;`
	`  DWORD  numcurrent;`
	`  DWORD  position;`
	`  DWORD  reasondone;`
	`  DWORD  auxerror;`
	`  DWORD  function;`
	`  DWORD  underruns;`
	`  DWORD  currentgain;`
	`  DWORD  currentspeed;`
	`  DWORD  encoding;`
	`  DWORD  framesize;`
	`  DWORD  frametime;`
	`} VCE_CONTEXT_INFO;`
	Refer to the Details section for a description of these fields.

size	Size in bytes of the ***status*** buffer.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_SIZE	***size*** is less than the size of DWORD.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.

Details

vceGetContextInfo returns the current Voice Message service status information of the context that ***ctahd*** specifies, including information about the current or most recent play or record operation on a context.

A context contains a current message and current position, used by functions including **vcePlay**, **vceRecord**, **vceErase**, **vceRead**, and **vceWrite**. The current message can consist of a list of messages that are treated as one contiguous message. The numcurrent field in the VCE_CONTEXT_INFO structure gives the number of messages in the list.

Use **vceGetCurrentList** to view the current message or list.

Note: During play or record, the current position is updated as each buffer is internally submitted for play or completed for record. When play ends, the values can be adjusted downward if not all of the bytes submitted to the device were actually played.

The VCE_CONTEXT_INFO structure contains the following fields:

Field	Description
size	Number of bytes written to the buffer to which ***status*** pointed.
numcurrent	Number of messages that compose the current message list. Use **vceGetCurrentList** to see the messages.
position	Current position, in milliseconds, in the current message.
reasondone	Reason for completion of the most recent play or record.
auxerror	Auxiliary error code (for example, OS error).
function	VCE_PLAY or VCE_RECORD if a function is currently active, otherwise zero (0).
underruns	Count of underruns. An underrun occurs when the system is unable to supply buffers to the hardware fast enough to keep up with speech. The underrun count is cumulative from when the context is opened.
currentgain	Current value of play gain. This value is set to the value of the play gain parameter when play starts. **vceSetPlayGain** can change the value when play is active.
currentspeed	Current value of play speed. This value is set to the value of the play speed parameter when play starts. **vceSetPlaySpeed** can change the value when play is active.
encoding	Encoding of the current message.
framesize	Frame size of the current encoding. See **vceGetEncodingInfo**.
frametime	Frame duration of the current encoding. See **vceGetEncodingInfo**.

See also

vceGetOpenInfo

Example

```
/* Get the underrun count */

unsigned myGetUnderruns (CTAHD ctahd)
{
    VCE_CONTEXT_INFO contextinfo;

    vceGetContextInfo (ctahd, &contextinfo, sizeof contextinfo);
    return contextinfo.underruns;
}
```

vceGetCurrentList

Returns the list of messages contained in the current message.

Prototype

DWORD **vceGetCurrentList** (CTAHD ***ctahd***, VCE_MESSAGE ****msglist[]***, unsigned ***maxcount***, unsigned ****actualcount***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.

msglist	Pointer to an array of the VCE_MESSAGE structure:
	`typedef struct`
	`{`
	`  VCEHD  vh;`
	`  unsigned message;`
	`} VCE_MESSAGE;`
	Refer to the Details section for a description of these fields.

maxcount	Number of elements in the ***msglist*** array.
actualcount	Pointer to returned number of elements written to the ***msglist*** array.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.

Details

vceGetCurrentList returns the list of messages contained in the current message for the context that ***ctahd*** specifies. When **vcePlayList** or **vceSetCurrentList** defines the current message, it can contain a list of messages.

The VCE_MESSAGE structure contains the following fields:

Field	Description
vh	Handle of an open voice object.
message	Message number in the voice object that ***vh*** specifies.

The number of messages in the list is returned in ***actualcount*** unless ***actualcount*** is NULL. If there is no current message, the function returns SUCCESS and zero (0) in ***actualcount***. If ***maxcount*** is less than or equal to the actual list size, the function returns SUCCESS and ***maxcount*** in ***actualcount***.

To retrieve the list size of the current message, use **vceGetContextInfo**.

Applications accessing the same voice object using **ctaAttachHandle** in a shared context environment are returned separate handles. The Voice Message service internally records the handle (***vh***) of the application that opened or created the voice object.

See also

vceGetContextInfo

Example

```
/*Return the current message (or first message if current message is a list*/

void myGetCurrentMessage (CTAHD ctahd, VCEHD *vh, unsigned *message)
{
    VCE_MESSAGE vcemsg;
    unsigned    actual;

    vceGetCurrentList (ctahd, &vcemsg, 1, &actual);
    if (actual == 0)
    {
        *message = VCE_UNDEFINED_MESSAGE;
        *vh      = 0;
    }
    else
    {
        *vh      = vcemsg.vh;
        *message = vcemsg.message;
    }
}
```

vceGetCurrentSize

Returns the size of the current message.

Prototype

DWORD **vceGetCurrentSize** (CTAHD ***ctahd***, unsigned ****actualsize***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
actualsize	Pointer to the returned size, in milliseconds, of the current message.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.
VCEERR_NO_MESSAGE	No current message is in the context.

Details

vceGetCurrentSize returns the size, in milliseconds, of the current message for the context that ***ctahd*** specifies. If the current message is a list, this function returns the total size of all messages in the list.

Use **vceGetContextInfo** to retrieve the current position in the current message.

Example

```
/* delete current message if smaller than threshold */
extern CTAHD Ctahd;

void myDeleteSmallMessage (unsigned minseconds)
{
    unsigned actualsize;      /* size in milliseconds */

    vceGetCurrentSize (Ctahd, &actualsize) ;
    if (actualsize < minseconds * 1000)
    {
        /* Use vceErase to erase the current message */
        vceSetPosition (Ctahd, 0, VCE_SEEK_SET, NULL) ;
        vceErase (Ctahd, actualsize, NULL);
    }
}
```

vceGetEncodingInfo

Returns information about a voice encoding.

Prototype

DWORD **vceGetEncodingInfo** (CTAHD ***ctahd***, unsigned ***encoding***, unsigned ****framesize***, unsigned ****frametime***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
encoding	Encoding ID (for example, VCE_ENCODE_NMS_24).
framesize	Pointer to the returned size (in bytes) of one frame. Can be NULL.
frametime	Pointer to the returned time (in milliseconds) represented by one frame. Can be NULL.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	Invalid encoding.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.

Details

vceGetEncodingInfo returns information about all encoding that the play or record service (for example, the ADI service) associated with the context supports.

framesize is the minimum data block size. Data at the specified encoding is always a multiple of this size. ***frametime*** is the duration to play or record one frame. The data rate in bytes per second is ***framesize*** * 1000 / ***frametime***.

Use the following formulas to convert between milliseconds and bytes. These translations truncate the results to frame multiples.

bytes = (milliseconds / ***frametime***) * ***framesize***

milliseconds = (bytes / ***framesize***) * ***frametime***

If **vceGetEncodingInfo** is called for a file containing an EDTX encoding type, the maximum frame size for the encoding type is returned in ***framesize*** and the uncompressed frame duration is returned in ***frametime***.

Use **vceGetOpenInfo** to return the encoding used in an open voice object.

Example

```
/* Return message size in bytes */

unsigned myMsgSize (VCEHD vh, unsigned message)
{
    VCE_OPEN_INFO openinfo;
    unsigned      framesize;
    unsigned      frametime;
    unsigned      msgsize;

    vceGetOpenInfo (vh, &openinfo, sizeof openinfo) ;
    vceGetEncodingInfo (openinfo.ctahd, openinfo.encoding, &framesize,
                        &frametime) ;
    vceGetMessageSize (vh, message, &msgsize) ;
    return  (msgsize / frametime * framesize);
}
```

vceGetHighMessageNumber

Returns the highest message number in an open voice object.

Prototype

DWORD **vceGetHighMessageNumber** (VCEHD ***vh***, unsigned ****highmsg***)

Argument	Description
vh	Handle of an open voice object.
highmsg	Pointer to the returned message number.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_SVR_COMM	Server communication error.

Details

vceGetHighMessageNumber returns the message number of the highest numbered non-empty message in the voice object that ***vh*** specifies. The returned message number is VCE_UNDEFINED_MESSAGE if there are no messages in the voice object.

The highest message returned is not necessarily an indication of how many messages exist in a voice object. To count the number of actual messages, call **vceGetMessageSize** for each message number from zero (0) to ***highmsg*** inclusive, and add the messages that have non-zero sizes.

Use this function to determine the next message number to use (one more than ***highmsg***) for a voice mail system where the messages must be maintained in chronological order even if deleted messages left gaps in the message numbers.

vceGetHighMessageNumber does not affect the current message.

See also

vceGetUniqueMessageNumber

Example

```
/* Count messages in a voice file */

unsigned myMessageCount (VCEHD vh)
{
    unsigned highmsg;
    unsigned message;
    unsigned msgsize;
    unsigned count = 0;

    vceGetHighMessageNumber (vh, &highmsg);
    if (highmsg == VCE_UNDEFINED_MESSAGE)
        return 0 ;

    for (message=0; message <= highmsg; message++)
    {
        vceGetMessageSize (vh, message, &msgsize);
        if (msgsize != 0)
            ++ count;
    }
    return count;
}
```

vceGetMessageSize

Returns the size (in milliseconds) of the specified message.

Prototype

DWORD **vceGetMessageSize** (VCEHD ***vh***, unsigned ***message***, unsigned *****msgsize***)

Argument	Description
vh	Handle of an open voice object.
message	Message number.
msgsize	Pointer to a returned message size (in milliseconds).

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Invalid message number for file type.

Details

vceGetMessageSize returns the size (in milliseconds) of the voice message that the voice handle ***vh*** and message number ***message*** specify.

vceGetMessageSize returns SUCCESS for any message number that is valid for the file type of ***vh***. Otherwise, the return code is VCEERR_INVALID_MESSAGE.

To convert the size into bytes, obtain the encoding format with **vceGetOpenInfo** and the frame size and frame duration with **vceGetEncodingInfo**. Convert using the following formula:

bytes = (milliseconds / ***frametime***) * ***framesize***

To obtain the total size of all messages in a voice object, use VCE_ALL_MESSAGES as the message number.

To obtain the size of the current message, use **vceGetCurrentSize**.

vceGetMessageSize does not affect the current message.

Example

Refer to **vceGetHighMessageNumber**.

vceGetOpenInfo

Returns status information about an open voice object.

Prototype

DWORD **vceGetOpenInfo** (VCEHD ***vh***, VCE_OPEN_INFO ****openinfo***, unsigned ***size***)

Argument	Description
vh	Handle of an open voice object.

openinfo	Pointer to a buffer to receive the following structure:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD filetype;`
	`  DWORD openmode;`
	`  DWORD encoding;`
	`  CTAHD ctahd;`
	`} VCE_OPEN_INFO;`
	Refer to the Details section for a description of these fields.

size	Size, in bytes, of the ***openinfo*** buffer.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_SIZE	***size*** is less than the size of DWORD.
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_SVR_COMM	Server communication error.

Details

vceGetOpenInfo returns status information about an open voice file or memory object.

The VCE_OPEN_INFO structure contains the following fields:

Field	Description
size	Number of bytes written to the buffer to which ***openinfo*** points.
filetype	Type of voice file (VCE_FILETYPE_xxx), or zero (0) if memory.
openmode	VCE_PLAY_ONLY or VCE_PLAY_RECORD.
encoding	Voice encoding of all messages in the file or memory block.
ctahd	Context handle used when ***vh*** is obtained.

The VCE_OPEN_INFO data remains constant for a given handle as long as it is open.

Use **vceGetContextInfo** to get the status of the context that ***ctahd*** identifies.

Use **vceGetEncodingInfo** for information about the encoding.

Example

```
/* Get the encoding of an open voice file*/

unsigned myGetEncoding (VCEHD vh)
{
    VCE_OPEN_INFO openinfo;

    if (vceGetOpenInfo (vh, &openinfo, sizeof openinfo) != SUCCESS)
        return 0;
    return openinfo.encoding;
}
```

vceGetUniqueMessageNumber

Returns the lowest unused message number in an open voice object.

Prototype

DWORD **vceGetUniqueMessageNumber** (VCEHD ***vh***, unsigned ****message***)

Argument	Description
vh	Handle of an open voice object.
message	Pointer to a returned message number or VCE_UNDEFINED_MESSAGE.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_SVR_COMM	Server communication error.

Details

Use **vceGetUniqueMessageNumber** to obtain a message number before recording a new message. This function returns the lowest empty (0-length) message in the open voice object. If the order of the messages must be maintained, use **vceGetHighMessageNumber**.

The returned message number is VCE_UNDEFINED_MESSAGE if there are no unused message numbers in the voice object.

vceGetUniqueMessageNumber does not affect the current message.

See also

vceGetMessageSize

Example

```
/* Start recording a new message */

void myAddMessage (VCEHD vh, unsigned *message)
{
    vceGetUniqueMessageNumber (vh, message) ;
    vceRecordMessage (vh, *message, VCE_NO_TIME_LIMIT, NULL) ;
}
```

vceGetWaveInfo

Returns wave header information for a given encoding value.

Prototype

DWORD **vceGetWaveInfo** (CTAHD ***ctahd***, unsigned ***encoding***, VCE_WAVE_INFO ****waveinfo***, unsigned ***size***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
encoding	Encoding ID (for example, VCE_ENCODE_NMS_24).

waveinfo	Pointer to the buffer to receive the following VCE_WAVE_INFO structure:
	`typedef struct`
	`{`
	`  DWORD  size;`
	`  WORD   format;`
	`  WORD   nchannels;`
	`  DWORD  samplespersec;`
	`  DWORD  datarate;`
	`  WORD   blocksize;`
	`  DWORD  bitspersample;`
	`} VCE_ WAVE_INFO;`
	Refer to the Details section for a description of these fields.

size	Size in bytes of the ***waveinfo*** buffer.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_SIZE	***size*** is less than the size of DWORD.
CTAERR_NOT_FOUND	No wave information found for specified encoding.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.

Details

vceGetWaveInfo returns information from the Voice Message service WAVE information table for a specified encoding. The Voice Message service maintains a table that maps encoding values to WAVE file header information. **vceCreateFile** uses this information to create a new WAVE file, and **vceOpenFile** uses it to determine the encoding of the file. For more information about WAVE files, refer to the *Microsoft Windows Multimedia Programmer's Reference*.

The table initially contains compiled-in entries for known encodings. Use **vceSetWaveInfo** to add new entries.

The VCE_WAVE_INFO structure contains the following fields:

Field	Description
size	Size of the structure, in bytes. This is set to: `sizeof (VCE_WAVE_INFO)`
format	The WAVE format type as defined by Microsoft. For example, WAVE_FORMAT_PCM (defined in the Microsoft header file *mmreg.h*).
nchannels	Number of discrete channels in the format. Use 1 for mono and 2 for stereo.
samplespersec	Sample rate in samples per second.
datarate	Average bytes per second.
blocksize	Minimum block size of the data. For PCM data, the block size is the number of bytes in a single sample.
bitspersample	Number of bits per sample.

Example

```
/* Display contents of WAVE info table */
/* Sample output:
    *
    * Encoding  Wave Type  M/S  SampleRate  DataRate  Block Bits
    * ------------------------------------------------------------
    *       1         56   M       8000         2100     42   2
    */

void myShowWaveInfo (CTAHD ctahd)
{
  unsigned      encoding;
  VCE_WAVE_INFO waveinfo;

  printf("Encoding  Wave Type  M/S" "  SampleRate  DataRate  Block Bits\n");
  printf("-----------------------" "-----------------------------------\n");
  for (encoding = 0; encoding < 256; encoding ++)
  {
    if (vceGetWaveInfo(ctahd,encoding,&waveinfo,sizeof waveinfo)!= SUCCESS)
          continue;
       printf("    %3d        %3d   %c" "     %6d       %6d     %3d  %2d\n",
              encoding,
              waveinfo.format,
              waveinfo.nchannels == 1 ? 'M' : 'S',
              waveinfo.samplespersec,
              waveinfo.datarate,
              waveinfo.blocksize,
              waveinfo.bitspersample); }
}
```

Sample run

```
Encoding  Wave Type  M/S  SampleRate  DataRate  Block Bits
------------------------------------------------------------
    1         56      M      8000        2100    42    2
    2         56      M      8000        3100    62    3
    3         56      M      8000        4100    82    4
    4         56      M      8000        8200   162    8
   10          7      M      8000        8000     1    8
   11          6      M      8000        8000     1    8
   13          1      M      8000       16000     2   16
   14         23      M      6000        3000     1    4
   15         23      M      8000        4000     1    4
   16          1      M     11025       11025     1    8
   17          1      M     11025       22050     2   16
   22         17      M      6000        6600   244    4
   23         17      M      8000        8800   244    4
```

vceLoadPromptRules

Loads the specified prompt rules table.

Prototype

DWORD **vceLoadPromptRules** (CTAHD ***ctahd***, char ****name***, VCEPROMPTHD ****prompthandle***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
name	Pointer to the file name of the prompt rule table. If the file does not include an extension, the default extension of *.tbl* is added to the file name.
prompthandle	Pointer to the integer variable to receive the handle of the specified prompt table.

Return values

Return value	Description
SUCCESS	
CTAERR_FILE_ACCESS_DENIED	Another application process or thread in the system may have the file open for recording.
CTAERR_FILE_NOT_FOUND	Specified file does not exist.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.

Details

vceLoadPromptRules reads the specified prompt rules table into memory. The returned prompt handle is passed to **vceBuildPromptList**.

Open the associated voice file using **vceOpenFile**. The voice file is used with the output from **vceBuildPromptList** to call **vcePlayList**.

Each rule table is usually specific to one language such as French, German, or English. More than one rule table can be loaded at a time.

See also

vceUnloadPromptRules

Example

Refer to **vceBuildPromptList**.

vceOpenFile

Opens a voice file and returns a voice handle.

Prototype

DWORD **vceOpenFile** (CTAHD ***ctahd***, char ****filename***, unsigned ***filetype***, unsigned ***openmode***, unsigned ***encoding***, VCEHD ****vh***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
filename	Pointer to a file name. If the file name does not include a full pathname, first the current directory is searched for the file, then the path in the CTA_DPATH environment variable. The current directory is either that of the application or of the Natural Access Server (*ctdaemon*), depending on whether the application is using the inproc default server or the localhost default server.
filetype	Acceptable values are: VCE_FILETYPE_VOX VCE_FILETYPE_WAVE VCE_FILETYPE_FLAT VCE_FILETYPE_AUTO Refer to the Details section for more information.
openmode	Either VCE_PLAY_ONLY or VCE_PLAY_RECORD.
encoding	For unformatted files, the encoding of the data in the file. Set this to zero (0) for other files.
vh	Pointer to a returned voice handle.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	An undefined value was passed for ***openmode*** or ***filetype***, or an encoding of zero (0) was specified for a flat file.
CTAERR_FILE_ACCESS_DENIED	Another application process or thread in the system has the file open for recording.
CTAERR_FILE_NOT_FOUND	Specified file does not exist.
CTAERR_FILE_OPEN_FAILED	File open failed due to a system error.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.
VCEERR_UNSUPPORTED_ENCODING	Neither the Voice Message service nor the associated play or record service, if any, supports the specified encoding.
VCEERR_WRONG_ENCODING	A non-zero encoding that was specified is not the same as the encoding in the file.
VCEERR_WRONG_FILE_TYPE	The file is not of the specified or implied type.

Details

vceOpenFile opens an existing voice file. The file can be opened for play only or for both play and record. A file opened for record can have one writer but many readers. Only one application process or thread in the system can open the file for recording, but simultaneous play access is always allowed.

The following file types are supported:

- NMS VOX
- WAVE
- Flat (unformatted)

To have the file type inferred from the ***filename***, set ***filetype*** to VCE_FILETYPE_AUTO.

If the filetype is VCE_FILETYPE_AUTO and the file ends in...	Then the file type will be...
.VOX	VCE_FILETYPE_VOX
.WAV or *.WAVE*	VCE_FILETYPE_WAVE
Any other extension	VCE_FILETYPE_FLAT

Note: The file extensions can also be lowercase.

If the file name has no extension and the file type is specified, then a *.vox*, *.wav*, or *.vce* extension is automatically implied. In UNIX, if no file with the implied name is found, **vceOpenFile** also looks for a file with the name as specified.

The NMS VOX file format stores multiple messages in one file and enables an application to delete, add, and edit messages.

WAVE files must conform to the Microsoft Multimedia Format and must contain a WAVE chunk as the first or only RIFF form in the file. For additional information, refer to the *Microsoft Windows Multimedia Programmer's Reference*.

Flat files contain no header information, only data.

The ***encoding*** argument is necessary for flat files. It is not necessary to specify the encoding for other file types because the encoding is contained in the file header. If a non-zero encoding is specified, it must match the encoding in the header, or an error is returned.

See also

vceAssignHandle, **vceCreateFile**

Example

```
/* Open ten files containing spoken digits and store the handles in an array. */
VCEHD prompt_hd[10] = {0};

void openprompts (CTAHD ctahd)
{
    unsigned index;

    for (index=0; index<10; index++)
    {
        char filename[20];

        sprintf (filename, "PROMPT%d.VCE", index);
        vceOpenFile (ctahd, filename,
                    VCE_FILETYPE_FLAT, VCE_PLAY_ONLY,
                    VCE_ENCODE_NMS_24, &prompt_hd [index] );
    }
}
```

vceOpenMemory

Assigns a voice handle to a memory block.

Prototype

DWORD **vceOpenMemory** (CTAHD ***ctahd***, BYTE ****address***, unsigned ***bytes***, unsigned ***encoding***, VCEHD ****vh***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext**.
address	Base address of the memory block in process space or shared memory.
bytes	Size of the memory block, in bytes.
encoding	Type of voice encoding of all data in the memory block.
vh	Pointer to a returned voice handle.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_NOT_IMPLEMENTED	This error is returned when an application that is not using the inproc default server launches **vceOpenMemory**.
VCEERR_UNSUPPORTED_ENCODING	Neither the Voice Message service nor the associated play or record service, if any, supports the specified encoding.

Details

vceOpenMemory assigns a voice handle to a range of the application memory. The size is specified in bytes.

vceOpenMemory is not supported when using the localhost default server. Use **vceCreateMemory** to allocate a memory block and assign a voice handle to it. **vceCreateMemory** can be launched when using either the inproc default server or the localhost default server.

An open memory object has initially only one non-empty message that is assigned message zero (0). The size of this message is equal to the size of the memory block. To record other than to message zero (0), either erase message zero (0), or resize it by recording or writing a message that is smaller than the entire memory block.

If your memory block contains multiple messages, use **vceDefineMessages** to tell the Voice Message service how the memory is partitioned.

Memory is always opened in PLAY_RECORD mode. There is no protection against opening overlapping memory regions. There is no interlock to prevent multiple threads recording to the same area from different contexts.

Unlike voice files, memory objects cannot grow.

Note: Recording to a memory object ends with CTA_REASON_FINISHED if the message is bounded by the next message or if the end of the memory block is reached.

vceClose releases the handle and internal resources.

Natural Access does not allocate the memory block in **vceOpenMemory** or free it in **vceClose**.

See also

vceAssignHandle, **vceCreateFile**, **vceErase**, **vceEraseMessage**, **vceOpenFile**

Example

```
/* myCreateMemory - Allocate a block of memory and get a voice handle to it. */

void myCreateMemory (CTAHD ctahd, unsigned maxtime, unsigned encoding, VCEHD *vh)
{
    unsigned framesize;
    unsigned frametime;
    unsigned bytes;
    void     *address;

    vceGetEncodingInfo  (ctahd, encoding, &framesize, &frametime);
    bytes = maxtime/frametime * framesize;
    address = malloc (bytes);
    vceOpenMemory (ctahd, address, bytes, encoding, vh);
}
```

vcePlay

Starts playing from the current position in the current message.

Prototype

DWORD **vcePlay** (CTAHD ***ctahd***, unsigned ***maxtime***, VCE_PLAY_PARMS ****parms***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
maxtime	Maximum amount to play, in milliseconds. Use VCE_NO_TIME_LIMIT to play with no time limit.

parms	Pointer to a parameter structure. Set this to NULL to use default values. The VCE_PLAY_PARMS structure is:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD DTMFabort;`
	`  INT32 gain;`
	`  DWORD speed;`
	`  DWORD maxspeed;`
	`} VCE_PLAY_PARMS;`
	Refer to *VCE_PLAY_PARMS* on page 134 for a description of these fields.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is already active on the context.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_OUTPUT_ACTIVE	Another function is controlling the MVIP output timeslot associated with the context.
CTAERR_SVR_COMM	Server communication error.
VCEERR_NO_MESSAGE	No current message in the context.

Events

Event	Description
VCEEVN_PLAY_DONE	Play completed. The value field of the event contains one of the following reasons or an error code: CTA_REASON_DIGIT Touch-tone digit was received and the corresponding bit in the DTMF abort parameter was set. CTA_REASON_FINISHED End of the current message or list was reached. CTA_REASON_RELEASED Stopped because the call was disconnected. CTA_REASON_STOPPED Stopped by **vceStop**, or the current message was invalidated because one of the referenced voice objects was closed. CTA_REASON_TIMEOUT Time limit ***maxtime*** was reached. CTAERR_FUNCTION_NOT_AVAIL Played object has encoding that is not supported.

Details

vcePlay initiates playing from the current position in the current message for the specified context. The current message is defined by **vceConvertMessage**, **vceCopyMessage**, **vceEraseMessage**, **vcePlayList**, **vcePlayMessage**, **vceRecordMessage**, **vceSetCurrentList**, or **vceSetCurrentMessage**.

vcePlay always returns immediately. If the return is SUCCESS, a VCEEVN_PLAY_DONE event occurs when play completes.

The current position is advanced by the number of milliseconds actually played. The number of milliseconds played is reported in the size field of the VCEEVN_PLAY_DONE event.

If the current position is already at the end of the current message, **vcePlay** returns SUCCESS and the DONE event occurs immediately.

Use this function to resume play after stopping a message in the middle. Use **vceSetPosition** to adjust the current position before resuming (for example, if you want to rewind or fast forward).

EDTX encoding types are not supported.

Refer to *Playing* on page 17 for more information.

See also

vceStop

Example

```
/* Sample to reposition play. Play a message, allowing pause, forward, and */
/* rewind using digits 8, 9, and 7, respectively. Any other digit stops    */
/* play. The function returns the reason for stopping.                     */

extern CTAHD      Ctahd;
extern CTAQUEUEHD CtaQueueHd;

int myFancyPlay(VCEHD vh, unsigned message)
{
    BOOL      done   = FALSE;
    BOOL      paused = FALSE;
    CTA_EVENT event;
    char      digit;

    vcePlayMessage (vh, message, NULL);
    while (!done)
    {
        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
        if (event.id == VCEEVN_PLAY_DONE)
        {
            if (event.value == CTA_REASON_DIGIT)
            {
                adiPeekDigit(Ctahd, &digit);
                switch (digit)
                {
                    case '7':                     /* Rewind 2 seconds */
                        vceSetPosition (Ctahd, -2000, VCE_SEEK_CUR, NULL);
                        break;
                    case '8':                     /* Toggle pause     */
                        paused = !paused;
                        break;
                    case '9':                    /* Forward 2 seconds */
                        vceSetPosition (Ctahd, 2000, VCE_SEEK_CUR, NULL);
                        break;
                    default:
                        done = TRUE;
                        break;

                 }
                 if (!done)
                 {

                    /* Remove and discard the digit */
                    adiGetDigit (Ctahd, &digit);

                    if (!paused)
                        vcePlay(Ctahd, VCE_NO_TIME_LIMIT, NULL);
                }
            }
            else
                done = TRUE;
        }
        /* Ignore other events */
    }
    return event.value;
}
```

vcePlayList

Starts playing a list of messages.

Prototype

DWORD **vcePlayList** (VCEHD ***vh***, unsigned ***messagelist***[], unsigned ***count***, VCE_PLAY_PARMS ****parms***)

Argument	Description
vh	Handle of an open voice object.
messagelist	List of message numbers in ***vh*** to play as one contiguous message.
count	Number of messages in ***messagelist***.

parms	Pointer to a parameter structure. Set this to NULL to use default values. The VCE_PLAY_PARMS structure is:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD DTMFabort;`
	`  INT32 gain;`
	`  DWORD speed;`
	`  DWORD maxspeed;`
	`} VCE_PLAY_PARMS;`
	Refer to *VCE_PLAY_PARMS* on page 134 for a description of these fields.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is already active on the context.
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_OUTPUT_ACTIVE	Another function is controlling the MVIP output timeslot associated with the context.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	One or more message numbers is out of range for the type of voice file of ***vh***.

Events

Event	Description
VCEEVN_PLAY_DONE	Play completed. The value field of the event contains one of the following reasons or an error code: CTA_REASON_DIGIT Touch-tone digit was received and the corresponding bit in the DTMF abort parameter was set. CTA_REASON_FINISHED End of the current message or list was reached. CTA_REASON_RELEASED Stopped because the call was disconnected. CTA_REASON_STOPPED Stopped by **vceStop**, or the current message was invalidated because one of the referenced voice objects was closed. CTAERR_FUNCTION_NOT_AVAIL Played object has encoding that is not supported.

Details

vcePlayList starts playing from the beginning of the first message in ***messagelist***. Messages in the list are played as one contiguous message with no delays between messages.

This function enables a message to be built from a library of words or phrases contained in a single voice object. To concatenate messages from multiple objects, use **vceSetCurrentList**, followed by **vcePlay**.

vcePlayList always returns immediately. If it returns SUCCESS, a VCEEVN_PLAY_DONE event occurs when play completes. If there are no messages or if all messages have a length of zero (0), the VCEEVN_PLAY_DONE event occurs immediately.

To view the number of milliseconds actually played after play ends, call **vceGetContextInfo** and look at the position field of the returned structure.

The list of messages becomes the current message for the context specified by ***ctahd***. After play stops, resume playing from the current position with **vcePlay**. Use **vceSetPosition** to adjust the current position to anywhere in the list before resuming.

Refer to *Playing* on page 17 for more information.

See also

vceBuildPromptList, **vcePlayMessage**, **vceStop**

Example

```
/* Play digit string from prompt file */

/* This routine starts speaking a digit string.
 * The spoken digits "oh" through "nine" are in 10 messages in
 * DIGITS.VOX where msg 0 = "oh", msg 1 = "one", etc.
*/

extern CTAHD      CtaHd;
extern CTAQUEUEHD CtaQueueHd;

void mySpeakDigits (char *digits)
{
    unsigned  msglist[50];
    unsigned  count;
    VCEHD     vh;
    CTA_EVENT event;

    vceOpenFile (Ctahd, "DIGITS.VOX", 0, VCE_PLAY_ONLY, 0, &vh) ;

    for (count = 0;
        *digits != '\0' && count < sizeof msglist/sizeof msglist[0];
        digits++)
    {
        if (isdigit(*digits))
            msglist[count++]  = *digits - '0' ;
    }

    vcePlayList(vh, msglist, count, NULL);
do
{
        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
} while (event.id != VCEEVN_PLAY_DONE); /* Ignore other events */
vceClose(vh);
}
```

vcePlayMessage

Starts playing one message from the specified voice object.

Prototype

DWORD **vcePlayMessage** (VCEHD ***vh***, unsigned ***message***, VCE_PLAY_PARMS ****parms***)

Argument	Description
vh	Handle of an open voice object.
message	Message number to play. Use zero (0) to play the contents of a flat file, WAVE file, or memory block.

parms	Pointer to a parameter structure. Set this to NULL to use default values. The VCE_PLAY_PARMS structure is:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD DTMFabort;`
	`  INT32 gain;`
	`  DWORD speed;`
	`  DWORD maxspeed;`
	`} VCE_PLAY_PARMS;`
	Refer to *VCE_PLAY_PARMS* on page 134 for a description of these fields.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is already active on the context.
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_OUTPUT_ACTIVE	Another function is controlling the MVIP output timeslot associated with the context.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Specified message number is invalid.

Events

Event	Description
VCEEVN_PLAY_DONE	Play completed. The value field of the event contains one of the following reasons or an error code: CTA_REASON_DIGIT Touch-tone digit was received and the corresponding bit in the DTMF abort parameter was set. CTA_REASON_FINISHED End of the current message or list was reached. CTA_REASON_RELEASED Stopped because the call was disconnected. CTA_REASON_STOPPED Stopped by **vceStop**, or the current message was invalidated because one of the referenced voice objects was closed. CTAERR_FUNCTION_NOT_AVAIL Played object has encoding that is not supported.

Details

vcePlayMessage starts playing from the beginning of the specified ***message*** number in the voice object specified by ***vh***.

vcePlayMessage always returns immediately. If the return is SUCCESS, a VCEEVN_PLAY_DONE event occurs when play completes.

All unused (or deleted) message numbers in the valid range refer to zero-length messages and do not result in an error when referenced.

The range of valid message numbers is determined by the file type:

File type	Range of valid message numbers
VOX	0 to 32,767
Flat	0 to 65,535
WAVE	0 only
Memory	0 to 65,535

Note: Use the special message number VCE_ALL_MESSAGES to play all messages in an object as one message.

To view the number of milliseconds actually played after play ends, call **vceGetContextInfo** and look at the position field of the returned structure.

The voice object and message number become the current message for the context in which the voice handle was opened. After play stops, resume playing from the current position with **vcePlay**. You can adjust the current position with **vceSetPosition** before resuming.

Refer to *Playing* on page 17 for more information.

See also

vceDefineMessages, **vcePlayList**, **vceStop**

Example

```
/* play a message and wait for completion */

extern CTAHD      CtaHd;
extern CTAQUEUEHD CtaQueueHd;

void myPlaymsg(VCEHD vh, unsigned message)
{
    CTA_EVENT event;

    vcePlayMessage (vh, message, NULL) ;
    do
    {
        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
    } while (event.id != VCEEVN_PLAY_DONE); /* Ignore other events */
}
```

vceRead

Reads data at the current position in the current message.

Prototype

DWORD **vceRead** (CTAHD ***ctahd***, BYTE ****buffer***, unsigned ***bytes***, unsigned ****bytesread***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
buffer	Pointer to a buffer to receive data.
bytes	Amount of data to read.
bytesread	Pointer to the returned number of bytes read.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is active on the context.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.
VCEERR_NO_MESSAGE	There is no current message in the context.

Details

vceRead copies the specified number of bytes of data at the current location in the current message in the context specified by ***ctahd*** to a specified ***buffer***. It returns the number of bytes read in ***bytesread***, unless ***bytesread*** is NULL.

The number of bytes read is always an integral multiple of the framesize of the current message. If you specify a buffer size that is not a multiple of the framesize, the number of bytes read is smaller than the requested size.

The following restrictions apply to using **vceRead**:

- EDTX encoding types are not supported.
- You cannot read while play or record is active.
- If you attempt to read beyond the end of the current message, SUCCESS is returned and the current position is set to the end of the current message.
- The current position is advanced by the amount read in millisecond units.

To convert between bytes and milliseconds, use **vceGetContextInfo** to get the frame size and frame time, then use the following formulas:

milliseconds = (bytes / ***framesize***) * ***frametime***

bytes = (milliseconds / ***frametime***) * ***framesize***

See also

vceWrite

Example

```
/* Read a message into memory */
void myLoadMessage (CTAHD ctahd, VCEHD vh, unsigned message,
                    BYTE **returned_address, unsigned *returned_msec,
                    unsigned *returned_bytes)
{
    VCE_CONTEXT_INFO contextinfo;
    unsigned         msec;
    unsigned         bytes;
    unsigned         bytesread;
    BYTE            *buffer;

    *returned_address = NULL;
    *returned_msec    = 0;

    vceSetCurrentMessage (vh, message) ;
    vceGetCurrentSize (ctahd, &msec) ;
    if (msec == 0)
        return ;

    /* Convert the size from milliseconds to bytes */
    vceGetContextInfo (ctahd, &contextinfo, sizeof contextinfo);
    bytes = msec / contextinfo.frametime * contextinfo.framesize;
    buffer = malloc(bytes) ;

    vceRead (ctahd, buffer, bytes, &bytesread);
    *returned_address = buffer;
    *returned_msec    = msec;
    *returned_bytes   = bytes;
    return;
}
```

vceReadMessageText

Reads the message text for a specified voice message in a VOX file.

Prototype

DWORD **vceReadMessageText** (VCEHD ***vh***, unsigned ***message***, void ****buffer***, unsigned ***bytes***, unsigned ****bytesread***)

Argument	Description
vh	Handle of an open VOX file.
message	Any valid VOX file message number.
buffer	Pointer to a buffer to receive data.
bytes	Size of buffer.
bytesread	Pointer to the returned number of bytes read.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is active on the context.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Message number is not in the valid range for VOX files (0 through 32767).
VCEERR_WRONG_FILE_TYPE	Not a VOX file.

Details

vceReadMessageText reads auxiliary data for a specified message in a VOX file. Although the function name suggests that this function is used for textual data, the data can be any binary data.

See also

vceCopyMessageText, **vceWriteMessageText**

Example

```
/* Displays descriptive text for a message in an open VOX file. */
void myReadMessageText (VCEHD vh, unsigned message)
{
char buf[80];
unsigned bytesread;

if (vceReadMessageText (vh, message, buf, sizeof buf-1, &bytesread)
    == SUCCESS
{
    /* Ensure the buffer is null-terminated. */
    buf[bytesread] = '\0';
    printf ("[%] %s\n", message, buf);
}
}
```

vceRecord

Starts recording at the current position in the current message.

Prototype

DWORD **vceRecord** (CTAHD ***ctahd***, unsigned ***maxtime***, unsigned ***insertmode***, VCE_RECORD_PARMS ****parms***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
maxtime	Maximum amount to record, in milliseconds. Specify VCE_NO_TIME_LIMIT to record with no time limit.
insertmode	VCE_OVERWRITE or VCE_INSERT.

parms	Pointer to a parameter structure. Set this to NULL to use default values. The VCE_RECORD_PARMS structure is:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD DTMFabort;`
	`  INT32 gain;`
	`  DWORD novoicetime;`
	`  DWORD silencetime;`
	`  INT32 silenceampl;`
	`  DWORD beepfreq;`
	`  INT32 beepampl;`
	`  DWORD beeptime;`
	`  DWORD AGCenable;`
	`} VCE_RECORD_PARMS;`
	Refer to *VCE_RECORD_PARMS* on page 135 for a description of these fields.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is already active on the context.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_OUTPUT_ACTIVE	Another function is controlling the MVIP output timeslot associated with the context.
CTAERR_RESOURCE_CONFLICT	Energy detector is in use in the ADI service.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_OPERATION	Insertion anywhere except at the end of a message is not supported by the file type of the current message. The operation is also invalid if the current message is a list.
VCEERR_NO_MESSAGE	No current message in the context.
VCEERR_OUT_OF_INDICES	No free header entries in the destination VOX file.
VCEERR_PLAY_ONLY	Voice file was not opened for record.

Events

Event	Description
VCEEVN_RECORD_DONE	Recording completed. The value field of the event contains one of the following reasons or an error code: CTA_REASON_DIGIT Touch-tone digit was received and the corresponding bit in the DTMF abort parameter was set. CTA_REASON_FINISHED No more space is available or recording in overwrite mode reached the end of the existing message. CTA_REASON_NO_VOICE No voice detected for novoicetime milliseconds at the beginning of recording. CTA_REASON_RELEASED Stopped because the call was disconnected. CTA_REASON_STOPPED Stopped by **vceStop** or the current message was invalidated because one of the referenced voice objects was closed. CTA_REASON_TIMEOUT Time limit ***maxtime*** was reached. CTA_REASON_VOICE_END Silence detected for silencetime milliseconds after some voice detected. CTAERR_FUNCTION_NOT_AVAIL Recorded object has encoding that is not supported. CTAERR_RESOURCE_CONFLICT Energy detector is in use in the ADI service.

Details

vceRecord initiates recording from the current position in the current message for the specified context. **vceRecord** always returns immediately. If the return is SUCCESS, a VCEEVN_RECORD_DONE event occurs when recording completes.

Note: If the message has zero length, the DONE event occurs immediately and is not a beep.

Set ***insertmode*** to VCE_OVERWRITE to replace existing data without changing the message size. Set ***insertmode*** to VCE_INSERT to insert new data before the current position. If the current position is the end of the message, VCE_INSERT appends the new data to the current message. Inserting data anywhere except the end of a message is allowed only on file types that support editing (currently only VOX files).

On completion, the current position is after the inserted data for either insert mode.

EDTX encoding types are not supported.

When the ADI energy detector is active, initiate a record operation (which uses the ADI service) by setting novoicetime and silencetime to zero. For more information about the record parameters, refer to *VCE_RECORD_PARMS* on page 135.

The current position is advanced by the number of milliseconds recorded. The number of milliseconds recorded is reported in the size field of the VCEEVN_RECORD_DONE event.

Refer to *Recording* on page 19 for more information.

See also

vcePlayMessage, **vceRecordMessage**, **vceSetCurrentMessage**, **vceStop**

Example

```
/* Append to an existing message */
extern CTAHD      CtaHd;
extern CTAQUEUEHD CtaQueueHd;

void myAppend (VCEHD vh, unsigned message, int maxseconds)
{
    CTA_EVENT event;
    unsigned  maxtime = maxseconds * 1000 ;

    vceSetCurrentMessage (vh, message) ;
    vceSetPosition (CtaHd, 0, VCE_SEEK_END, NULL) ;
    vceRecord (CtaHd, maxtime, VCE_INSERT, NULL);
    do
    {
        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
    } while (event.id != VCEEVN_RECORD_DONE);/* Ignore other events */
}
```

vceRecordMessage

Records a message in an open voice object.

Prototype

DWORD **vceRecordMessage** (VCEHD ***vh***, unsigned ***message***, unsigned ***maxtime***, VCE_RECORD_PARMS ****parms***)

Argument	Description
vh	Handle of an open voice object.
message	Message number to record.
maxtime	Maximum amount to record, in milliseconds. Specify VCE_NO_TIME_LIMIT to record with no time limit.

parms	Pointer to a parameter structure. Set this to NULL to use default values. The VCE_RECORD_PARMS structure is:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD DTMFabort;`
	`  INT32 gain;`
	`  DWORD novoicetime;`
	`  DWORD silencetime;`
	`  INT32 silenceampl;`
	`  DWORD beepfreq;`
	`  INT32 beepampl;`
	`  DWORD beeptime;`
	`  DWORD AGCenable;`
	`} VCE_RECORD_PARMS;`
	Refer to *VCE_RECORD_PARMS* on page 135 for a description of these fields.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is already active on the context.
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_OUTPUT_ACTIVE	Another function is controlling the MVIP output timeslot of the context.
CTAERR_RESOURCE_CONFLICT	Energy detector is in use in the ADI Service.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Message number is out of range for the object.
VCEERR_OUT_OF_INDICES	No free header entries in the destination VOX file.
VCEERR_PLAY_ONLY	Voice file was not opened for record.

Events

Event	Description
VCEEVN_RECORD_DONE	Recording completed. The value field of the event contains one of the following reasons or an error code: CTA_REASON_DIGIT Touch-tone digit was received and the corresponding bit in the DTMF abort parameter was set. CTA_REASON_FINISHED No more space is available or recording in overwrite mode reached the end of the existing message. CTA_REASON_NO_VOICE No voice detected for novoicetime milliseconds at the beginning of recording. CTA_REASON_RELEASED Stopped because the call was disconnected. CTA_REASON_STOPPED Stopped by **vceStop** or the current message was invalidated because one of the referenced voice objects was closed. CTA_REASON_TIMEOUT Time limit ***maxtime*** was reached. CTA_REASON_VOICE_END Silence detected for silencetime milliseconds after some voice detected. CTAERR_FUNCTION_NOT_AVAIL Recorded object has encoding that is not supported. CTAERR_RESOURCE_CONFLICT Energy detector is in use in the ADI service.

Details

vceRecordMessage starts recording to the specified ***message*** number in the voice object that ***vh*** specified. Any existing data in the message is erased. **vceRecordMessage** always returns immediately. If the return is SUCCESS, a VCEEVN_RECORD_DONE event occurs when recording completes.

After recording ends, call **vceGetContextInfo** to get the number of milliseconds actually recorded and look in the position field of the status.

The voice handle and message number become the current message for the context in which the voice handle was opened. After recording stops, you can resume recording at the end of the current message with **vceRecord**. Refer to *Recording* on page 19 for more information.

To trim data at the end of the recording:

- Use **vceSetPosition** to back up the desired amount.
- Use **vceErase** to erase data to the end of the message.

When the ADI energy detector is active, initiate a record operation (which uses the ADI Service) by setting novoicetime and silencetime to zero.

Refer to *VCE_RECORD_PARMS* on page 135 for more information about the record parameters.

See also

vceStop

Example

```
/* Record to an existing VOX file. */

extern CTAQUEUEHD CtaQueueHd;

void myRecordFile (CTAHD ctahd, char *filename, unsigned msgnum)
{
    VCEHD     vh;
    CTA_EVENT event;

    vceOpenFile (ctahd, filename, VCE_FILETYPE_VOX,
                 VCE_PLAY_RECORD, 0, &vh) ;
    vceRecordMessage (vh, msgnum, VCE_NO_TIME_LIMIT, NULL)  ;
    do
    {
        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
    } while (event.id != VCEEVN_RECORD_DONE);/* Ignore other events */
    vceClose (vh);
}
```

vceSetCurrentList

Defines the current message to be a list of messages.

Prototype

DWORD **vceSetCurrentList** (CTAHD ***ctahd***, VCE_MESSAGE ***msglist[]***, unsigned ***count***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.

msglist	Pointer to an array of the VCE_MESSAGE structure:
	`typedef struct`
	`{`
	`  VCEHD  vh;`
	`  unsigned  message;`
	`} VCE_MESSAGE;`
	Refer to the Details section for a description of these fields.

count	Number of elements in the ***msglist*** array.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is already active on the context.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_INVALID_HANDLE	One or more voice handles is invalid.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	One or more message numbers is out of range for the corresponding voice object.
VCEERR_MIXED_ENCODING	Messages have different encoding.

Details

vceSetCurrentList defines the current message for the context that ***ctahd*** specified. The current message can be a list of messages that are treated as one contiguous message.

Messages can be in different files or memory blocks but all must have been opened with ***ctahd*** as the context and all must have the same encoding.

The VCE_MESSAGE structure contains the following fields:

Field	Description
vh	Handle of an open voice object.
message	Message number in the voice object that vh specified.

To define a single current message, define a list with one element or use **vceSetCurrentMessage**.

Use **vceSetCurrentList** to select a message list prior to using the following position-oriented functions: **vceSetPosition**, **vcePlay**, and **vceRead**.

If a message in ***msglist*** does not exist, **vceSetCurrentList** returns SUCCESS as long as the message number is one that could exist in the corresponding vh field. If a message is out of range, the function returns VCEERR_INVALID_MESSAGE.

You cannot set the current message while play or record is active.

Use **vceGetCurrentList** to query the current message.

Example

```
/* Play digit string from multiple prompt files */

/* This routine starts speaking a digit string */
/* The spoken digits "oh" through "nine" are in message 0 in
 * 10 separate files. The open file handles are in an array vh[] where
 * vh[0] = "oh", vh[1] = "one", etc.
 */

extern CTAHD ctahd;

VCEHD Vh[10];       /* Array of open file handles */

void mySpeakDigitString (char *digits)
{
    VCE_MESSAGE msglist[50];
    unsigned    count;
    CTA_EVENT   event;

    /* Convert the digit string into a list of VCE_MESSAGE structs */
    for (count = 0;
        *digits != '\0' && count < sizeof msglist/sizeof msglist[0];
        digits++)
   {
        if (isdigit(*digits))
        {
            msglist[count].vh      = Vh[*digits - '0'];
            msglist[count].message = 0;
            ++ count;
        }
    }

    vceSetCurrentList(ctahd, msglist, count) ;
    vcePlay (ctahd, VCE_NO_TIME_LIMIT, NULL) ;
    do
    {

       ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
    } while (event.id != VCEEVN_PLAY_DONE);/* Ignore other events */
}
```

vceSetCurrentMessage

Defines a new current message.

Prototype

DWORD **vceSetCurrentMessage** (VCEHD ***vh***, unsigned ***message***)

Argument	Description
vh	Handle of an open voice object.
message	Message number in opened voice object.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is already active on the context.
CTAERR_INVALID_HANDLE	***vh*** is not a valid handle to an open voice object.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	***message*** is out of range for the specified voice object.

Details

vceSetCurrentMessage defines the current message for the context that ***vh*** implied. It is a special case of **vceSetCurrentList** where the list consists of one message.

Use this function to select a message prior to using position-oriented functions that include **vceSetPosition**, **vcePlay**, **vceRecord**, **vceErase**, **vceRead**, and **vceWrite**.

vceSetCurrentMessage returns SUCCESS for a message number that is valid for the file type of ***vh***. Otherwise, the return code is VCEERR_INVALID_MESSAGE.

To define the current message as all messages in ***vh***, concatenated in numerical order, specify VCE_ALL_MESSAGES as the message number.

You cannot set the current message while play or record is active.

Use **vceGetCurrentList** to query the current message.

Example

```
/* Play the first 5 seconds of the specified message */

void myPreviewMessage (CTAHD ctahd, VCEHD vh, unsigned message)
{
    CTA_EVENT event;

    vceSetCurrentMessage (vh, message);
    vcePlay (ctahd, 5000, NULL);
    do
    {
        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
    } while (event.id != VCEEVN_PLAY_DONE);/* Ignore other events */
}
```

vceSetPlayGain

Set the current play gain to a specified amount.

Prototype

DWORD **vceSetPlayGain** (CTAHD ***ctahd***, int ***db***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
db	The new gain, in decibels.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.

Details

vceSetPlayGain changes the volume (loudness) of message playback by adjusting the amplification, or gain, applied to the message being played.

The change in volume is expressed in dB units. Positive values represent amplification and negative values represent attenuation. A gain of zero (0) dB provides no amplification or attenuation. A change of one dB is the smallest change in volume discernible by the human ear.

The new gain is stored in the current context that ***ctahd*** identified. If play is currently active, the change takes effect immediately. To start the next play with the current gain, set the play ***gain*** parameter on the next play to VCE_CURRENT_VALUE.

Use **vceGetContextInfo** to get the current value.

Refer to *Playing* on page 17 for more information.

See also

vceSetPlaySpeed

Example

```
/*
 * Sample to adjust play gain by a specific amount. If play is currently
 * active, the change takes effect immediately. Otherwise, the change takes
 * effect if play is started with the 'gain' parameter set to VCE_CURRENT_VALUE.
 */

void myAdjustGain(CTAHD ctahd, int change)
{
    VCE_CONTEXT_INFO contextinfo;
    int   dB;

    vceGetContextInfo(ctahd, &contextinfo, sizeof contextinfo) ;
    dB = contextinfo.currentgain + change;
    vceSetPlayGain(ctahd, dB);
}
```

vceSetPlaySpeed

Changes the current play speed to a specified rate.

Prototype

DWORD **vceSetPlaySpeed** (CTAHD ***ctahd***, unsigned ***speed***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
speed	The new speed, expressed as a percentage, where 100 is normal speed.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.

Details

vceSetPlaySpeed changes the speed (rate) of message playback. The speed is expressed in hundredths of normal speed. For example, to change the speed to 160 percent of normal, set the speed value to 160.

Note: This functionality is not implemented for the PacketMedia HMP process.

The new speed is stored in the current context that ***ctahd*** identified. If play is currently active, the change takes effect immediately. To start the next play with the current speed, set the play ***speed*** parameter on the next play to VCE_CURRENT_VALUE.

Use **vceGetContextInfo** to get the current speed value.

The actual playback speed may not be the same as the value of the current speed in the context. The ability to change playback speed depends on the capabilities of the hardware and may not be the same for all encodings. The maximum speed is also limited by the value of the ***maxspeed*** play parameter.

Refer to *Playing* on page 17 for more information.

Speed control is available for the NMS ADPCM encodings (VCE_ENCODE_NMS***_xxx***) and the OKI ADPCM encodings (VCE_ENCODE_OKI***_xxx***).

If you are using an AG or a CG board to enable speed control of NMS encodings, refer to the DSP files section of the *ADI Service Developer's Reference Manual* for more information.

See also

vceSetPlayGain

Example

```
/*
 * Play a message where digit '6' changes to double speed and digit 4
 * restores normal speed. Any other digit stops play. The function returns
 * the reason for stopping.
 */

extern CTAHD      Ctahd;
extern CTAQUEUEHD CtaQueueHd;

void myPlay(VCEHD vh, unsigned message)
{
    CTA_EVENT      event;
    VCE_PLAY_PARMS parms;
    BOOL           done =  FALSE;
    char           digit ;

     /* Modify default play parms to allow speed up and to not abort
      * on digits 4 or 6.
      */
    ctaGetParms (Ctahd, VCE_PLAY_PARMID, &parms, sizeof parms);
    parms.maxspeed = 200;
    parms.DTMFabort = VCE_DIGIT_ANY & ~(VCE_DIGIT_4 | VCE_DIGIT_6);

    vcePlayMessage (vh, message, &parms);

    do
    {
        ctaWaitEvent (CtaQueueHd, &event, CTA_WAIT_FOREVER);
        if (event.id == ADIEVN_DIGIT_BEGIN)
        {
            if (event.value == '4')
                vceSetPlaySpeed (Ctahd, 100);
            else if (event.value == '6')
                vceSetPlaySpeed (Ctahd, 200);

            adiGetDigit (Ctahd, &digit); /* Remove and discard the digit */
        }
    } while (event.id != VCEEVN_PLAY_DONE);
}
```

vceSetPosition

Sets the current position in the current message.

Prototype

DWORD **vceSetPosition** (CTAHD ***ctahd***, int ***msec***, unsigned ***seekmode***, unsigned ****actual***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
msec	Number of milliseconds to reposition.
seekmode	Type of repositioning. Refer to the Details section for acceptable values.
actual	Pointer to returned new position.

Return values

Return value	Description
SUCCESS	
CTAERR_FUNCTION_ACTIVE	Playing or recording is active on the context.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.
VCEERR_NO_MESSAGE	No current message in the context.

Details

vceSetPosition changes the current position in the current message for the context that ***ctahd*** specified. The new position is returned in ***actual***, unless ***actual*** is NULL.

The ***msec*** argument can be a negative value when ***seekmode*** is VCE_SEEK_CUR. For other modes, ***msec*** must be a positive value.

Valid values for ***seekmode*** are:

Value	Description
VCE_SEEK_SET	Absolute position.
VCE_SEEK_CUR	Relative position (+ or -).
VCE_SEEK_END	Backward from end.

Use this function when preparing to use functions that operate at the current position, including **vcePlay**, **vceRecord**, **vceErase**, **vceRead**, and **vceWrite**.

You cannot change the position while play or record is active.

To query the current position without changing it, set ***msec*** to zero (0) and ***seekmode*** to VCE_SEEK_CUR, even when a function is active.

Note: During play or record, the current position is updated as each buffer is internally submitted for play or completed for record. When play ends, the values can be adjusted downward if not all of the bytes submitted to the device were actually played.

If you attempt to seek beyond the end or beginning of the current message, SUCCESS is returned. The new position is set to the current size if seeking beyond the end, or to zero (0) if seeking before the beginning.

See also

vceSetCurrentList, **vceSetCurrentMessage**

Example

```
/* Report the current position in the current message.*/

unsigned MyTell (CTAHD ctahd)
{
    unsigned position;

    if (vceSetPosition (ctahd, 0, VCE_SEEK_CUR, &position) != SUCCESS)
        return 0;
    return position;
}
```

vceSetWaveInfo

Loads a new entry in the WAVE encoding table.

Prototype

DWORD **vceSetWaveInfo** (CTAHD ***ctahd***, unsigned ***encoding***, VCE_WAVE_INFO ****waveinfo***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
encoding	Encoding ID.

waveinfo	Buffer to receive the following VCE_WAVE_INFO structure:
	`typedef struct`
	`{`
	`  DWORD  size;`
	`  WORD  format;`
	`  WORD  nchannels;`
	`  DWORD  samplespersec;`
	`  DWORD  datarate;`
	`  WORD  blocksize;`
	`  DWORD  bitspersample;`
	`} VCE_WAVE_INFO;`
	The VCE_WAVE_INFO structure fields are described in the Details section.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_SIZE	***size*** is less than the size of VCE_WAVE_INFO.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.
VCEERR_MIXED_ENCODING	A different entry for the same encoding exists in this or another context.

Details

vceSetWaveInfo adds an entry to the Voice Message service wave information table. The Voice Message service maintains this table to map encoding values to WAVE file header information. **vceCreateFile** uses this table to create a new WAVE file for a specified encoding. **vceOpenFile** uses this table to determine the encoding used in an existing WAVE file.

The table contains compiled-in entries for all of the encoding values in *vcedef.h*. If an attached play or record service makes another encoding available, add a new entry to the table using **vceSetWaveInfo** to play or record WAVE files with the new encoding.

The VCE_WAVE_INFO structure contains the following fields:

Field	Description
size	Size of the structure, in bytes. This is set to: `sizeof (VCE_WAVE_INFO)`
format	WAVE format type as defined by Microsoft, for example, WAVE_FORMAT_PCM (defined in the Microsoft header file *mmreg.h*).
nchannels	Number of discrete channels in the format. Use 1 for mono and 2 for stereo.
samplespersec	Sample rate in samples per second.
datarate	Average bytes per second.
blocksize	Minimum block size of the data. For PCM data, the block size is the number of bytes in a single sample.
bitspersample	Number of bits per sample.

See also

vceGetWaveInfo

Example

```
/*
 * Add 11 khz 8 bit stereo PCM encoding to the WAVE info table.
 * The encoding value is passed in.
 */

#define WAVE_FORMAT_PCM  1    /* As defined in mmreg.h */

DWORD myAdd11khzStereo(CTAHD ctahd, unsigned encoding)
{
    VCE_WAVE_INFO waveinfo;
    waveinfo.size           = sizeof waveinfo ;
    waveinfo.format         = WAVE_FORMAT_PCM;
    waveinfo.nchannels      = 2;       /* stereo           */
    waveinfo.samplespersec  = 11025;
    waveinfo.datarate       = 22050;   /* bytes per second */
    waveinfo.blocksize      = 2;       /* 2 8-bit channels */
    waveinfo.bitspersample  = 8;

    return vceSetWaveInfo(ctahd, encoding, &waveinfo);
}
```

vceStop

Stops the currently active play or record.

Prototype

DWORD **vceStop** (CTAHD ***ctahd***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_INVALID_SEQUENCE	Already stopping.
CTAERR_SVR_COMM	Server communication error.

Events

Event	Description
VCEEVN_PLAY_DONE or VCEEVN_RECORD_DONE	Playing or recording completed. The value field of the event contains CTA_REASON_STOPPED.

Details

vceStop sends a command to the play or record service to stop playing or recording. A VCEEVN_PLAY_DONE or VCEEVN_RECORD_DONE event with the value field containing CTA_REASON_STOPPED indicates successful completion of the command. If playing or recording ended before the stop command was sent, the event contains a different reason code.

Because play or record may have already ended, this function returns SUCCESS even if play and record is not active. Therefore, a return of SUCCESS does not guarantee that a DONE event is pending. It is up to the application to determine whether play or record was started and whether to expect a DONE event.

See also

vcePlay, **vcePlayList**, **vcePlayMessage**, **vceRecord**, **vceRecordMessage**

Example

```
/* Play message until stopped by keyboard event*/
extern CTAHD      CtaHd;
extern CTAQUEUEHD CtaQueueHd;

void myPlayMessage (VCEHD vh, unsigned message)
{
    CTA_EVENT event;

    vcePlayMessage (vh, message, NULL);
    do
    {
        ctaWaitEvent( CtaQueueHd, &event, CTA_WAIT_FOREVER);
        if (event.id == KBDEVN_KEY)
            vceStop(CtaHd);
    } while (event.id != VCEEVN_PLAY_DONE); /* Ignore other events */
}
```

vceUnloadPromptRules

Releases resources associated with the specified prompt rules.

Prototype

DWORD **vceUnloadPromptRules** (VCEPROMPTHD ***prompthandle***)

Argument	Description
prompthandle	Handle of the prompt table returned by **vceLoadPromptRules**.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_INVALID_HANDLE	***prompthandle*** is not a valid handle to an open prompt rules table.
CTAERR_SVR_COMM	Server communication error.

Details

vceUnloadPromptRules releases resources associated with the prompt rules table that ***prompthandle*** specified. The handle is not valid after this call.

See also

vceLoadPromptRules

Example

Refer to **vceBuildPromptList**.

vceWrite

Writes data at the current position in the current message.

Prototype

DWORD **vceWrite** (CTAHD ***ctahd***, BYTE ****buffer***, unsigned ***bytes***, unsigned ***insertmode***, unsigned ****byteswritten***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
buffer	Pointer to a buffer of data to write.
bytes	Amount of data to write.
insertmode	VCE_OVERWRITE or VCE_INSERT.
byteswritten	Pointer to the returned number of bytes written.

Return values

Return value	Description
SUCCESS	
CTAERR_DISK_FULL	There is not enough room on the disk to complete the write operation. No data was written.
CTAERR_FUNCTION_ACTIVE	Playing or recording is already active on the context.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_OPERATION	File type of the current message does not support insertion anywhere except at the end of a message. The operation is also invalid if the current message is a list.
VCEERR_NO_MESSAGE	Context contains no current message.
VCEERR_NO_SPACE	Insufficient room in the destination message.
VCEERR_OUT_OF_INDICES	Destination VOX file contains no free header entries.
VCEERR_PLAY_ONLY	Voice file was not opened for record.

Details

vceWrite copies the specified number of bytes of data from ***buffer*** to the current location in the current message of the context that ***ctahd*** specified. This function returns the number of bytes of data written in ***byteswritten***, unless ***byteswritten*** is NULL.

EDTX encoding types are not supported.

Set ***insertmode*** to VCE_OVERWRITE to replace existing data without changing the message size. In overwrite mode, if you attempt to write beyond the end of the current message, SUCCESS is returned and the current position is set to the end of the current message.

Set ***insertmode*** to VCE_INSERT to insert new data before the current position. If the current position is the end of the message, VCE_INSERT appends the new data to the current message. Inserting data anywhere except at the end of a message is allowed only on file types that support editing (currently only VOX files).

On completion, the current position is after the inserted data for either insert mode. The current position is advanced by the amount written, in millisecond units.

To convert between bytes and milliseconds, use **vceGetContextInfo** to get the frame size and frame time, and then use the following formulas:

bytes = (milliseconds / ***frametime***) * ***framesize***

milliseconds = (bytes / ***framesize***) * ***frametime***

See also

vceRead

Example

```
/* Append one message to another */

void myMsgCat (CTAHD ctahd, VCEHD destvh, unsigned destmsg,
                          VCEHD srcvh, unsigned srcmsg)
{
    BYTE    *buffer;
    unsigned  msec;
    unsigned  bytes;
    unsigned  byteswritten;

    /* Read source message to memory - see example in vceRead */
    myLoadMessage (ctahd, srcvh, srcmsg, &buffer, &msec, &bytes);

    /* Seek to end of destination */
    vceSetCurrentMessage (destvh, destmsg);
    vceSetPosition (ctahd, 0, VCE_SEEK_END, NULL);

    /* Append source message */
    vceWrite (ctahd, buffer, bytes, VCE_INSERT, &byteswritten);

    /* Release temporary buffer */
    free (buffer);

    if (byteswritten < bytes)
        printf("Insufficient space");
    return;
}
```

vceWriteMessageText

Writes the message text for a specified voice message in a VOX file.

Prototype

DWORD **vceWriteMessageText** (VCEHD ***vh***, unsigned ***message***, void ****buffer***, unsigned ***size***)

Argument	Description
vh	Handle of an open VOX file.
message	Any valid VOX file message number.
buffer	Pointer to a buffer of data to write.
size	Amount of data to write.

Return values

Return value	Description
SUCCESS	
CTAERR_DISK_FULL	There is not enough room on the disk to complete the write operation. No data was written.
CTAERR_SVR_COMM	Server communication error.
VCEERR_INVALID_MESSAGE	Message number is not in the valid range for VOX files (0 through 32767).
VCEERR_OUT_OF_INDICES	Destination VOX file contains no free header entries.
VCEERR_PLAY_ONLY	Voice file was not opened for record.
VCEERR_WRONG_FILE_TYPE	Not a VOX file.

Details

vceWriteMessageText stores auxiliary data for a specified message in a VOX file. The data can be text or binary. Existing data for ***message*** is replaced with the new data.

To erase message data for a specific ***message***, set ***size*** to zero (0). ***buffer*** can be NULL in this case. To erase all message data, set ***message*** number to VCE_ALL_MESSAGES and set ***size*** to zero (0).

vceWriteMessageText does not affect the current message for the context.

Rewriting with different size buffers can result in file fragmentation. To use the message text to store dynamic information (for example, a message time stamp), use a fixed size for all writes.

See also

vceCopyMessageText, **vceReadMessageText**

Example

```
/* Add text to a message in an open VOX file */

void myAddMessageText (VCEHD vh, unsigned message, char *text)
{
    unsigned size = strlen(text) ;

    vceWriteMessageText (vh, message, text, size);
}
```

7 Demonstration programs and utilities

Summary of the demonstration programs and utilities

The Voice Message service provides the following demonstration programs and utilities both as executable programs and as source code:

Program	Description
vcecopy	Copies messages from one voice file to another, and copies, deletes, and reads message text from a file.
vceinfo	Displays voice file information and message text.
vceplay	Demonstrates using the Voice Message service to play messages in voice files.
vcerec	Records one or more messages to a voice file and displays message text.
mkprompt	Creates a prompt rules table.
testpmpt	Tests a prompt rules table.
voxinfo	Displays information about a VOX file.

Voice file copy and convert: vcecopy

vcecopy is a universal voice file copy and conversion demonstration program. It includes the following features:

- Copies one or more messages from one voice file to another.
- Supports source and destination files of different types (any combination of VOX, WAVE, or flat).
- Supports source and destination files with different encodings. Conversion is supported for NMS ADPCM, IMA ADPCM, OKI ADPCM, G.726-compliant ADPCM, PCM (including all common WAVE file PCM), as well as mu-law and A-law.
- Makes the destination file louder or softer than the source.

Files are assumed to have an extension of *.vox* if no extension is specified. Input files can be anywhere in the CTA_DPATH path. The current directory is always searched first.

Usage

```
vcecopy source file destination file [options]
```

options can either precede or follow the two file names. Valid ***options*** are:

Encoding options

Option	Use this option to...
-e ***encoding***	Specify the encoding of the source file. Use this option only if the source is a flat (*.vce*) file. Entering -e? displays the following valid values: 1 NMS-16 2 NMS-24 3 NMS-32 4 NMS-64 10 MULAW 11 ALAW 13 PCM8M16 14 OKI-24 15 OKI-32 16 PCM11M8 17 PCM11M16 20 G726 22 IMA-24 23 IMA-32 24 GSM 26 G723-5.3 27 G723-6.4 28 G729A

File creation options

Option	Use this option to...
-x	Delete the destination file if it exists. This option is the default when the destination is not a VOX file. The default for VOX files is to write to an existing file, if any.
-n ***indices***	Specify the number of indices to allocate in a newly created VOX file. The default is the larger of 250 or 2 times the number of messages.

Conversion options

Option	Use this option to...
-g ***gain***	Specify the gain in dB. The default is 0 (no gain). The valid range is -24 to +24.
-s ***energy***	Delete silence from the beginning and end of a message containing an NMS encoding. ***energy*** is a relative threshold. The default is 25.
-c ***encoding***	Specify the encoding of the destination file when converting it from the source. **Note:** You can alternately specify a WAVE PCM encoding as rate M|S bits. For example, 22M16 for a coding of 22000 (nominal) samples/sec, Mono, 16 bits per sample. The rate can be 8, 11, 22, or 44.

Message number options

Option	Use this option to...
-m ***src msg*** [,***dest msg***]	Specify source message numbers and destination message numbers. The default is to copy all messages in the source file. If you specify only a source message, the destination message number is the same as the source message number. Source message numbers and destination message numbers are ignored if the respective file type is not VOX.

Message text options

Option	Use this option to...
-t N	Not copy message text.
-t O	Copy only message text.
-t F	Specify that the source file is a text file containing lines of the form ***message number***:***text***.

File options

Option	Use this option to...
-i ***input file type*** (VOX | WAV | VCE) -o ***output file type*** (VOX | WAV | VCE)	Specify the file type. By default, the file type is inferred from the file name. Any file type other than *.vox* or *.wav* is considered VCE (flat).

Examples

```
vcecopy prompts temp -s25 -x
```

Copies all messages in *prompts.vox* to *temp.vox* and trims silence.

```
vcecopy one.wav numbers.vox -c3 -m1 -g-6
vcecopy two.wav numbers.vox -c3 -m2 -g-6
```

Copies messages from two wave files to a single VOX file with NMS32 encoding, and reduces the volume by 6 dB.

```
vcecopy -x -e10 test.vce message1
```

Copies a message in a flat file named *test.vce* that has mu-law encoding to a VOX file named *message1.vox*. A new file is created.

```
vcecopy greeting.vox test.vce -e 14 -i VCE
```

Copies a flat file named *greeting.vox* containing OKI-24 encoding to a flat file named *test.vce*.

```
vcecopy input.wav output.wav -c 22m16
```

Copies a wave file named *input.wav* to a new file named *output.wav* with 22 kHz, 16 bit encoding.

```
vcecopy prompts.txt prompts.vox -t file
```

Copies message descriptions from a text file to a VOX file.

Voice file information: vceinfo

Displays voice file information.

Usage

```
vceinfo [options] filename
```

where ***options*** are:

Option	Use this option to...
-e ***encoding***	Specify the encoding of the file. This option is needed only for flat files. Entering -e? displays the following valid values: 1 NMS-16 2 NMS-24 3 NMS-32 4 NMS-64 10 MULAW 11 ALAW 13 PCM8M16 14 OKI-24 15 OKI-32 16 PCM11M8 17 PCM11M16 20 G726 22 IMA-24 23 IMA-32 24 GSM 26 G723-5.3 27 G723-6.4 28 G729A
-f ***filetype***	Specify the file type. File types are *.vox*, *.wav*, and *.vce*. By default, the file type is inferred from the file name. A file having an extension other than *.vox* or *.wav* is considered VCE (flat). If no file type is specified and the file name does not have an extension, *.vox* is assumed.
-n	Not show message information.
-?	Display an on-screen list of options.

Featured functions

vceClose, **vceGetHighMessageNumber**, **vceGetMessageSize**, **vceGetOpenInfo**, **vceOpenFile**, **vceSetCurrentMessage**

Example

```
vceinfo ctademo
Natural Access Voice File Info Utility V 1.5 (Aug 30 2001)

File: 'ctademo.vox'
Type: NMS Vox
Encoding: NMS 24 Kbps
High Msg: 18
      Msg   0 is  1080 ms  "zero"
     Msg   1 is   940 ms  "one"
     Msg   2 is   940 ms  "two"
      Msg   3 is   960 ms  "three"
      Msg   4 is   920 ms  "four"
      Msg   5 is   960 ms  "five"
      Msg   6 is   920 ms  "six"
      Msg   7 is   920 ms  "seven"
      Msg   8 is   920 ms  "eight"
      Msg   9 is   940 ms  "nine"
      Msg  10 is  1940 ms  "the digits dialed were"
      Msg  11 is  2300 ms  "the calling party's number is"
      Msg  12 is  1580 ms  "thank you for calling"
      Msg  13 is  3700 ms  "press 1 to record, 2 to play, 3 to hang up"
      Msg  14 is  1820 ms  "there is nothing to play"
      Msg  15 is  2340 ms  "you are being called by the demo"
      Msg  16 is  4040 ms  "please enter the extension of the party you"
      Msg  17 is  2240 ms  "I'll try that extension now"
      Msg  18 is   960 ms  "goodbye"
```

Voice file play: vceplay

Plays voice files. The user interface from keyboard or touch-tones allows control of pause or resume, forward or rewind, louder or softer, and faster or slower. Refer to *Playing* on page 17 for more information.

Usage

```
vceplay [options] filename [additional filenames]
```

where ***options*** are:

Option	Use this option to...
-b ***board***	Specify the board number. Default = 0.
-e ***encoding***	Specify the encoding of the file. This option is needed only for flat files. -e? displays a list of choices.
-m ***start*** [***:end***]	Specify the message or messages to play. If only ***start*** is specified, then only that message is played. If ***end*** is also specified, then all messages from start to end, inclusive, are played.
-p ***protocol***	Wait for and connect to an incoming call. -p specifies the call control protocol. Default is LPS0. To use *vceplay* with a phone directly connected to a DID port on an AG 2000/C board, specify the NOCC protocol. For example: `vceplay -pnocc filename`
-s [***stream:***]***slot***	Specify the port (DSP) address. Default = 0:0. You need to specify only the slot.
-?	Display an on-screen list of program options.

Keyboard and touch tone control

You can enter the following single-digit commands with either a keystroke or a touch tone.

Note: If you enter a 1 (one) at the beginning of a message, you return to the previous message.

Command	Description
1	Beginning of message or previous message
2	Next message
3	Next file
4	Rewind 2 seconds
5	Pause or resume (toggle)
6	Forward 2 seconds
7	Slower
8	Softer
9	Faster
0	Louder
*	Redisplay commands
# or **ESC**	Exit

Featured functions

vceClose, **vceGetContextInfo**, **vceGetHighMessageNumber**, **vceGetMessageSize**, **vceOpenFile**, **vcePlay**, **vceSetCurrentMessage**, **vceSetPlayGain**, **vceSetPlaySpeed**, **vceSetPosition**, **vceStop**

Examples

```
vceplay -m2:4 american.vox
```

Plays messages 2, 3, and 4 in *american.vox.*

```
vceplay c:\test\hello.wav
```

Plays the message in *hello.wav.*

```
vceplay -e3 test.vce
```

Plays a flat file named *test.vce*. The encoding of the file is NMS24.

```
vceplay -e?
```

Displays the encoding values and their names.

```
vceplay .\*.wav
```

Plays the wave files in the current directory.

Voice file record: vcerec

Records voice files. Refer to *Recording* on page 19 for more information.

Usage

```
vcerec [options] filename [additional filenames]
```

where ***options*** are

Option	Use this option to...
-b ***board***	Specify the board number. Default = 0.
-e ***encoding***	Specify the encoding of the file. This option is needed only for flat files. Entering -e? displays the following valid values: 1 NMS-16 2 NMS-24 3 NMS-32 4 NMS-64 10 MULAW 11 ALAW 13 PCM8M16 14 OKI-24 15 OKI-32 16 PCM11M8 17 PCM11M16 20 G726 22 IMA-24 23 IMA-32 24 GSM 26 G723-5.3 27 G723-6.4 28 G729A
-f ***filetype***	Specify the file type. File types are VOX, WAV, and VCE. By default, the file type is inferred from the file name. A file with an extension other than *.vox* or *.wav* is considered VCE (flat). If no file type is specified and the file name does not have an extension, *.vox* is assumed.
-i ***voxindices***	Specify the number of index entries in a newly created VOX file (48 to 6500). The default number of index entries in a VOX file is 250.
-m ***start***:***end***	Specify the message or messages to record. If only ***start*** is specified, only that message is recorded. If ***end*** is also specified, all messages from start to end, inclusive, are recorded.
-p ***protocol***	Wait for and connect to an incoming call. -p specifies the call control protocol. Default is LPS0. To use *vcerec* with a phone directly connected to a DID port on an AG 2000/C board, specify the NOCC protocol. For example: `vcerec -pnocc filename`
-r ***seconds***	Specify the maximum seconds per message. If not specified, there is no limit.
-s [***stream:***]***slot***	Specify the port (DSP) address. Default = 0:0. You need to specify only the slot.
-x	Overwrite the existing destination file. The default is to merge into an existing file, if any.
-?	Display an on-screen list of program options.

Keyboard and touch tone control

You can enter the following single-digit commands with either a keystroke or a touch tone:

Command	Description
1	Restart message
2	Next message
5	Pause or resume (toggle)
# or **ESC**	Exit

Featured functions

vceClose, **vceCreateFile**, **vceEraseMessage**, **vceGetContextInfo**, **vceGetHighMessageNumber**, **vceGetMessageSize**, **vceGetOpenInfo**, **vceOpenFile**, **vceRecord**, **vceRecordMessage vceSetCurrentMessage**, **vceSetPlayGain**, **vceSetPlaySpeed**, **vceSetPosition**, **vceStop**

Examples

```
vcerec test.wav
```

Re-records one message to *test.wav.*

```
vcerec -pnocc -m0:9 digits
```

Records 10 messages to *digits.vox*. Does not wait for an incoming call.

```
vcerec -14 -fVCE  oki24.vox
```

Records to a flat file named *oki24.vox* using OKI-24 encoding.

```
vcerec -e?
```

Displays the encoding values and their names.

Make prompt: mkprompt

Creates a prompt rules table. *mkprompt* compiles prompt rule source files (*.ptx*) into prompt rule tables (*.tbl*). The *.tbl* file is created only if there are no errors during the compile. If errors occur, they are reported with the line number where the error occurred.

For more information, refer to *Creating a prompt rules table* on page 142.

Usage

```
mkprompt source[.ptx] [dest[.tbl]]
```

where:

Argument	Description
source[*.ptx****]***	Source text file to be compiled. If no extension is specified, *.ptx* is assumed.
[dest[*.tbl****]]***	Optional output file name. If no output file is specified, the source file name is used with the *.tbl* extension. If no extension is specified, *.tbl* is assumed.

Example

```
mkprompt american
```

Creates *american.tbl* from the source *american.ptx* file.

Test prompt: testpmpt

Tests a prompt rules table. For more information, refer to *Creating a prompt rules table* on page 142.

testpmpt tests new prompt rules without having to pre-record the actual voice messages. After rules are created or edited in a prompt rules text file (*.ptx*), they are compiled into a rules table (*.tbl*) file using *mkprompt*.

You must create a text description for each message in a text file with the same root name as the *.tbl* file. The text description has the following format:

nn*:*description

where ***nn*** is the message number being described.

Use the *vcecopy* utility to copy the message text into the *.vox* file that contains, or will contain, the recorded prompts.

When this utility is executed, the rules table and the voice or text file are loaded. You are prompted to enter a text string that the prompt builder processes.

Usage

```
testpmpt [options]
```

where ***options*** are

Option	Use this option to specify the...
-p ***table***	Prompt table name (assumes a *.tbl* extension). The default name is *american*.
-v ***voicefile***	Name of the voice file containing the message text (assumes a *.vox* extension). The default is the same name as the table.
-t ***text***	Name of the file containing prompts in text form (assumes a *.txt* extension). *testpmpt* does not look for a voice file if you specify a text file.

Example

```
testpmpt -p american
Natural Access Test Prompt Utility V 1.2 (Dec 11 1997)
               Prompt table = american.tbl
               Prompt text from american.vox
Enter text: 1/15/98
January fifteenth nineteen ninety eight
Enter text: 12345
twelve thousand three hundred forty five
Enter text: 12:34:34
twelve thirty four and thirty four seconds P.M.
Enter text: 12:34
twelve thirty four P.M.
Enter text: $416.23
four hundred sixteen dollars and twenty three cents
Enter text:
```

VOX file information: voxinfo

Displays information about a specified NMS VOX file, including the header fields and details about each message.

The unit size in VOX files is the frame. A frame represents 20 ms when speech is encoded with one of the NMS voice encodings, and 10 ms for most other encodings.

For more information, refer to *Overview of VOX file format* on page 137.

Usage

```
voxinfo filename
```

where ***filename*** is the NMS VOX file for which you want to display information.

Example

```
voxinfo mprompt
NMS Voice File Information Utility     Version 1.3    Aug 30 2001
Copyright (c) 1988-2001 NMS Communications.  All Rights Reserved.

File:                          MYPROMPT

Compression rate:                  2   (NMS 24 Kbps)
Maximum number of indices:        48
Number of active indices:          2
Number of freed indices:           0
Number of indices in use:          2
Number of available indices:      46
Highest message number:            0
Total number of frames:           99   (2.0 seconds)
Total number of bytes:          6650
Number of active messages:         1
Number of message text entries:    1

--------------------ACTIVE INDICES----------------------
Idx#  Msg#     Frames     Bytes     Start byte  End byte
0     0        0x62       0x17bc    0x200       - 0x19bb
1     T-0      0x1        0x3e      0x19bc      - 0x19f9

-------------MESSAGE INFORMATION-----------------
Msg#  Seconds   Index Cnt   Frame Cnt   Byte Cnt
0      2.0       1           98          6076

-------------MESSAGE TEXT INFORMATION----------------
Msg#  Frame Cnt  Byte Cnt
0     1          62

End of Summary
```

8 Errors, events, and reason codes

Alphabetical error summary

All Natural Access functions return a status code. If the return code is not SUCCESS (0), it is an error code indicating that the function failed and the reason for the failure. Error codes can also appear in the value field of a DONE event. Use the CTA_IS_ERROR macro to determine if a value is an error.

Voice Message service error codes are defined in the *vcedef.h* include file. The error codes are prefixed with VCEERR.

For errors beginning with CTAERR, refer to the *Natural Access Developer's Reference Manual*.

The following table lists the Voice Message service errors. All errors are 32 bits.

Error name	Hex	Decimal	Description
VCEERR_BAD_PROMPT_COMMAND	0x3000A	196618	*Problem*: There is an error in the prompt rules table referred to by the ***prompthandle*** argument in **vceBuildPromptList**, or an invalid ***method*** was used in **vceBuildPromptList**. *Solution:* Correct the rules table entry or use one of the following ***methods*** of translation: 0, 1, 2, 3, or 4. Refer to **vceBuildPromptList** for a description of these table-specific methods.
VCEERR_CONVERSION_FAILED	0x3000C	196620	An error was returned by the voice conversion library.
VCEERR_INVALID_MESSAGE	0x30000	196608	*Problem*: A message number is out of range for the file type of the voice object to which the message belongs. For example, zero (0) is the only valid message number for WAVE files. *Solution:* Refer to the range of valid message numbers.

Error name	Hex	Decimal	Description
VCEERR_INVALID_OPERATION	0x30002	196610	One of the following situations occurred: • An attempt was made to record, write, or erase a list. For example, there was more than one message in the current message list. • An attempt was made to record or write to the message VCE_ALL_MESSAGES. • An attempt was made to copy a message to itself. • An attempt was made to insert a message in a flat file, WAVE file, or memory block. • An attempt was made to erase a piece from the middle of a message in a flat file, WAVE file, or memory block.
VCEERR_MIXED_ENCODING	0x30005	196613	One of the following situations occurred: • Voice objects in the list passed to **vceSetCurrentList** did not all have the same encoding. • An attempt was made to copy a message to an object with a different encoding. • An attempt was made to define a WAVE encoding that was already defined with different values. All threads in a process must use the same values.
VCEERR_NO_MESSAGE	0x30001	196609	A function that operates on the current message (for example, **vcePlay**) was called when there was no current message.
VCEERR_NO_SPACE	0x30004	196612	The destination message of **vceCopyMessage** or **vceConvertMessage** cannot be expanded to the necessary size. This can happen if the voice object is a memory block, or if the voice object is a flat file and the message is replacing an existing message.
VCEERR_OUT_OF_INDICES	0x30008	196616	*Problem*: The function was unable to create a new segment in a *.vox* file because there were no more segment descriptors in the file header. *Solution:* Create a new file with a large number of indices (**vceCreateFile**). Copy all messages from the existing file to the new file.
VCEERR_PLAY_ONLY	0x30003	196611	An edit or record function was attempted on a voice file open for play only.

Error name	Hex	Decimal	Description
VCEERR_PROMPT_BUILD_FAIL	0x3000B	196619	*Problem*: **vceBuildPromptList** encountered an error in the prompt rules table. *Solution:* Ensure that you are using the correct prompt rules file (*.tbl*) and that it is not damaged.
VCEERR_UNSUPPORTED_ENCODING	0x30009	196617	*Problem*: An encoding format was used that is not supported by Natural Access, or an attempt was made to convert to or from an encoding format that is not supported by the conversion library. *Solution:* Refer to *Encoding descriptions* on page 149 and *Encoding and WAVE information* on page 150 for more information. Refer to **vceConvertMessage** for a table of supported conversions.
VCEERR_WRONG_ENCODING	0x30007	196615	*Problem*: The encoding specified as an argument to **vceOpenFile** does not match the encoding value contained or derived from information in the file header. *Solution:* It is not necessary to specify an encoding for VOX files or WAVE files.
VCEERR_WRONG_FILE_TYPE	0x30006	196614	*Problem*: **vceOpenFile** failed because the file header does not contain the correct format for the specified VOX or WAVE file. This error is also returned by **vceDefineMessages** if the voice object is not a flat file or memory block.

Numerical error summary

The following table lists the Voice Message service errors:

Hex	Decimal	Error name
0x30000	196608	VCEERR_INVALID_MESSAGE
0x30001	196609	VCEERR_NO_MESSAGE
0x30002	196610	VCEERR_INVALID_OPERATION
0x30003	196611	VCEERR_PLAY_ONLY
0x30004	196612	VCEERR_NO_SPACE
0x30005	196613	VCEERR_MIXED_ENCODING
0x30006	196614	VCEERR_WRONG_FILE_TYPE
0x30007	196615	VCEERR_WRONG_ENCODING
0x30008	196616	VCEERR_OUT_OF_INDICES
0x30009	196617	VCEERR_UNSUPPORTED_ENCODING
0x3000A	196618	VCEERR_BAD_PROMPT_COMMAND
0x3000B	196619	VCEERR_PROMPT_BUILD_FAIL
0x3000C	196620	VCEERR_CONVERSION_FAILED

Events

Voice Message service events are defined in the *vcedef.h* include file. Events are prefixed with VCEEVN.

Event name	Hex	Decimal	Description
VCEEVN_PLAY_DONE	0x32101	205057	Play completed.
VCEEVN_RECORD_DONE	0x32102	205058	Recording completed.

Reason codes

The following table lists the Voice Message service reason codes:

Reason code name	Hex	Decimal	Description
CTA_REASON_DIGIT	0x1004	4100	A touch-tone digit was received and the corresponding bit in the DTMF abort parameter was set.
CTA_REASON_FINISHED	0x1001	4097	During play, the end of the current message or current list was reached. During record, no more space was available in the message, or recording in overwrite mode reached the end of the existing message. For other functions, this reason code indicates successful completion.
CTA_REASON_NO_VOICE	0x1005	4101	Recording stopped because no voice was detected at the beginning of recording. You can specify the maximum length of time that silence can occur before recording is stopped.
CTA_REASON_RECOGNITION	0x1008	4104	Play or record stopped because of a speech recognition event.
CTA_REASON_RELEASED	0x1007	4103	Function stopped because the call was disconnected.
CTA_REASON_STOPPED	0x1002	4098	Play or record was stopped by **vceStop**, or the current message was invalidated because one of the referenced voice objects was closed.
CTA_REASON_TIMEOUT	0x1003	4099	***maxtime*** time limit specified in **vcePlay**, **vceRecord**, or **vceRecordMessage** was reached.
CTA_REASON_VOICE_END	0x1006	4102	Recording stopped because the maximum length of silence after voice (audio energy) was detected.

For more information on any of these reason codes, refer to the following functions: **vcePlay**, **vcePlayList**, **vcePlayMessage**, **vceRecord**, or **vceRecordMessage**.

9 Voice message service parameters

Overview of the Voice Message service parameters

Some Natural Access functions have parameters that you can modify to:

- Enable or disable function features.
- Adapt a function for exceptional configurations.

For example, when recording speech data you can specify:

- The subset of DTMF keys entered by the telephone caller that abort the function.
- The gain applied to the input signal.
- An initial timeout in which the caller must start speaking before the function terminates.
- The amount of silence after a caller has stopped speaking before the function terminates.
- The beep tone frequency, amplitude, and duration.
- Automatic gain control settings.

These parameters are grouped together into structures for convenience and efficiency. Each parameter structure has a default value that is sufficient for many configurations.

VCE_PLAY_PARMS

Dependent functions

vcePlay, **vcePlayList**, **vcePlayMessage**

Field name	Type	Default	Units	Description
DTMFabort	DWORD	0xFFFF	Mask	Enables you to control which DTMF tones abort play. Refer to DTMFabort valid values.
gain	INT32	0	dB	Gain applied to the encoded audio. Set to VCE_CURRENT_VALUE to use the most recent gain setting. The valid range for AG boards and CG boards is -54 to 24. The valid range for QX boards is -48 to 42.
maxspeed	DWORD	100	Percent	Maximum play speed. The valid range for AG boards and CG boards is 100 to 200. This value is ignored for encoding types that do not support speed modification. Not used for QX 2000 boards.
speed	DWORD	100	Percent	Initial speed. Set to VCE_CURRENT_VALUE to use the most recent speed setting. The valid range for AG boards and CG boards is 50 to maxspeed. This value is ignored for encoding types that do not support speed modification. Not used for QX 2000 boards.

DTMFabort valid values

Form a value by using the OR operator with any of the following:

Digit	Name	Value
0	VCE_DIGIT_0	0x0001
1	VCE_DIGIT_1	0x0002
2	VCE_DIGIT_2	0x0004
3	VCE_DIGIT_3	0x0008
4	VCE_DIGIT_4	0x0010
5	VCE_DIGIT_5	0x0020
6	VCE_DIGIT_6	0x0040
7	VCE_DIGIT_7	0x0080
8	VCE_DIGIT_8	0x0100
9	VCE_DIGIT_9	0x0200
*	VCE_DIGIT_STAR	0x0400
#	VCE_DIGIT_POUND	0x0800
A	VCE_DIGIT_A	0x1000
B	VCE_DIGIT_B	0x2000

Digit	Name	Value
C	VCE_DIGIT_C	0x4000
D	VCE_DIGIT_D	0x8000

VCE_RECORD_PARMS

Dependent functions

vceRecord, **vceRecordMessage**

Field name	Type	Default	Units	Description
AGCenable	DWORD	0	Mask	Flag to enable automatic gain control (AGC). Set to 1 to enable AGC and 0 to disable it.
beepampl	INT32	-20	dBm	Amplitude of the record beep tone. The valid range for AG boards and CG boards is-54 to 3. The valid range for QX boards is -90 to 0.
beepfreq	DWORD	1000	Hz	Frequency of the record beep tone. Use zero (0) to disable the beep. The valid range for AG boards and CG boards is 200 to 3600. The valid range for QX boards is 0 to 4000.
beeptime	DWORD	200	ms	Duration of the record beep tone. 0 disables the beep. The valid range for AG boards and CG boards is 0 to 65535. The valid range for QX boards is 0 to 8000.
DTMFabort	DWORD	0xFFFF	Mask	Control of the DTMFs abort record. Refer to *DTMFabort valid values* on page 134.
gain	INT32	0	dB	Gain applied to the signal before it is encoded. The valid range for AG boards and CG boards is -54 to 24. If AGC is enabled, this is the initial gain when recording starts. The valid range for QX boards is -42 to 48.
novoicetime	DWORD	5000	ms	Maximum length of silence at the beginning of a recording before recording is stopped with CTA_REASON_NO_VOICE. Use zero (0) to disable this timer. The valid range for AG boards, CG boards, and QX boards is 0 to 65535.
silenceampl	INT32	-45	dBm	Maximum signal level that is considered to be silence. The valid range for AG boards and CG boards is -51 to -15. The valid range for QX boards is -45 to 0.
silencetime	DWORD	3000	ms	Maximum length of silence after audio energy has been detected before record is stopped with CTA_REASON_VOICE_END. Use zero (0) to disable this timer. The valid range for AG boards, CG boards, and QX boards is 0 to 65535.

Note: The parameters presented in this table override the corresponding fields in the ADI_RECORD category. ADI_RECORD contains additional fields you can use to control automatic gain control (AGC) characteristics.

10 VOX file format

Voice file segments

A voice file is partitioned into segments. A message can consist of more than one segment if it was edited, or if there were some existing re-usable segments when the message was created.

One of the following descriptors in the file header describes each segment of the file:

Segment descriptor	Description
Active	Messages currently in existence. The message number orders the active descriptors. When there are multiple segments for one message, the descriptors' order corresponds to the order of the segments in the message.
Freed	Sections of the file that are available for re-use, in order of their position in the file.
Unused	Descriptors that contain nothing.

Active descriptors are first, followed by freed descriptors, and then unused descriptors.

Overview of VOX file format

This topic presents an overview of a VOX file format and describes:

- Index structure
- Frame sizes

A VOX file consists of a 32 byte fixed size header section, followed by an array of 10 byte segment descriptors called indices. The number of segments is specified at file creation and can range from 48 to 6500. Segment sizes are in units of frames. The number of bytes in a frame depends on the encoding.

The following table illustrates the VOX file format:

Encoding	VTYPE	2 bytes	**Header** (32 bytes)
Total number of indices	TOTLIDX	2	
Number of indices in use	USEDIDX	2	
Number of indices in messages	ACTVIDX	2	
Number of indices in free pool	FREEIDX	2	
Highest message number in use	HIGHMSG	2	
Total number of bytes	TOTLBYTE	4	
Total number of frames	TOTLFRM	4	
Reserved		12	
Active indices 10 x ACTVIDX bytes			**Indices** [10 x TOTLIDX bytes] (see Index structure)
Freed indices 10 x (USEDIDX - ACTVIDX) bytes			
Unused indices 10 x FREEIDX bytes			
Actual voice frames in segments of various length where an index describes each segment in either the Active Index list or the Freed Index list.			**Frames** [TOTLFRM x framesize] (see *Summary of frame sizes* on page 138)

Index structure

The following table details the index structure:

Message number	MSGNO	2 bytes	Index structure [10 bytes]
Starting byte in file	STRTBYTE	4	
Number of frames in segment	NFRAMES	4	

Summary of frame sizes

The following table details the frame sizes of common encodings:

Encoding	Description	msec/frame	bytes/frame
1	NMS ADPCM 16 kbit/s	20	42
2	NMS ADPCM 24 kbit/s	20	62
3	NMS ADPCM 32 kbit/s	20	82
4	Framed PCM 64 kbit/s	20	162
10	mu-law 64 kbit/s	10	80

Encoding	Description	msec/frame	bytes/frame
11	A-law 64 kbit/s	10	80
13	PCM 8 kss mono 16-bit	10	160
14	OKI ADPCM 24 kbit/s	10	30
15	OKI ADPCM 32 kbit/s	10	40
16	PCM 11 kss 8-bit	10	110
17	PCM 11 kss 16-bit	10	220
20	G.726 ADPCM 32 kbit/s	10	40
22	IMA ADPCM 24 kbit/s	10	36
23	IMA ADPCM 32 kbit/s	10	46
24	MS-GSM 13 kbit/s	80	130
26	G.723 5.3 kbit/s	30	20
27	G.723 6.3 kbit/s	30	24
28	G.729A 8 kbit/s	10	10
46	mu-law 64 kbit/s formatted in EDTX frames	variable typically 10	variable Max = 82
47	A-law 64 kbit/s formatted in EDTX frames	variable typically 10	variable Max = 82
48	CCITT G.726 ADPCM 32 kbit/s formatted in EDTX frames	variable typically 10	variable Max = 42
49	G.729A 8 kbit/s formatted in EDTX frames	variable typically 10	variable Max = 12
50	G.723 frames formatted with EDTX headers	variable typically 30	variable Max = 26
51	G.723 5.3 kbit/s frames formatted with EDTX headers	variable typically 30	variable Max = 22
52	G.723, 6.4 kbit/s frames formatted with EDTX headers	variable typically 30	variable Max = 26

Note: kss = kilo-samples per second.

Use the *voxinfo* utility to display information about a VOX file.

11 Customizing prompt rules

Prompt builder files

This topic presents the runtime and source files delivered with the prompt builder.

Runtime files

File name	Description
american.tbl	Compiled rules table for standard American English. The prompt builder uses the rules contained in this file to build a list of message numbers that can be sent directly to **vcePlayList** to say dates, times, numbers, and dollar amounts. The supplied American rules table contains the following criteria: • Any number from 0 to 999,999,999,999 with or without commas. • Decimal fractions of any length. • Dates in MM/DD, MM/DD/YY, or MM/DD/YYYY format. • Time in HH:MM or HH:MM:SS 24 hour format where hour is 0 to 23. • Dollar amounts in $DDD,DDD.CC format, with or without cents or commas. • Months and days of the week in the following format: MON, TUE, WED, THU, FRI, SAT, SUN, JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC. • The words: AT, AND. Any space-separated list of these formats is transmitted in succession separated by pauses. Strings starting with @ (at sign) are spoken as individual digits, with pauses at spaces and hyphens. All other characters are ignored.
american.vox	Default voice message library for the standard American prompts. This file is opened and specified in **vcePlayList** in conjunction with the message list output of **vceBuildPromptList**. This file can be re-recorded to change the voice that speaks the prompt builder prompts. This file also contains a brief text description of each message.
mkprompt.exe	Prompt table compiler used to compile prompt rule text files (*.ptx*) into prompt rules tables (*.tbl*). For more information, refer to the *mkprompt* utility.
testpmpt.exe	Prompt test utility, used when developing new rules or editing existing ones. This utility enables you to try rules without having to record voice messages. For more information, refer to the *testpmpt* utility.

Note: *mkprompt* and *testpmpt* are *.exe* files in Windows only.

Source programs

File name	Description
american.ptx	Source rule file for the standard American rules. It can be compiled into *american.tbl* using *mkprompt*. New rules can be added and then compiled into a *.tbl* file.
testpmpt.c	Source to the prompt test utility. This file demonstrates how an application uses the prompt builder functions.

To re-record *american.vox*, use *vceinfo* to list the message descriptions in *american.vox*. Follow the list to create a new *american.vox*.

Creating a prompt rules table

A prompt rules table file (*.tbl*) is created from a prompt rules source file (*.ptx*) using *mkprompt*. *mkprompt* expects a text source file that contains prompt table commands.

A prompt rules table is composed of a list of commands, keywords, and labels that:

- Perform operations on an input string
- Output message numbers to a list
- Affect flow of control

The following table lists the commands:

Command	Description
CONVERT	Converts a string to a message number or label. Appends a computed message number to an output list or jumps to a computed label.
FIND	Parses an input string. Performs conditional jumps.
OUTPUT	Appends message numbers to an output list. Can also be used to terminate processing.
TEST	Performs conditional jumps based upon the result of a string or integer comparison.

In addition to the commands, the following elements are used:

Element	Description
;	Identifies the comment field. All remaining text on the line is ignored.
label=num	Assigns a value to a label. This element is primarily used to assign a name to each message in the prompt message file.
label:	Marks the command as a target for a jump from a CONVERT, TEST, or FIND command.

The syntax of the commands is:

```
command [option1|option2] value
```

where:

command	Name of the command.
[option1|option2]	Optional keywords. The available choices are within the brackets. Select only one. That is, specify either ***option1*** or ***option2***, but not both. Both choices can be omitted, and the default option is used.
value	A number, string, or label. This value is dependent on the command syntax.

CONVERT

The CONVERT command converts a substring to a value used as either a message number to append to the output list of messages or as a label to which to jump.

Usage

CONVERT [CASE|NOCASE] ***subval*** [MESSAGE|CALL|GOTO] ***baseval*** [EXIT|CONT]

CASE	Do a case sensitive character subtract.
NOCASE	Do a case insensitive character subtract. This is the default.
subval	Subtract the specified integer or character from the string.
MESSAGE	Generate and append a message number to the output list.
CALL	Generate a label to call as a subroutine.
GOTO	Generate a label to which to jump. This is the default.
baseval	Use the specified base label or message number.
EXIT	Return after MESSAGE.
CONT	Continue when done (MESSAGE). This is the default.

Note: EXIT and CONT apply only if MESSAGE is used.

Description

The message number or label is constructed by first subtracting ***subval*** from the input string and then adding ***baseval*** to the result. If the input string is to be treated as an integer, as many as five digits are used. If the input string is to be treated as a character string, the first character is used, as is the first character of ***subval***. By default, the input string and ***subval*** are treated as integers. If the keyword CASE or NOCASE is used, they are treated as characters.

If a message number is generated, it is appended to the end of the output list. If the EXIT keyword is specified, processing returns to the line following the command that called EXIT. If the EXIT keyword is not specified, processing resumes at the next line in the prompt rules table.

If a GOTO label is generated, processing continues with the command at the computed label.

If a CALL label is generated, processing continues with the command at the computed label. When control is returned, processing resumes at the next line in the prompt rules table.

Examples

```
TEN = 30                  ;message number to say TEN.
CONVERT 1 MESSAGE TEN     ;subtract 1 from passed digit
                          ;and add to label value TEN,
                          ;output resulting message
                          ;number then goto next cmd.
                          ;Note that case 0 is ruled out
                          ;beforehand.
LETTERA = 60              ;message number for letters
                          ;'a' - 'z'
CONVERT NOCASE "A" MESSAGE LETTERA
                          ;convert letters to appropriate
                          ;message number
CONVERT 0 GOTO PLACE
                          ;say first, second etc.
```

FIND

The FIND command searches for a character in the input string and uses its position as a beginning or end pointer of a substring to be passed to the next command. The command performs a conditional jump or error return.

Usage

FIND [FORWARD|BACKWARD] [CASE|NOCASE] [***occur***] ["***letter***"] [INCLUDE|EXCLUDE] [FULL|LEFT ***count***|RIGHT ***count***] [[FOUND|NOTFOUND] [CALL|GOTO|ERROR] [***label***] [[ELSE]EXIT]

FORWARD	Search forward from the beginning of the input string. This is the default.
BACKWARD	Search backward from the end of the input string.
CASE	Do a case-sensitive search.
NOCASE	Do a case-insensitive search. This is the default.
occur	Occurrence number to find, such as find the second occurrence of a colon (:). An absolute position can also be specified if "***letter***" is not specified.
"***letter***"	The letter to find. If not specified, the search is absolute positioning based on ***occur***.
INCLUDE	Include the found character when passing the string. This is the default.
EXCLUDE	Exclude the found character when passing the string.
FULL	Pass the full string. This is the default.
LEFT	Pass count characters to the left of the found character.
RIGHT	Pass count characters to the right of the found character.
count	Pass a number of characters to the specified command. The maximum is 127.
FOUND	Perform a CALL or GOTO if the string is found. This is the default.
NOTFOUND	Perform a CALL or GOTO if the string is not found.
CALL	Call a FOUND or NOTFOUND ***label*** as a subroutine.
GOTO	Go to a FOUND or NOTFOUND ***label***. This is the default.
ERROR	Return VCEERR_PROMPT_BUILD_FAIL immediately to the application. This keyword enables the rules to notify the application that an invalid input string was specified.
label	Command to CALL or GOTO if found or not found. The computed substring is passed to the command. If ***label*** is omitted, the substring is passed to the next command.
ELSE	Allow optional NO-OP keyword before EXIT to enhance readability.
EXIT	Return if FOUND or NOTFOUND condition is not met.

Description

The character search can be performed forward from the beginning of the input string, or backward from the end of the input string, as specified by the keywords FORWARD or BACKWARD. The ***n***th occurrence of the character, as specified by ***occur***, is sought. If no character is specified by ***letter***, then ***occur*** is used as an offset from the beginning or end of the string. Once this position is established, the substring to the right or to the left of this position can be passed to the next command. By default, the entire string is passed.

The substring to be passed is specified by the RIGHT, LEFT, or FULL keyword. For RIGHT or LEFT, the number of characters in the substring is specified by ***count***. Use a large ***count*** to pass all characters to the right or left. The found letter is passed as part of the substring unless the keyword EXCLUDE is used.

If a GOTO jump is performed, the substring is passed to the command at the specified ***label***, where processing continues.

If a CALL jump is performed, the substring is passed to the command at the specified ***label***, where processing continues. When control is returned, the input string originally presented to the FIND command is restored and passed to the command on the next line in the prompt rules table.

If the search condition is not met and EXIT is not specified, the unmodified input string is passed to the next command.

The FIND command can be used as an unconditional GOTO. (Refer to the examples.)

Examples

```
FIND 2 ":" RIGHT 2 EXCLUDE FOUND SAYSECONDS
```

If a second colon is found, pass 2 characters to the right to a routine that says the seconds count.

```
FIND 1 "$" FULL FOUND SAYDOLLAR
```

Find the first occurrence of a $ (dollar sign). If found, pass the full string to the dollar routine, otherwise fall through to the next record.

```
FIND GOTO SAYMORE
```

Unconditional GOTO.

```
FIND FORWARD 1 "$" RIGHT EXCLUDE 127
```

Extract the substring to the right of a $ (dollar sign) but continue to the next command.

OUTPUT

The OUTPUT command appends the specified message numbers to the output list. From zero to three messages can be specified in the OUTPUT command. If more than three messages are needed, use additional OUTPUT commands.

Usage

OUTPUT [***msg*** [,***msg*** [,***msg***]]] [EXIT|CONT|QUIT]

msg	A message number. As many as three messages can be specified.
EXIT	Return to the previous command when done.
CONT	Continue to the next line. This is the default.
QUIT	Stop processing commands.

Description

If this keyword is specified...	Then processing....
CONT	Resumes at the next line in the prompt rules table.
EXIT	Returns to the line following the command that called this one.
QUIT	Terminates and SUCCESS is returned to the application.

Examples

```
OUTPUT OCLOCK,LETTERA,LETTERM
```

Output the following messages: o'clock, a, m.

```
OUTPUT MSGSUNDAY
```

Output the following message: Sunday.

```
OUTPUT DEGREES,FAHR EXIT
```

Output the following messages: degrees, fahrenheit. Then return.

```
OUTPUT HELLO QUIT
```

Output the following message: hello. Then stop processing commands.

```
OUTPUT QUIT
```

Stop processing commands.

TEST

The TEST command compares the input string with a specified value and jumps if the test evaluation is true.

Usage

TEST [GREATER|LESS|EQUAL|NOT] [CASE|NOCASE] ***cmpval*** [CALL|GOTO|ERROR] [***label***] [[ELSE] EXIT|CONT]

GREATER	True if substring is greater than ***cmpval***.
LESS	True if substring is less than ***cmpval***.
EQUAL	True if substring is equal to ***cmpval***.
NOT	True if substring is not equal to ***cmpval***. This is the default.
CASE	Do a case-sensitive compare.
NOCASE	Do a case-insensitive compare. This is the default.
cmpval	Value with which to compare substring.
CALL	Call the command at ***label***, if true.
GOTO	Go to the command at ***label***, if true. This is the default.
ERROR	Return VCEERR_PROMPT_BUILD_FAIL immediately.
label	Execute specified command if the evaluation is true.
ELSE	Allow specified optional NO-OP keyword before EXIT to enhance readability.
EXIT	Return if a ***label*** was specified and the evaluation is false, or no ***label*** was specified and the evaluation is true.
CONT	Continue if the evaluation is false. This is the default.

Description

If the keyword CASE or NOCASE is specified, a character comparison is performed. Otherwise an integer comparison of up to five digits is performed.

If the test evaluation is true, processing continues with the command at the specified ***label***. If a CALL jump is specified, processing resumes at the next line in the prompt rules table once control is returned.

If the test evaluation is false or no ***label*** is specified, processing continues directly at the next line unless the EXIT keyword is used. In this case, processing resumes at the line following the command that called this one.

Examples

```
TEST GREATER 99 CALL DO100 ;if >100 process hundreds
```

If the input string contains a number greater than 100, call the subroutine that handles them.

```
TEST EQUAL NOCASE "SUN" GOTO SAYSUNDAY
```

If the input string starts with SUN, go to the routine to output the message for SUNDAY.

```
TEST NOT 0 EXIT
```

Return if the input string is not equal to 0 (zero).

Prompt rules table processing

Prompt rules table processing occurs when an application passes a string and a prompt table handle to **vceBuildPromptList**. Prompt table commands are processed line by line. Each command receives an input string passed to it by the previous command. The initial input string is passed in by the application. The initial line is determined by the ***method*** parameter passed to **vceBuildPromptList**.

Modify control flow with the GOTO, CALL, EXIT, QUIT, and ERROR keywords. The GOTO keyword specifies a jump to a specified label. Command processing then continues line by line. The CALL keyword also specifies a jump to a specified label. A CALL jump results in the following actions:

1. Saves the current state: input string, current line.
2. Jumps to the label.
3. Processes commands until an EXIT signifying a return to this saved state is encountered.
4. Restores the saved state of the input string and current line.
5. Continues processing at the following line.

Use the EXIT keyword to return to the previously saved state that consists of the saved input string and command line. The EXIT keyword can be optionally preceded by the keyword ELSE when used with the FIND and TEST commands.

Use the QUIT keyword only with the OUTPUT command. It terminates processing of the prompt rules table and returns SUCCESS to the application.

Use the ERROR keyword only with the FIND and TEST commands. It terminates processing of the prompt rules table and returns VCE_PROMPT_BUILD_ERROR to the application.

String operations

A substring of the input string can be compared to a string or an integer with the TEST command. It can be translated to a message or label with the CONVERT command. The input string can be parsed with the FIND command. The FIND command is the only command that affects the input string itself by passing a substring to the next command. The other commands pass an unmodified string . Once a string is shortened by the FIND command, it cannot be restored unless an EXIT is used to return from a CALL jump.

Processing termination

Command processing terminates when there are no commands left to process. In this case, SUCCESS is returned to the application. Command processing also terminates when either a QUIT or an ERROR is encountered.

12 Voice encoding formats

Encoding descriptions

Value	Name	Description
1	VCE_ENCODE_NMS_16	NMS ADPCM, 16 kbit/s
2	VCE_ENCODE_NMS_24	NMS ADPCM, 24 kbit/s
3	VCE_ENCODE_NMS_32	NMS ADPCM, 32 kbit/s
4	VCE_ENCODE_NMS_64	NMS framed PCM, 64 kbit/s
10	VCE_ENCODE_MULAW	mu-law, 64 kbit/s
11	VCE_ENCODE_ALAW	A-law, 64 kbit/s
13	VCE_ENCODE_PCM8M16	PCM, 8 kilo-samples per second, 16 bits per sample
14	VCE_ENCODE_OKI_24	OKI ADPCM, 6 kilo-samples per second, 24 kbit/s *
15	VCE_ENCODE_OKI_32	OKI ADPCM, 8 kilo-samples per second, 32 kbit/s
16	VCE_ENCODE_PCM11M8	PCM, 11 kilo-samples per second, 8 bits per sample, mono *
17	VCE_ENCODE_PCM11M16	PCM, 11 kilo-samples per second, 16 bits per sample, mono *
20	VCE_ENCODE_G726	CCITT G.726 ADPCM, 32 kbit/s
22	VCE_ENCODE_IMA_24	IMA/DVI ADPCM, 8 kilo-samples per second, 24 kbit/s *
23	VCE_ENCODE_IMA_32	IMA/DVI ADPCM, 8 kilo-samples per second, 32 kbit/s
24	VCE_ENCODE_GSM	MS-GSM, 13 kbit/s. This encoding format is not supported on QX boards. *
26	VCE_ENCODE_G723_5	G.723, 5.3 kbit/s. This encoding format is supported on CG boards only. *
27	VCE_ENCODE_G723_6	G.723, 6.3 kbit/s This encoding format is supported on CG boards only. *
28	VCE_ENCODE_G729A	G.729A, 8 kbit/s This encoding format is supported on CG boards only. *
46	VCE_ENCODE_EDTX_MULAW	mu-law, 64 kbit/s formatted in EDTX frames. This encoding format is supported on CG boards only.
47	VCE_ENCODE_EDTX_ALAW	A-law, 64 kbit/s formatted in EDTX frames. This encoding format is supported on CG boards only.
48	VCE_ENCODE_EDTX_G726	CCITT G.726 ADPCM, 32 kbit/s formatted in EDTX frames. This encoding format is supported on CG boards only.
49	VCE_ENCODE_EDTX_G729A	G.729A, 8 kbit/s formatted in EDTX frames. This encoding format is supported on CG boards only.
50	VCE_ENCODE_EDTX_G723	G.723 frames formatted with EDTX headers. This encoding format is supported on CG boards only.

Value	Name	Description
51	VCE_ENCODE_EDTX_G723_5	G.723, 5.3 kbit/s frames formatted with EDTX headers. This encoding format is supported on CG boards only.
52	VCE_ENCODE_EDTX_G723_6	G.723, 6.4 kbit/s frames formatted with EDTX headers. This encoding format is supported on CG boards only.

* The PacketMedia HMP process does not support this encoding format.

Encoding and WAVE information

		Encoding information		WAVE information					
Name	Value	Frame size (bytes)	Frame time (msec)	WAVE type	Num chan	Sample rate	Data rate	Block size	Bits/ sample
VCE_ENCODE_NMS_16	1	42	20	0x38	1	8000	2100	42	2
VCE_ENCODE_NMS_24	2	62	20	0x38	1	8000	3100	62	3
VCE_ENCODE_NMS_32	3	82	20	0x38	1	8000	4100	82	4
VCE_ENCODE_NMS_64	4	162	20	0x38	1	8000	8100	162	8
VCE_ENCODE_MULAW	10	80	10	7	1	8000	8000	1	8
VCE_ENCODE_ALAW	11	80	10	6	1	8000	8000	1	8
VCE_ENCODE_PCM8M16	13	160	10	1	1	8000	16000	2	16
VCE_ENCODE_OKI_24	14	30	10	0x17	1	6000	3000	1	4
VCE_ENCODE_OKI_32	15	40	10	0x17	1	8000	4000	1	4
VCE_ENCODE_PCM11M8	16	110	10	1	1	11025	11025	1	8
VCE_ENCODE_PCM11M16	17	220	10	1	1	11025	22050	2	16
VCE_ENCODE_G726	20	130	80	0x64	1	8000	4000	1	4
VCE_ENCODE_IMA_24	22	36	10	0x11	1	6000	3041	256	4
VCE_ENCODE_IMA_32	23	46	10	0x11	1	8000	4055	256	4
VCE_ENCODE_GSM	24	130	80	0x31	1	8000	1625	65	NA
VCE_ENCODE_G723_5	26	20	30	NA	NA	NA	NA	NA	NA
VCE_ENCODE_G723_6	27	24	30	0x111	1	8000	800	24	NA
VCE_ENCODE_G729A	28	10	10	0x83	1	8000	1000	10	NA

		Encoding information	
Name	**Value**	**Frame size (bytes)**	**Frame time (msec)**
VCE_ENCODE_EDTX_MULAW	46	variable Max = 82	variable uncompressed = 10
VCE_ENCODE_EDTX_ALAW	47	variable Max = 82	variable uncompressed = 10
VCE_ENCODE_EDTX_G726	48	variable Max = 42	variable uncompressed = 10
VCE_ENCODE_EDTX_G729A	49	variable Max = 12	variable uncompressed = 10
VCE_ENCODE_EDTX_G723	50	variable Max = 26	variable uncompressed = 30
VCE_ENCODE_EDTX_G723_5	51	variable Max = 22	variable uncompressed = 30
VCE_ENCODE_EDTX_G723_6	52	variable Max = 26	variable uncompressed = 30

Note: If you are using an AG board or a CG board, refer to the *ADI Service Developer's Reference Manual* for a list of DSP files that are required for each encoding format.

Index

W