

Basilisk

Call Management API

Version 2.1.1

Developer's Reference Guide

March, 2007.

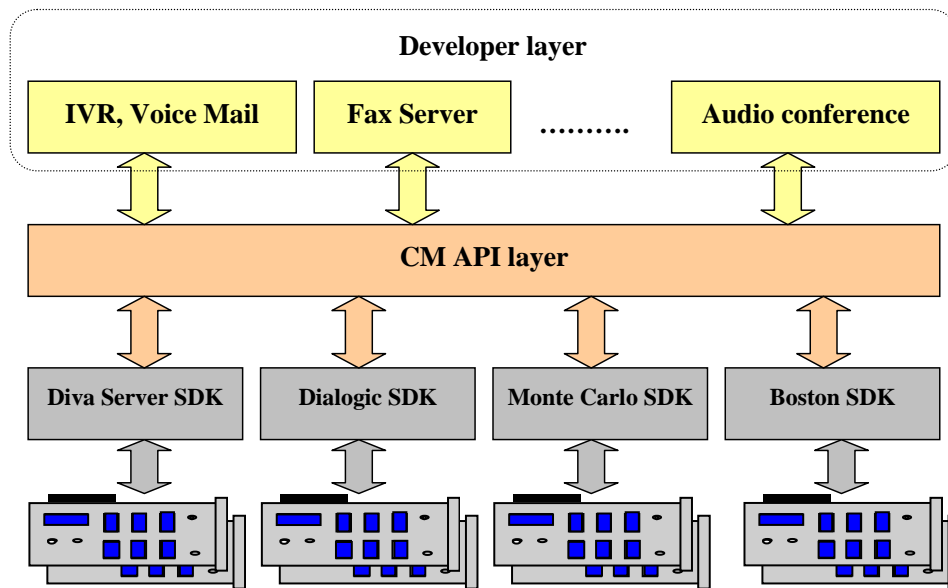
CONTENTS

CONTENTS	II
BASILISK API OVERVIEW	3
BRIEF OVERVIEW.....	3
MIGRATING TO NEWER VERSIONS.....	3
PRODUCT FILES AND LICENSING POLICY.....	3
SUPPORTED HARDWARE.....	4
CM API FUNCTIONALITY BREAKDOWN	4
INTRODUCTION.....	4
FUNCTION INDEX.....	5
<i>CMAddToConference()</i>	5
<i>CMClearDigitBuffer()</i>	6
<i>CMCloseChannel()</i>	6
<i>CMCreateConference()</i>	6
<i>CMDeleteConference()</i>	8
<i>CMDetectSpeech()</i>	8
<i>CMGetChassisInfo()</i>	9
<i>CMGetConfereeCount()</i>	9
<i>CMGetDigits()</i>	10
<i>CMGetDSPResCount()</i>	10
<i>CMGetVersion()</i>	11
<i>CMInitLib()</i>	11
<i>CMMakeCall()</i>	16
<i>CMOpenChannel()</i>	17
<i>CMPlayAudioFile()</i>	17
<i>CMPlayDTMFTones()</i>	19
<i>CMPlayIndexedAudioFile()</i>	19
<i>CMReceiveFaxFile()</i>	21
<i>CMRecordAudioFile()</i>	21
<i>CMRemoveFromConference()</i>	23
<i>CMSendFaxFiles()</i>	24
<i>CMSetConfereeStatus()</i>	24
<i>CMSetOnHook()</i>	25
<i>CMStopChannel()</i>	26
<i>CMUninitLib()</i>	26
<i>CMWaitCall()</i>	27
DATA STRUCTURE INDEX.....	27
<i>BOARDDESC</i>	28
<i>CHASSIS</i>	29
<i>CHASSIS2</i>	30
<i>DEVDESC</i>	30
<i>FaxParams</i>	31
APPENDIX A (HELPER FUNCTION INDEX)	33
<i>CMConvertMp3ToRawFile()</i>	33
<i>CMConvertMp3ToWaveFile()</i>	33
<i>CMConvertRawToRawFile()</i>	34
<i>CMConvertRawToWaveFile()</i>	34
<i>CMConvertWaveToRawFile()</i>	35
<i>CMConvertWaveToWaveFile()</i>	35
APPENDIX B (AUDIO FORMATS)	36

Basilisk API Overview.

Brief overview.

Basilisk Call Management API (further referred to as CM API throughout this guide) developed in the framework of project *Basilisk* is a high-level programming interface formally presented in a set of C/C++ functions (a library). It is designed as the middle level software for the development of the layered-structure call controlling applications using the CM API as unified basis for working with a variety of hardware platforms. The main goal of CM API library is to facilitate the process of CT application development allowing the developer to abstract all hardware communication details and to concentrate on a particular call handling scenario implementation.



Another great advantage offered by the CM API is its embedded audio conversion library. The developers don't have to stick with audio codecs and bit-rates allowed for each particular media adapter – if needed, all audio format conversion is done internally “on the fly”. Right now any of the following voice formats may be equally used for audio streaming for all hardware platforms:

- VOX (OKI Dialogic raw audio)
- WAV (Microsoft wave audio files)
- MP3 files (for audio playback only)

This guide assumes that the developer has basic knowledge of Microsoft Visual C++ 6.0 or higher, the library files included with CM API have been tested with and are specific to Microsoft Visual C++ 6.0. Changes may need to be made to the library files when using higher versions of Microsoft Visual C++ compilers. Contact CM API tech support and maintenance team to get the updated versions of the library parts.

Migrating to newer versions.

CM API might be extended on the time-to-time basis to incorporate new features and functionality to better support underlying hardware platforms. All new versions will be backwards compatible which guarantees that all applications developed with the use of earlier CM API versions will also work with the new versions though recompilation of some application parts with the newer C-headers and library file may be required.

Product files and licensing policy.

CM API dynamically loads/unloads required driver DLLs of all supported hardware installed on the current chassis. This allows to correctly handle cases when some of the drivers are not installed or no specific board is found during start-up. CM API consists of the following files:

- **CMLib.h** – C/C++ header file defines CM API constants, structures and function prototypes. It internally includes all other CM include files.

- **CMLib.lib** – C/C++ dynamic library including references to CM API functions from the CMLib.dll dynamic link library.
- **CMLib.dll** – CM API dynamic link library.
- **CMLib.key** – CM API software license file, it contains security information on the total number of channels and boards the CM library is allowed to work with. Must be placed in the root directory of the 'C:' logical drive prior to any CM API usage.

All library parts except the license file CMLib.key are distributed freely. A valid license file restricts the maximum number of channels handled by the CM library simultaneously and it also includes the serial number of one board of any kind (as it is returned by the hardware driver) which must be present on the pc where the library is to be used. If the total number of channels allowed by the CMLib.key exceeds the total number of channels available on the board whose serial number is included in the license file, then other boards of any kind may be added to the same pc to work together forming a common pool of channels. E.g., if the total number of channels in the license is 50 and the board which must be present is Pika PrimeNet MM-1E1 (30 PRI channels) then the rest of the channels allowed by the license (20 channels) may be used by placing on the chassis another PrimeNet MM-1E, or a Dialogic D/300JCT-E1, or any set of boards of any connectivity type (analog, E1/T1).

If either the license file or the board required by the CMLib.key is absent then any attempt to open a channel via corresponding CM API calls will fail as required by the software distribution policy. To obtain the license file contact the CM API maintenance team or an authorized CM API reseller.

Supported hardware.

With a few exceptions CM API equally supports Dialogic, Eicon Diva Server, Brooktrout and Pika single (voice, fax, conference) and combined media boards. The complete list of hardware supported is shown in Table 1. Any combination of the boards listed below may run on same chassis under control of a single application which gives the developer an ultimate hardware flexibility and scalability offered by none of competitors. The total number of adapters per chassis is limited only by specific manufacturer requirements.

Table 1. Supported hardware breakdown.

Hardware platform\ Application type	Voice	FAX	Conferencing
Dialogic (Former Intel-Dialogic)	D/4PCI, D/4PCIUF, D/41JCT-LS, VFX series, Springware E1/T1 boards, DM3 series	D/4PCIUF, VFX series	DM/V480A-2T1, DM/V600A-2E1, Springware E1/T1 boards + DCB series
Eicon Diva Server	Diva Server Universal series, Diva Server V-series	Diva Server Universal series	Diva Server Universal series, Diva Server V-series
Pika	InLine MM, Daytona MM series, PrimeNet MM series	InLine MM, Daytona MM series, PrimeNet MM series	Daytona MM series, PrimeNet MM series
Brooktrout	TR1000 series, TR1034 series	TR1000 series, TR1034 series	N/A

CM API Functionality breakdown.

Introduction.

While call establishment and termination is generally an asynchronous multi-step process the CM API implements the multithreaded synchronous programming model. It means that with a few exceptions all API functions block application execution until the function completes. The synchronous mode requires that the application controls each channel from a separate thread. A call termination in the synchronous mode initiated by either the application (e.g. by calling asynchronously the *CMStopChannel()* to abort a blocking CM API function) or call disconnection or failure. Either of these terminations can occur at any point in the process of setting up a call and during any call state.

The channels provided by the CM API can be seen as a pool of resources (ports) and the application sees each channel as a dedicated resource (port). Once it is seized the channel cannot be acquired by another thread. The library does all required resources allocation/releasing internally, it also performs even more sophisticated tasks

such as distribution and bridging of conference members and groups of conferees amongst the available DSPs for a specific hardware platforms, etc.

Function index

This chapter gives a description of the CM API functions provided that all functions are listed alphabetically for ease of use. Some library functions use special structures that are defined in the *CmLib.h* header file.

CMAddToConference()

Syntax

```
BOOL CMAddToConference(CM_CONFHANDLE *pconf, CM_HANDLE *pdevh, int *piAttr);
```

Parameters

pconf

[in] Pointer to a conference handle returned by *CMCreateConference()*.

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*. This handle identifies the conferee to be added to the conference specified by the *pconf* handle.

piAttrl

[in] Pointer to an integer that specifies the conferee desired status. This can be any of the following values defined in the *CmLib.h*:

CM_Conferee_Monitor – a single duplex conferee, this kind of a conference party can only listen to the other parties.

CM_Conferee_Active – a full duplex conferee (default), this kind of a conference party can listen to and speak with the other parties.

CM_Conferee_Coach – the *Coach*, this is a so-called supervision link party who can listen to all the parties but can be heard by the *Pupil* only. There can be only one supervision link and *Coach* in a conference. An attempt to add one more *Coach* will produce an error. To make another conferee the *Coach*, first assign the existing *Coach* a new attribute (single or full duplex) with the *CMSetConfereeStatus()* function, then set the *Coach* attribute to a new conferee with another *CMSetConfereeStatus()* call. That will link the new *Coach* to the existing *Pupil*.

CM_Conferee_Pupil – the *Pupil*, this is another supervision link party who can listen to and can be heard by all the parties including the *Coach*. There can be only one supervision link and *Pupil* in a conference. An attempt to add one more *Pupil* will produce an error. To make another conferee the *Pupil*, first assign the existing *Pupil* a new attribute (single or full duplex) with the *CMSetConfereeStatus()* function, then set the *Pupil* attribute to a new conferee with another *CMSetConfereeStatus()* call. That will link the new *Pupil* to the existing *Coach*.

If **piAttr* is assigned a value different from the listed above the conferee will be created with the default attribute (*CM_Conferee_Active*).

Returns

If this function succeeds the return value is *TRUE*, otherwise the return value is *FALSE*.

Description

This function adds a new conferee identified by the *pdevh* channel handle to the active conference specified by the *pconf* handle. The conference must be created with the *CMCreateConference()* prior to this function call. The channel state stays unchanged after this function terminates. Same caller may take part in several conferences simultaneously. To remove the caller identified by the *pdevh* handle use the *CMRemoveFromConference()* function.

See also

[CMCreateConference\(\)](#), [CMRemoveFromConference\(\)](#), [CMSetConfereeStatus\(\)](#), [CMOpenChannel\(\)](#)

CMClearDigitBuffer()

Syntax

```
void CMClearDigitBuffer(CM_HANDLE *pdevh);
```

Parameters

pdevh
[in] Pointer to a channel handle returned by *CMOpenChannel()*.

Returns

Nothing.

Description

This function causes the digits present in the internal digit buffer of the channel specified by the *pdevh* to be flashed. This function is internally called by the *CMSetOnHook()* before disconnecting an active call.

See also

[CMOpenChannel\(\)](#), [CMGetDigits\(\)](#), [CMSetOnHook\(\)](#)

CMCloseChannel()

Syntax

```
void CMCloseChannel(CM_HANDLE *pdevh);
```

Parameters

pdevh
[in] Pointer to a channel handle returned by *CMOpenChannel()*.

Returns

Nothing.

Description

This function closes a channel that was previously opened with *CMOpenChannel()* and releases all resources allocated by the library internally and via the underlying hardware driver API calls. The *pdevh* channel handle can no longer be used after the *CMCloseChannel()* function returns. If the channel is not in idle state this function internally calls *CMSetOnHook()*.

See also

[CMOpenChannel\(\)](#), [CMSetOnHook\(\)](#)

CMCreateConference()

Syntax

```
typedef struct _CM_CONFHANDLE CM_CONFHANDLE;  
  
CM_CONFHANDLE* CMCreateConference(CM_HANDLE *pdevh, int *piAttr);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*. This handle identifies the first conferee added to the conference being created. This parameter must not be *NULL* since the library doesn't create an empty conference.

piAttrl

[in] Pointer to an integer that specifies the first conferee desired status. This can be any of the following values defined in the *CmLib.h*:

CM_Conferee_Monitor – a single duplex conferee, this kind of a conference party can only listen to the other parties.

CM_Conferee_Active – a full duplex conferee (default), this kind of a conference party can listen to and speak with the other parties.

CM_Conferee_Coach – the *Coach*, this is a so-called supervision link party who can listen to all the parties but can be heard by the *Pupil* only. There can be only one supervision link and *Coach* in a conference. An attempt to add one more *Coach* will produce an error. To make another conferee the *Coach*, first assign the existing *Coach* a new attribute (single or full duplex) with the *CMSetConfereeStatus()* function, then set the *Coach* attribute to a new conferee with another *CMSetConfereeStatus()* call. That will link the new *Coach* to the existing *Pupil*.

CM_Conferee_Pupil – the *Pupil*, this is another supervision link party who can listen to and can be heard by all the parties including the *Coach*. There can be only one supervision link and *Pupil* in a conference. An attempt to add one more *Pupil* will produce an error. To make another conferee the *Pupil*, first assign the existing *Pupil* a new attribute (single or full duplex) with the *CMSetConfereeStatus()* function, then set the *Pupil* attribute to a new conferee with another *CMSetConfereeStatus()* call. That will link the new *Pupil* to the existing *Coach*.

If **piAttr* is assigned a value different from the listed above the first conferee will be created with the default attribute (*CM_Conferee_Active*).

Returns

If function succeeds, the return value is a pointer to the valid conference handle. If the function fails, the return value is *NULL*. The handle returned by *CMCreateConference()* is *opaque*, that is the application is not supposed to know about its contents. All subsequent library calls for this conference must pass the handle as an argument.

Description

This function creates a new conference on a board having conferencing resources of any supported kind that is installed on the pc and returns a handle to be used in subsequent calls to conferencing routines of CM API. If this operation fails due to some reason (e.g. no free resources as indicated by the *CMGetDSPResCount()* function) then *NULL* is returned.

¹**N.B.:** It is important to know that due to hardware platforms incompatibility it is impossible to create a conference shared by hardware devices of different manufacturers. For example, a conference created on a Pika board cannot share resources with Eicon and Dialogic boards installed on the same chassis. Thus by selecting a network type (see values of the *NetType* member of *BOARDDESC* structure) with the first conferee channel handle *pdevh* we also restrict the network type of other conferees who can join this conference. An attempt to add a conferee with incompatible network type will fail.

²**N.B.:** Net types *NT_PKLSDT* and *NT_PKISDN* are compatible for conferencing.

See also

[CMOpenChannel\(\)](#), [CMSetConfereeStatus\(\)](#), [CMDeleteConference\(\)](#), [CMGetDSPResCount\(\)](#), [BOARDDESC](#)

CMDeleteConference()

Syntax

```
BOOL CMDeleteConference(CM_CONFHANDLE *pconf);
```

Parameters

pconf
[in] Pointer to a conference handle returned by *CMCreateConference()*.

Returns

If this function succeeds the return value is *TRUE*, otherwise the return value is *FALSE*.

Description

This function deletes a conference that was previously created with the *CMCreateConference()* and releases all resources allocated by the library internally and via the underlying hardware driver API calls. The *pconf* conference handle can no longer be used after the *CMDeleteConference()* function returns. It is a responsibility of the application to remove from the conference all the conferees but one (the last conferee is removed when the conference is deleted) with the *CMRemoveFromConference()* before calling this function. Otherwise some resource leaks may occur and the further library behavior is unexpected.

N.B.: For conferences realized on Pika platform all conferees including group bridges are automatically removed by the *CMDeleteConference()*. But if the application itself removes all conferees but one with the *CMRemoveFromConference()* before deleting a conference then no collision will occur, so the common rule of conference deletion is preferred.

See also

[CMCreateConference\(\)](#), [CMRemoveFromConference\(\)](#)

CMDetectSpeech()

Syntax

```
int CMDetectSpeech(CM_HANDLE *pdevh, int iTimeout);
```

Parameters

pdevh
[in] Pointer to a channel handle returned by *CMOpenChannel()*.
iTimeout
[in] Specifies the maximum timeout delay to detect speech on the channel in seconds.

Returns

When the function terminates its return value indicates the termination reason. This can be any of the following bit-flags defined in the *CmLib.h*.

TM_NORMTERM – Human speech has been detected
TM_USRSTOP – The function has been aborted by the application with the *CMStopChannel()*
TM_TONE – Fax tone has been detected
TM_MAXTIME – Maximum allowed timeout delay to detect speech exceeded. (No speech)
TM_LCOFF – Loop current OFF detected on an analog line
TM_PATTERN – Disconnect signal detected
TM_ERROR – An IO-device error occurred

Description

This function initiates human speech detection on the channel specified by *pdevh* handle. It tries to detect speech for the duration of the *iTimeout*. *CMDetectSpeech()* returns when either human speech or a fax tone

is positively detected. Another termination causes are disconnect signal (loop current drop) detection and abortion by the *CMStopChannel()*. The channel must be in the connected state, , any attempt to call this function on a channel whose state is idle (or any transition state such as call waiting or call progressing) will produce an error.

See also

[CMOpenChannel\(\)](#), [CMStopChannel\(\)](#)

CMGetChassisInfo()

Syntax

```
BOOL CMGetChassisInfo(CHASSIS *lpChassis, CHASSIS2 *lpChassis2);
```

Parameters

lpChassis

[out] Pointer to a variable that receives the *CHASSIS* structure. Set to *NULL* if you are not interested in getting the device specific information.

lpChassis2

[out] Pointer to a variable that receives the *CHASSIS2* structure. Set to *NULL* if you are not interested in getting the device independent information.

Returns

If function succeeds, the return value is nonzero. If the function fails, the return value is zero.

Description

This function retrieves complete information on the hardware devices supported by the CM library that are installed on the current chassis. The retrieved information is stored in two structures: platform specific *CHASSIS* and platform independent *CHASSIS2*. The CM library must be initialized with the *CMInitLib()* prior to this function call. This function does not produce additional loading to hardware and may be freely called at any time before the library is uninitialized with the *CMUninitLib()*.

See also

[CMInitLib\(\)](#), [CHASSIS](#), [CHASSIS2](#), [CMUninitLib\(\)](#)

CMGetConfereeCount()

Syntax

```
int CMGetConfereeCount(CM_CONFHANDLE *pconf);
```

Parameters

pconf

[in] Pointer to a conference handle returned by *CMCreateConference()*.

Returns

The number of conferees.

Description

This function returns the number of conferees in the conference specified by *pconf* handle. The conference must be created with the *CMCreateConference()* before.

See also

[CMCreateConference\(\)](#)

CMGetDigits()

Syntax

```
int CMGetDigits(CM_HANDLE *pdevh, int Count, char *digits, int iInterruptFlags, int iUserTimeout);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

Count

[in] Number of DTMF digits to collect.

digits

[out] Pointer to a buffer that receives DTMF digits collected in the internal channel digit buffer. It is a responsibility of the application to allocate enough room for this buffer.

iInterruptFalgs

[in] Bitmask of termination falgs, may include:

CM_Channel_DetectFaxTone – terminate upon a fax tone detection.

iUserTimeout

[in] Maximum allowed timeout delay between two subsequent DTMF digits.

Returns

When the function terminates its return value indicates the termination reason. This can be any of the following bit-flags defined in the *CmLib.h*.

TM_MAXDTMF – *Count* of DTMF digits collected

TM_DIGIT – The function returned after a '#' digit detected

TM_USRSTOP – The function has been aborted by the application with the *CMStopChannel()*

TM_TONE – Fax tone has been detected. This might be returned only when the

CM_Channel_DetectFaxTone interruption flag is specified in the *iInterruptFlags* parameter

TM_MAXTIME – Maximum allowed delay between two subsequent DTMF digits specified by the *iUserTimeout* timed out

TM_LCOFF – Loop current OFF detected on an analog line

TM_PATTERN – Disconnect signal detected

TM_ERROR – An IO-device error occurred

Description

This function initiates the collection of DTMF digits into the internal channel buffer for the channel specified by *pdevh* handle. If some digits are present in the internal channel buffer at the moment of the *CMGetDigits()* call then the function collects *Count* less the number of digits stored initially. To have the *CMGetDigits()* to collect exactly *Count* digits flash the internal channel digit buffer with the *CMClearDigitBuffer()* before.

See also

[CMOpenChannel\(\)](#), [CMClearDigitBuffer\(\)](#), [CMStopChannel\(\)](#), [CMPlayDTMFTones\(\)](#)

CMGetDSPResCount()

Syntax

```
int CMGetDSPResCount(void);
```

Parameters

None

Returns

The total number of free conferencing resources.

Description

This function returns the total number of free conferencing resources available for all hardware installed on the chassis. It is zero if there is no device having conferencing resources (see *BOARDDESC* structure description) or all available resources are seized by existing active conferences.

See also

[BOARDDESC](#)

CMGetVersion()

Syntax

```
const char* CMGetVersion(void);
```

Parameters

None

Returns

Pointer to a null-terminated string that specifies the library version.

Description

This function returns the current CM library version for parsing the programming issues.

See also

Nothing

CMInitLib()

Syntax

```
int CMInitLib(LPSTR pRegKeyName);
```

Parameters

pRegKeyName

[in] Pointer to a null-terminated string that specifies the registry key where the library initialization parameters reside. All parameters must be located in two subkeys named *'Hardware'* and *'Tones'*. If *pRegKeyName* is *NULL* or points to the empty string the library uses default values for all parameters. If *pRegKeyName* is not *NULL* and some of the parameters listed on the table are not present in the registry key the default values for those parameters will be used. For description of those parameters see the table below. Since not every parameter is meaningful for all hardware the applicable platforms are listed in the last column and when need be the affected connectivity type is shown in the parentheses.

'Hardware' subkey parameters

Name	Meaning	Default value	Used by platforms
Ca_cnosig	Call progress duration of continuous no signal time-out delay. If exceeded a <i>no ringback</i>	40	Dialogic(analog)

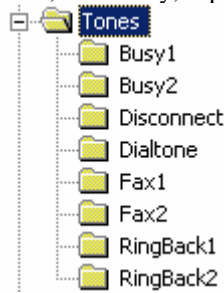
	is returned. Units: seconds		
Ca_maxintering	Call progress maximum interring delay before connection. Units: seconds	8	Dialogic(analog)
Ca_dtn_deboff	Call progress dial tone debounce, maximum gap allowed otherwise continuous dial tone before it considered invalid. Units: ms	100	Dialogic(analog)
Ca_dtn_npres	Call progress maximum time to wait for dial tone. Units: ms	3000	Dialogic(analog)
Ca_dtn_pres	Call progress length of time that a dial tone must be continuously present. Units: ms	1000	Dialogic(analog)
Ca_pamd_failtime	Call progress maximum time to wait for Positive Answering Machine Detection (PAMD) or Positive Voice Detection (PVD) after a cadence break. This parameter is used when the <i>UsePamd</i> is nonzero. Units: ms	4000	Dialogic
Ca_pamd_minring	Call progress minimum allowable ring duration for PVD/PAMD. This parameter is used when the <i>UsePamd</i> is nonzero. Units: ms	1900	Dialogic
Ca_pamd_spdval	(I) <i>Dialogic</i> : Call progress PVD/PAMD speed value. Possible values: 0 – quick decision 1 – full evaluation of response 2 – accurate evaluation of response This parameter is used when the <i>UsePamd</i> is nonzero. (II) <i>Brooktrout</i> : Call protocol selector during the call progress analysis. Possible values: 0 – <i>voice</i> protocol, invokes the call progress analysis along with the raw Hi/Lo data reporting; 1 – <i>voice no raw</i> protocol, invokes the call progress analysis without including the raw data; 2 – <i>fax</i> protocol, invokes the call progress analysis along with the raw Hi/Lo data reporting;	1	Dialogic, Brooktrout
Ca_stdely	Delay after dialing before starting call analysis. Units: ms	250	Dialogic(analog)
CallInfoAddressInfo	Custom outbound call parameter, indicates if address is partial (overlap mode) or complete (end of dialing). Possible values are: 1- enblock 2 - overlapped This parameter is invoked when the <i>UseMakeCallParameters</i> is set to nonzero.	1	Dialogic(E1/T1)
CallInfoCategory	Custom outbound call parameter, specifies the category of the call.	1	Dialogic(E1/T1)

	<p>Possible values are:</p> <ul style="list-style-type: none"> 1 – subscriber without priority 2 – subscriber with priority 3 – maintenance equipment 4 – coinbox or subscriber with charge metering 5 – operator 6 – data transmission 7 – C.P.T.P. 8 – special line 9 – mobile users 10 – virtual private network line <p>This parameter is invoked when the <i>UseMakeCallParameters</i> is set to nonzero.</p>		
ChanInfoMediumSel	<p>Custom outbound call parameter, specifies if the time-slot is preferred or exclusive. Possible values are:</p> <ul style="list-style-type: none"> 1 – preferred 2 – exclusive <p>This parameter is invoked when the <i>UseMakeCallParameters</i> is set to nonzero.</p>	1	Dialogic(E1/T1)
DestinationAddressPlan	<p>Custom outbound call parameter, specifies the Numbering plan for the address. Possible values are:</p> <ul style="list-style-type: none"> 1 – Unknown 2 – ISDN 3 – Telephony E.164 4 – Private <p>This parameter is invoked when the <i>UseMakeCallParameters</i> is set to nonzero.</p>	1	Dialogic(E1/T1)
DestinationAddressType	<p>Custom outbound call parameter, specifies the address type. Possible values are:</p> <ul style="list-style-type: none"> 1 – Transparent 2 – National 3 – International 4 – Local 5 – IP address <p>This parameter is invoked when the <i>UseMakeCallParameters</i> is set to nonzero.</p>	1	Dialogic(E1/T1)
DTMFDeb	<p>DTMF debounce time – max length of time in which a DTMF can be absent and then come back again and still be considered the same DTMF tone. Units: ms.</p>	0	Dialogic
DTMFTalk	<p>Minimum time for a DTMF tone must be present during audio playback to be considered valid. Units: ms.</p>	80	Dialogic, Eicon Diva Server, Pika N.B.: same parameter for Brooktrout is configured in <i>btcall.cfg</i>
FlashTime	<p>Specifies the hook flash duration for analog boards. Valid range is from 10 to 1000. Units: ms</p>	500	Dialogic(analog), Pika(analog) N.B.: same parameter for Brooktrout (analog) is configured in <i>callctrl.cfg</i> ; for Eicon(analog) in the driver settings

IntrusionLevel	Intrusion protection level for ISDN calls. Possible values are: 1 – level 1 2 – level 2 3 – level 3 This parameter is used when the local <i>bUseIntrusion</i> parameter of <i>CMMakeCall()</i> is set to <i>TRUE</i> .	3	Dialogic(E1/T1)
MaxSilenceToFinishRecord	Defines the maximum silence allowed while audio recording. If exceeded the recording is finished. Units: seconds.	5	Dialogic, Brooktrout, Eicon Diva Server
MessageLength	Defines the maximum message length allowed for audio recording. If exceeded the recording is finished. Units: seconds.	1000	All
NumRingsAnswer	Defines the number of rings to wait before reporting a new incoming call	2	Dialogic(analog), Pika(analog) N.B.: same parameter for Brooktrout (analog) is configured in <i>callctrl.cfg</i> ; for Eicon(analog) in the driver settings
Rings	Call progress maximum number of rings during which the call must be established, or a <i>no answer</i> will be returned.	8	All
SpeechDetectionThreshold	Sets the speech detection threshold to a gain of level dBm0. Increasing the level decreases the sensitivity of the detector to noise and other non-speech signals and reduces the ability to detect low level speech signals. Allowed integer range is -60 (min) to 0 (max).	-25	Pika
UseDialtone	If set to nonzero a dialtone is expected at the outbound call establishment and if it is not detected a <i>no dialtone</i> is returned.	1	Dialogic(analog), Brooktrout(analog)
UseDigitalPerfectCallAnalysis	Set to nonzero to invoke the Perfect Call Analysis on digital lines, otherwise the Basic Call Analysis will be invoked.	0	Dialogic(E1/T1)
UseMakeCallParameters	Set to nonzero to use custom outbound call parameters: <i>DestinationAddressType</i> , <i>DestinationAddressPlan</i> , <i>ChanInfoMediumSel</i> , <i>CallInfoCategory</i> and <i>CallInfoAddressInfo</i> ; otherwise the default values should be used for calls	0	Dialogic(E1/T1)
UsePamd	If set to nonzero the Positive Answering Machine Detection (PAMD) will be used during the call progress	0	Dialogic
Volume	This parameter adjusts the hardware volume gain level. Allowed values must be integer in range of -10 (min) to +5 (max)	0	All

'Tones' subkey parameters

The 'Tones' subkey is in fact a table of tone detector tones used by all hardware platforms at any time of a call life from the call establishment until the call termination. Currently this table consists of 8 entries, each entry, if present, must be located in the separate subkey as shown below.



Meaning of each tone entry corresponds to its name. E.g., 'Busy1' designates the main Busy signal, 'Busy2' designates the alternate busy signal and 'Disconnect' designates an analog Disconnect that is detected when the hook goes to the on-hook state. Each entry has same set of tone description parameters. The following tables list these parameters names and description and their default values for each tone entry.

Name	Description
freq1	Frequency of the fist subtone, Hz
freq1dev	Frequency deviation for the first subtone, Hz
freq2	Frequency of the second subtone, Hz
freq2dev	Frequency deviation for the second subtone, Hz
offtime	Tone off duration, ms
offtimeDEV	Tone off time deviation, ms
ontime	Tone on duration, ms
ontimeDEV	Tone on time deviation, ms
repetition	The number of times the signal pattern must repeat before being recognized as valid

Name	Default values							
	Busy1	Busy2	Disconnect	Dialtone	Fax1	Fax2	RingBack1	RingBack2
freq1, Hz	500	500	500	400	2150	1100	450	450
freq1dev, Hz	200	200	200	125	150	50	150	150
freq2, Hz	0	500	500	400	0	0	0	450
freq2dev, Hz	0	200	200	125	0	0	0	150
offtime, ms	550	550	250	0	0	0	5800	5800
offtimeDEV, ms	400	400	100	0	0	0	4150	4150
ontime, ms	550	550	250	0	250	0	1300	1300
ontimeDEV, ms	400	400	100	0	-250	0	1050	1050
repetition	4	4	4	0	0	0	0	0

Returns

This function returns a bitmask that specifies the underlying hardware drivers/libraries the function was able to successfully load and initialize. This can be any combination of the following values defined in the *CmLib.h*.

- CM_DIALOGIC_SR – Dialogic driver
- CM_DIALOGIC_DX – Dialogic Voice library
- CM_DIALOGIC_CC – Dialogic ISDN library
- CM_DIALOGIC_GC – Dialogic Global Call library
- CM_DIALOGIC_DTI – Dialogic dti-boards library
- CM_DIALOGIC_MSI – Dialogic msi-boards library
- CM_DIALOGIC_DCB – Dialogic dcb-boards library
- CM_DIALOGIC_FX – Dialogic Fax library
- CM_GAMMALINK_GDK – GammaLink driver
- CM_BROOKTROUT_BFV – Brooktrout Boston driver
- CM_EICON_DS – Eicon Diva Server driver
- CM_PIKA_MC – Pika Monte Carlo driver

Description

This function dynamically loads and initializes supported hardware drivers/libraries that are installed on the current chassis. The function is blocking and may take a considerable amount of time. If the function is called twice it does nothing but returns the valid bitmask discovered at the first function call. This function must be called first before any other *CM API* function call. The resources allocated by the *CMInitLib()* are released with the *CMUninitLib()* call which unloads and stops all the drivers.

See also

[CMUninitLib\(\)](#)

CMMakeCall()

Syntax

```
int CMMakeCall(CM_HANDLE *pdevh, LPSTR dialstr, BOOL bUseToneDialing, BOOL bUseIntrusion);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

dialstr

[in] Pointer to a null-terminated string that specifies the destination number to dial. This string may contain any characters but only digits '0'..'9' take effect.

bUseToneDialing

[in] Specifies the mode of dialing. If *TRUE* the DTMF tone dialing mode is used, otherwise the pulse dialing is used. This parameter is not applicable to E1/T1 boards and affects dialing on analog boards except Eicon Diva Server boards whose dialing mode is configured in the driver settings.

bUseIntrusion

[in] Set to *TRUE* to enable intrusion on E1/T1 lines. The intrusion protection level is specified by the *IntrusionLevel* parameter value set with the *CMInitLib()* at the library initialization. **N.B.:** Currently intrusion is realized for Dialogic hardware only.

Returns

When the function terminates its return value indicates the result of call progress. This can be any of the following values defined in the *CmLib.h*.

CR_BUSY – Line is busy

CR_NOANS – No answer detected

CR_NORB – No ringback detected

CR_CNCT – Call connected due to Positive Voice Detection (PVD)

CR_CEPT – Operator intercepted the call

CR_STOPD – Call analysis stopped by the application (Dialogic only, see CR_ERROR)

CR_NODIALTONE – No dialtone detected

CR_FAXTONE – Fax tone detected

CR_PAMD – Call connected due to Positive Answering Machine Detection (PAMD), Dialogic only. **N.B.:**

This result may be returned when the *UsePamd* initialization parameter of *CMInitLib()* is nonzero.

CR_ERROR – Call analysis error or abortion by the application

Description

This function is used to establish an outgoing call on a channel previously opened by the *CMOpenChannel()*. The channel state must be idle that is it must not be connected and no *CMWaitCall()* must be running on the channel at the same moment. An attempt to call the *CMMakeCall()* on a channel which is not in the idle state will produce the *CR_ERROR*. The function is blocking and takes a considerable amount of time. The return value indicates the call progress result. The behavior of this function is affected by a set of the library initialization call progress parameters discovered at the *CMInitLib()* call (see *CMInit()* 'Hardware' parameters description for complete information). E.g.:

- To invoke the Call Progress Analysis on E1/T1 line of Dialogic set the *UseDigitalPerfectCallAnalysis* to nonzero in the 'Hardware' registry subkey before you call *CMInitLib()*. Other platforms always use Call Progress Analysis for digital channels by default.
- To enable the answering machine detection (currently Dialogic only), set to nonzero the *UsePamd* parameter in the registry.
- To use custom ISDN protocol parameters on E1/T1 line (Dialogic) set the *UseMakeCallParameters* to nonzero. For platforms other than Dialogic custom ISDN protocol parameters are configured in the board driver settings.

N.B.: For E1/T1 digital lines: if the time slot corresponding to the channel the *CMMakeCall()* is called on is BLOCKED (as reported by the network) then this function returns *CR_NODIALTONE*.

See also

[CMOpenChannel\(\)](#), [CMInitLib\(\)](#), [CMWaitCall\(\)](#)

CMOpenChannel()

Syntax

```
typedef struct _CM_HANDLE CM_HANDLE;

CM_HANDLE* CMOpenChannel(USHORT board, USHORT channel);
```

Parameters

board

[in] Board index of a particular board device. It takes values in range of 0 to *BoardsCount* as defined by the *CHASSIS2* structure.

Channel

[in] Channel index that specifies a particular channel for the board device given by the *board* index. It takes values in range of 0 to *ChannelsCount* as defined in the *lpBoards[board]* member of the *CHASSIS2* structure.

Returns

If function succeeds, the return value is a pointer to the valid channel handle. If the function fails, the return value is *NULL*. The handle returned by *CMOpenChannel()* is *opaque*, that is the application is not supposed to know about its contents. All subsequent library calls for this channel must pass the handle as an argument.

Description

This function opens a channel on a board of any supported kind that is installed on the pc and returns a handle to be used in subsequent calls to routines of CM API. If the channel open operation fails due to some reason then *NULL* is returned. The CM library must be initialized with the *CMInitLib()* prior to this function call.

See also

[CMCloseChannel\(\)](#), [CMGetChassisInfo\(\)](#), [CMInitLib\(\)](#)

CMPlayAudioFile()

Syntax

```
int CMPlayAudioFile(CM_HANDLE *pdevh, LPSTR pFile, int iFormat, int iInterruptFlags);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

pFile

[in] Pointer to a null-terminated string that specifies the audio file to be played.

iFormat

[in] Audio format of the file given by the *pFile*. This parameter takes sense only when the audio file contains raw voice data (headerless file), that is when the extension of *pFile* is different from '.wav' or '.mp3'. This can be any of the following bit-flags defined in the *CmLib.h*:

Pika proprietary ADPCM formats:

CM_AUDIO_3bit_4kHz_PIKA_ADPCM
CM_AUDIO_3bit_6kHz_PIKA_ADPCM
CM_AUDIO_3bit_8kHz_PIKA_ADPCM
CM_AUDIO_3bit_11kHz_PIKA_ADPCM
CM_AUDIO_4bit_4kHz_PIKA_ADPCM
CM_AUDIO_4bit_6kHz_PIKA_ADPCM
CM_AUDIO_4bit_8kHz_PIKA_ADPCM
CM_AUDIO_4bit_11kHz_PIKA_ADPCM

Oki-Dialogic ADPCM formats:

CM_AUDIO_4bit_6kHz_OKI_ADPCM
CM_AUDIO_4bit_8kHz_OKI_ADPCM
CM_AUDIO_4bit_11kHz_OKI_ADPCM

A-law/u-law PCM audio:

CM_AUDIO_8bit_4kHz_aLaw_PCM
CM_AUDIO_8bit_6kHz_aLaw_PCM
CM_AUDIO_8bit_8kHz_aLaw_PCM
CM_AUDIO_8bit_11kHz_aLaw_PCM
CM_AUDIO_8bit_4kHz_uLaw_PCM
CM_AUDIO_8bit_6kHz_uLaw_PCM
CM_AUDIO_8bit_8kHz_uLaw_PCM
CM_AUDIO_8bit_11kHz_uLaw_PCM

Linear audio formats:

CM_AUDIO_8bit_4kHz_PCM
CM_AUDIO_8bit_6kHz_PCM
CM_AUDIO_8bit_8kHz_PCM
CM_AUDIO_8bit_11kHz_PCM
CM_AUDIO_16bit_4kHz_PCM
CM_AUDIO_16bit_6kHz_PCM
CM_AUDIO_16bit_8kHz_PCM
CM_AUDIO_16bit_11kHz_PCM

iInterruptFlags

[in] Bitmask of termination flags, may include any combination of the following:

CM_Channel_DetectDTMF – terminate upon a DTMF tone detection.

CM_Channel_DetectFaxTone – terminate upon a fax tone detection.

Returns

When the function terminates its return value indicates the termination reason. This can be any of the following bit-flags defined in the *CmLib.h*.

TM_NORMTERM – The function terminated normally after playback is complete

TM_MAXDTMF – The function returned after a DTMF digit detected. This might be returned only when the *CM_Channel_DTMF* interruption flag is specified in the *iInterruptFlags* parameter

TM_USRSTOP – The function has been aborted by the application with the *CMStopChannel()*

TM_TONE – Fax tone has been detected. This might be returned only when the

CM_Channel_DetectFaxTone interruption flag is specified in the *iInterruptFlags* parameter

TM_LCOFF – Loop current OFF detected on an analog line

TM_PATTERN – Disconnect signal detected

TM_ERROR – An IO-device error occurred

Description

This function plays pre-recorded voice data stored in the file given by the *pFile* on a channel specified by the *pdevh* handle. The channel must be in the connected state, any attempt to call this function on a channel

whose state is idle (or any transition state such as call waiting or call progressing) will produce an error. Termination conditions for *CMPlayAudioFile()* other than normal due to the end of file are set using the *iInterruptFalgs* bitmask of termination flags.

This is a convenience function that is capable of playing any WAVE and RAW audio recorded in any format listed above (see *iFormat* parameter description) or MPEG Layer 3 audio (MP3-files) on any channel. If needed this function internally performs audio data conversion to one of the hardware supported formats. The current default value of audio format for all hardware platforms is *CM_AUDIO_8bit_8kHz_uLaw_PCM*, which means this format will be used as the target for internal conversions if the source file cannot be played on the local hardware channel. The initial audio file stays unchanged.

See also

[CMOpenChannel\(\)](#), [CMPlayIndexedAudioFile\(\)](#), [CMRecordAudioFile\(\)](#), [CMStopChannel\(\)](#)

CMPlayDTMFTones()

Syntax

```
int CMPlayDTMFTones(CM_HANDLE *pdevh, char *digits, int iToneDuration, int iInterdigitTimeout);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

digits

[in] Pointer to a null-terminated string that specifies the DTMF sequence to be played on the channel. Valid DTMF tones (case insensitive) are '0' to '9', 'A' to 'D', '*' and '#', all other characters will be ignored.

iToneDuration

[in] Specifies the DTMF tone duration in *msec*.

N.B.: This parameter is not used for Pika platform, the hardware default value *80 msec* is used instead.

iInterdigitTimeout

[in] Specifies the minimum silence duration following a DTMF tone in *msec*.

¹**N.B.:** This parameter is not used for Pika platform, the hardware default value *80 msec* is used instead.

²**N.B.:** For Brooktrout platform this value is configured in the *bt_cparm.cfg* hardware configuration file (*tone_inter_time* parameter).

Returns

If the function succeeds the return value is 0, otherwise the return value is (-1).

Description

This function plays the given sequence of DTMF tones (digits) on the channel specified by the *pdevh* handle. The channel must be in the connected state, any attempt to call this function on a channel whose state is idle (or any transition state such as call waiting or call progressing) will produce an error.

See also

[CMOpenChannel\(\)](#), [CMGetDigits\(\)](#)

CMPlayIndexedAudioFile()

Syntax

```
int CMPlayIndexedAudioFile(CM_HANDLE *pdevh, LPSTR pFile, int iIndex, int iFormat, int iInterruptFlags);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

pFile

[in] Pointer to a null-terminated string that specifies the indexed (VAP) audio file to be played.

iIndex

[in] Specifies the index of a VAP-file segment to be played.

iFormat

[in] Audio format of the file given by the *pFile*. This parameter takes sense only when the audio file contains raw voice data (headerless file), that is when the extension of *pFile* is different from '.wav' or '.mp3'. This can be any of the following bit-flags defined in the *CmLib.h*:

Pika proprietary ADPCM formats:

CM_AUDIO_3bit_4kHz_PIKA_ADPCM
CM_AUDIO_3bit_6kHz_PIKA_ADPCM
CM_AUDIO_3bit_8kHz_PIKA_ADPCM
CM_AUDIO_3bit_11kHz_PIKA_ADPCM
CM_AUDIO_4bit_4kHz_PIKA_ADPCM
CM_AUDIO_4bit_6kHz_PIKA_ADPCM
CM_AUDIO_4bit_8kHz_PIKA_ADPCM
CM_AUDIO_4bit_11kHz_PIKA_ADPCM

Oki-Diallogic ADPCM formats:

CM_AUDIO_4bit_6kHz_OKI_ADPCM
CM_AUDIO_4bit_8kHz_OKI_ADPCM
CM_AUDIO_4bit_11kHz_OKI_ADPCM

A-law/u-law PCM audio:

CM_AUDIO_8bit_4kHz_aLaw_PCM
CM_AUDIO_8bit_6kHz_aLaw_PCM
CM_AUDIO_8bit_8kHz_aLaw_PCM
CM_AUDIO_8bit_11kHz_aLaw_PCM
CM_AUDIO_8bit_4kHz_uLaw_PCM
CM_AUDIO_8bit_6kHz_uLaw_PCM
CM_AUDIO_8bit_8kHz_uLaw_PCM
CM_AUDIO_8bit_11kHz_uLaw_PCM

Linear audio formats:

CM_AUDIO_8bit_4kHz_PCM
CM_AUDIO_8bit_6kHz_PCM
CM_AUDIO_8bit_8kHz_PCM
CM_AUDIO_8bit_11kHz_PCM
CM_AUDIO_16bit_4kHz_PCM
CM_AUDIO_16bit_6kHz_PCM
CM_AUDIO_16bit_8kHz_PCM
CM_AUDIO_16bit_11kHz_PCM

iInterruptFalgs

[in] Bitmask of termination falgs, may include any combination of the following:

CM_Channel_DetectDTMF – terminate upon a DTMF tone detection.

CM_Channel_DetectFaxTone – terminate upon a fax tone detection.

Returns

When the function terminates its return value indicates the termination reason. This can be any of the following bit-flags defined in the *CmLib.h*.

TM_NORMTERM – The function terminated normally after playback is complete

TM_MAXDTMF – The function returned after a DTMF digit detected. This might be returned only when the *CM_Channel_DTMF* interruption flag is specified in the *iInterruptFlags* parameter

TM_USRSTOP – The function has been aborted by the application with the *CMStopChannel()*

TM_TONE – Fax tone has been detected. This might be returned only when the

CM_Channel_DetectFaxTone interruption flag is specified in the *iInterruptFlags* parameter

TM_LCOFF – Loop current OFF detected on an analog line

TM_PATTERN – Disconnect signal detected

TM_ERROR – An IO-device error occurred

Description

This function is quite similar to the *CMPlayAudioFile()* but its purpose is to play a part of the indexed VAP-file as given by the VAP index *iIndex*. A VAP file is nothing more than a series of raw audio data files. In its initial segment, a VAP file contains information about the location of beginnings (starting points) of individual files (speech phrases), as well as their text descriptions, in order to facilitate differentiation between individual phrases. The VAP-files specification should be found elsewhere. VAP files are widely used for creating voice libraries, e.g. libraries for spelling calendar dates, numbers, currency values, etc.

See also

[CMOpenChannel\(\)](#), [CMPlayAudioFile\(\)](#), [CMRecordAudioFile\(\)](#), [CMStopChannel\(\)](#)

CMReceiveFaxFile()

Syntax

```
int CMReceiveFaxFile(CM_HANDLE *pdevh, LPSTR pFile, FaxParams *Params);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

pFile

[in] Pointer to a null-terminated string that will be assigned as the file name to the fax received. All received fax data are stored by the library in the TIFF/F Group 3 image format.

Params

[in/out] Pointer to the FaxParams structure. On input this structure specifies incoming fax parameters such as maximum baud rate, local Id and desired fax resolution. On output it receives detailed information on the fax received. If set to *NULL* then hardware default settings will be invoked for fax receiving. (See *FaxParams* structure description for details)

Returns

If the function succeeds the return value is 0, the **Params* structure contains information on the factual baud rate, fax resolution, remote fax id and the number of pages successfully transmitted during the session. If the function fails the return value is (-1), the **Params* structure contains hardware error message string.

Description

This function receives fax data from a channel specified by the *pdevh* handle and stores data as a TIFF/F Group 3 image file. The channel must be in the connected state, any attempt to call this function on a channel whose state is idle (or any transition state such as call waiting or call progressing) will produce an error.

See also

[CMOpenChannel\(\)](#), [CMSendFaxFiles\(\)](#), [FaxParams](#)

CMRecordAudioFile()

Syntax

```
int CMRecordAudioFile(CM_HANDLE *pdevh, LPSTR pFile, int iFormat, int iInterruptFlags, BOOL bBeepTone);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

pFile

[in] Pointer to a null-terminated string that specifies the file name used for recording. If a file with same name already exists its content will be deleted. The supplied file extension specifies whether the resulting file will contain WAVE audio (if extension set to '.wav') or raw voice data (if extension differs from '.wav').

iFormat

[in] Specifies the audio data format in the resulting file given by the *pFile*. This can be any of the following bit-flags defined in the *CmLib.h*:

Pika proprietary ADPCM formats:

CM_AUDIO_3bit_4kHz_PIKA_ADPCM
CM_AUDIO_3bit_6kHz_PIKA_ADPCM
CM_AUDIO_3bit_8kHz_PIKA_ADPCM
CM_AUDIO_3bit_11kHz_PIKA_ADPCM
CM_AUDIO_4bit_4kHz_PIKA_ADPCM
CM_AUDIO_4bit_6kHz_PIKA_ADPCM
CM_AUDIO_4bit_8kHz_PIKA_ADPCM
CM_AUDIO_4bit_11kHz_PIKA_ADPCM

Oki-Dialogic ADPCM formats:

CM_AUDIO_4bit_6kHz_OKI_ADPCM
CM_AUDIO_4bit_8kHz_OKI_ADPCM
CM_AUDIO_4bit_11kHz_OKI_ADPCM

A-law/u-law PCM audio:

CM_AUDIO_8bit_4kHz_aLaw_PCM
CM_AUDIO_8bit_6kHz_aLaw_PCM
CM_AUDIO_8bit_8kHz_aLaw_PCM
CM_AUDIO_8bit_11kHz_aLaw_PCM
CM_AUDIO_8bit_4kHz_uLaw_PCM
CM_AUDIO_8bit_6kHz_uLaw_PCM
CM_AUDIO_8bit_8kHz_uLaw_PCM
CM_AUDIO_8bit_11kHz_uLaw_PCM

Linear audio formats:

CM_AUDIO_8bit_4kHz_PCM
CM_AUDIO_8bit_6kHz_PCM
CM_AUDIO_8bit_8kHz_PCM
CM_AUDIO_8bit_11kHz_PCM
CM_AUDIO_16bit_4kHz_PCM
CM_AUDIO_16bit_6kHz_PCM
CM_AUDIO_16bit_8kHz_PCM
CM_AUDIO_16bit_11kHz_PCM

iInterruptFalgs

[in] Bitmask of termination falgs, may include any combination of the following:

CM_Channel_DetectDTMF – terminate upon a DTMF tone detection.

CM_Channel_DetectFaxTone – terminate upon a fax tone detection.

CM_Channel_DetectMaxSilence – terminate if the maximum silence timeout delay is exceeded

CM_Channel_DetectMaxTime – terminate if the maximum message length is exceeded

bBeepTone

Specifies if the “beep” tone should be played before recording. Set to *TRUE* to prompt the caller with a “beep”.

Returns

When the function terminates its return value indicates the termination reason. This can be any of the following bit-flags defined in the *CmLib.h*.

TM_MAXTIME – The maximum message length specified by the *MessageLength* library initialization parameter has been exceeded. (See *CMInitLib()* function description for details).

¹**N.B.:** For Dialogic this result might be returned only when the *CM_Channel_DetectMaxTime* interruption flag is specified in the *iInterruptFlags* parameter.

²**N.B.:** For the other hardware platforms this result can be returned regardless the *CM_Channel_DetectMaxTime* flag, that is the maximum message timeout always works for these platforms.

TM_MAXDTMF – The function returned after a DTMF digit detected. This might be returned only when the *CM_Channel_DTMF* interruption flag is specified in the *iInterruptFlags* parameter

TM_MAXSIL - The maximum silence allowed while audio recording has been exceeded. This timeout delay is specified by the *MaxSilenceToFinishRecord* library initialization parameter. (See *CMInitLib()* function description for details).

¹**N.B.:** For Dialogic this result might be returned only when the *CM_Channel_DetectMaxSilence* interruption flag is specified in the *iInterruptFlags* parameter.

²**N.B.:** For Pika this result is never returned.

³**N.B.:** For Brooktrout and Eicon Diva Server this result can be returned regardless the *CM_Channel_DetectMaxSilence* flag, that is the maximum silence timeout always works for these platforms.

TM_USRSTOP – The function has been aborted by the application with the *CMStopChannel()*

TM_TONE – Fax tone has been detected. This might be returned only when the *CM_Channel_DetectFaxTone* interruption flag is specified in the *iInterruptFlags* parameter

TM_LCOFF – Loop current OFF detected on an analog line

TM_PATTERN – Disconnect signal detected

TM_ERROR – An IO-device error occurred

Description

This function records audio data on a channel specified by the *pdevh* handle and stores them in the file. Depending on the file name extension specified by the *pFile* the returned file contains whether headerless raw data or WAVE standard audio. The channel must be in the connected state, any attempt to call this function on a channel whose state is idle (or any transition state such as call waiting or call progressing) will produce an error. Termination conditions for *CMRecordAudioFile()* are set using the *iInterruptFalgs* bitmask of termination flags.

This is a convenience function that is capable of recording audio in any format listed above (see *iFormat* parameter description) on any channel. If needed this function internally performs audio data conversion from one of the hardware supported formats. The current default value of audio format for all hardware platforms is *CM_AUDIO_8bit_8kHz_uLaw_PCM*, which means this format will be used for real recording if the requested format cannot be used on the local hardware channel. The resulting file will always have data encoded in the *iFormat*.

See also

[CMOpenChannel\(\)](#), [CMPlayAudioFile\(\)](#), [CMPlayIndexedAudioFile\(\)](#), [CMInitLib\(\)](#), [CMStopChannel\(\)](#)

CMRemoveFromConference()

Syntax

```
BOOL CMRemoveFromConference(CM_CONFHANDLE *pconf, CM_HANDLE *pdevh);
```

Parameters

pconf

[in] Pointer to a conference handle returned by *CMCreateConference()*.

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*. This handle identifies the conferee to be removed from the conference.

Returns

If this function succeeds the return value is *TRUE*, otherwise the return value is *FALSE*.

Description

This function removes a conferee specified by the *pdevh* channel handle from an active conference specified by the *pconf* handle. The channel state stays unchanged after this function terminates. Upon a successful termination the caller identified by the *pdevh* handle may be freely added to the same or another conference with the *CMAddToConference()* function call.

See also

[CMCreateConference\(\)](#), [CMAddToConference\(\)](#), [CMOpenChannel\(\)](#)

CMSendFaxFiles()

Syntax

```
int CMSendFaxFiles(CM_HANDLE *pdevh, int FilesCount, LPSTR *ppFiles, FaxParams *Params,
                  BOOL bFaxCheck);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

FilesCount

[in] The number of entries in the buffer specified by the *ppFiles*. This is the fax files count to be sent.

ppFiles

[in] Pointer to a buffer containing an array of pointers to null-terminated strings specifying the fax files to be sent. CM library allows faxing of ASCII-files and TIFF-files complying with the TIFF/F Group 3 specification. Any files whose extensions are different from '.txt', '.tif' and '.tiff' will be cast away.

¹**N.B.:** Sending ASCII text-files is supported by Dialogic and Brooktrout platforms only.

²**N.B.:** Dialogic fax reference requires that if present, the RowsPerStrip TIFF tag must equal the ImageLength. That is the image is considered to be one large strip and multiple strips per image are not supported.

Params

[in/out] Pointer to the FaxParams structure. On input this structure specifies outgoing fax parameters such as maximum baud rate, local Id, fax header and desired fax resolution. On output it receives detailed information on the fax transmission. If set to *NULL* then hardware default settings will be invoked for fax sending. (See *FaxParams* structure description for details)

bFaxCheck

Reserved for internal use, must be set to 0.

Returns

If the function succeeds the return value is 0, the **Params* structure contains information on the factual baud rate, fax resolution, remote fax id and the number of pages successfully transmitted during the session. If the function fails the return value is (-1), the **Params* structure contains hardware error message string.

Description

This function transmits multiple fax files from the *ppFiles* list in a single fax session on a channel specified by the *pdevh* handle. Currently this function supports sending ASCII text-files and images complying with the TIFF/F Group 3 specification. The channel must be in the connected state, any attempt to call this function on a channel whose state is idle (or any transition state such as call waiting or call progressing) will produce an error.

See also

[CMOpenChannel\(\)](#), [CMReceiveFaxFile\(\)](#), [FaxParams](#)

CMSetConfereeStatus()

Syntax

```
BOOL CMSetConfereeStatus(CM_CONFHANDLE *pconf, CM_HANDLE *pdevh, int *piAttr);
```

Parameters

pconf

[in] Pointer to a conference handle returned by *CMCreateConference()*.

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*. This handle identifies the conferee that is a party of the conference specified by the *pconf* handle.

piAttrl

[in] Pointer to an integer that specifies the conferee desired status. This can be any of the following values defined in the *CmLib.h*:

CM_Conferee_Monitor – a single duplex conferee, this kind of a conference party can only listen to the other parties.

CM_Conferee_Active – a full duplex conferee (default), this kind of a conference party can listen to and speak with the other parties.

CM_Conferee_Coach – the *Coach*, this is a so-called supervision link party who can listen to all the parties but can be heard by the *Pupil* only. There can be only one supervision link and *Coach* in a conference. An attempt to add one more *Coach* will produce an error. To make another conferee the *Coach*, first assign the existing *Coach* a new attribute (single or full duplex) with the *CMSetConfereeStatus()* function, then set the *Coach* attribute to a new conferee with another *CMSetConfereeStatus()* call. That will link the new *Coach* to the existing *Pupil*.

CM_Conferee_Pupil – the *Pupil*, this is another supervision link party who can listen to and can be heard by all the parties including the *Coach*. There can be only one supervision link and *Pupil* in a conference. An attempt to add one more *Pupil* will produce an error. To make another conferee the *Pupil*, first assign the existing *Pupil* a new attribute (single or full duplex) with the *CMSetConfereeStatus()* function, then set the *Pupil* attribute to a new conferee with another *CMSetConfereeStatus()* call. That will link the new *Pupil* to the existing *Coach*.

If **piAttr* is assigned a value different from the listed above the conferee will be created with the default attribute (*CM_Conferee_Active*).

Returns

If this function succeeds the return value is *TRUE*, otherwise the return value is *FALSE*.

Description

This function modifies the attribute (status) of a conferee identified by the *pdevh* channel handle. The conferee must be added to the conference specified by the *pconf* handle with either *CMCreateConference()* or *CMAddToConference()* prior to this function call. Otherwise the *CMSetConfereeStatus()* will return *FALSE* to indicate an error. The channel state stays unchanged after this function terminates.

See also

[CMCreateConference\(\)](#), [CMAddToConferenceConference\(\)](#), [CMOpenChannel\(\)](#)

CMSetOnHook()

Syntax

```
void CMSetOnHook(CM_HANDLE *pdevh);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

Returns

Nothing.

Description

This function disconnects the current call (for all type of channels) and switches the hookstate to on-hook for analog channels if the channel specified by the *pdevh* is connected. If the channel specified by the *pdevh* is already idle, the *CMSetOnHook()* has no effect. When the function returns the channel state is always idle. *CMSetOnHook()* is used for termination of active calls, after that the channel might be used for establishing a new call with the *CMMakeCall()* or with the *CMWaitCall()* functions. This function is internally called by the *CMCloseChannel()* at the channel closure. This function calls the *CMClearDigitBuffer()* before disconnecting an active call to flash the digits present in the channel digit buffer.

See also

[CMOpenChannel\(\)](#), [CMCloseChannel\(\)](#), [CMMakeCall\(\)](#), [CMWaitCall\(\)](#), [CMClearDigitBuffer\(\)](#)

CMStopChannel()

Syntax

```
void CMStopChannel(CM_HANDLE *pdevh);
```

Parameters

pdevh
[in] Pointer to a channel handle returned by *CMOpenChannel()*.

Returns

Nothing.

Description

This function forces termination of currently active IO-functions (such as audio streaming, fax transmission, etc.) on a channel leaving the channel state connected. If there's no IO-function pending on the channel specified by the *pdevh*, the *CMStopChannel()* has no effect. This function is not blocking and is supposed to be called asynchronously from a separate thread.

See also

[CMOpenChannel\(\)](#), [CMCloseChannel\(\)](#)

CMUninitLib()

Syntax

```
void CMUninitLib(void);
```

Parameters

None.

Returns

Nothing.

Description

This function releases all resources allocated with the *CMInitLib()*, stops the hardware drivers and unloads all their libraries. This function must be called before the application terminates. No *CM API* function is allowed to be called after *CMUninitLib()* returns.

See also

[CMInitLib\(\)](#)

CMWaitCall()

Syntax

```
int CMWaitCall(CM_HANDLE *pdevh, DWORD dwTimeout, char *ani, char *dnis);
```

Parameters

pdevh

[in] Pointer to a channel handle returned by *CMOpenChannel()*.

dwTimeout

[in] Specifies the maximum length of time in seconds to wait for a ring.

ani

[out] Pointer to a buffer that receives the calling party number (ANI for digital E1/T1 lines or *CallerId* for analog lines) after the channel gets connected. It is a responsibility of the application to allocate enough room for this buffer. **N.B.:** *CallerId* feature for analog lines is not currently supported for Pika analog boards.

dnis

[out] Pointer to a buffer that receives the called party number (DNIS) for digital E1/T1 lines only after the channel gets connected. It is a responsibility of the application to allocate enough room for this buffer.

Returns

The return value indicates the call waiting result:

- >0 – the channel state is connected,
- 0 – the function returned due to timeout,
- <0 – an error occurred during the function execution

Description

This function waits for an incoming call on a channel previously opened by the *CMOpenChannel()* for the duration of time given by the *dwTimeout*. The channel state must be idle that is it must not be connected and no *CMMakeCall()* must be running on the channel at the same moment. An attempt to call the *CMWaitCall()* on a channel which is not in the idle state will produce an error. The return value indicates the call waiting result. For digital E1/T1 lines this function accepts a call as soon as the *SETUP* message comes. For analog lines *CMWaitCall()* waits for a number of rings specified by the *NumRingsAnswer* library initialization parameter (see *CMInit()* 'Hardware' parameters description for complete information).

N.B.: For E1/T1 digital lines: if the time slot corresponding to the channel the *CMWaitCall()* is called on is BLOCKED (as reported by the network) then this function returns (-1).

See also

[CMMakeCall\(\)](#), [CMInitLib\(\)](#)

Data structure index

This chapter gives a description of the CM API data structures provided that they are listed alphabetically for ease of use. All structures of the CM library are defined in the *CmLib.h* header file.

BOARDDESC

Syntax

```
typedef struct _BOARDDESC {
    USHORT NetType;
    int ModuleNum;
    BOOL bVoiceSupported;
    BOOL bFaxSupported;
    BOOL bConfSupported;
    char SerialId[15];
    int ChannelsCount;
    int ExtsCount;
    int DSPCount;
} BOARDDESC, *LPBOARDDESC;
```

Members

NetType

Specifies the network connection type of the board. May be any of the following values defined in the *CmLib.h*:

- NT_LSI – Dialogic analogue - loop start
- NT_DTI – Dialogic dti board
- NT_BRI – Dialogic ISDN BRI
- NT_GCISDN – Dialogic ISDN PRI Springware board
- NT_GCR2 – Dialogic E1(R2) Springware board
- NT_GCDM3 – Dialogic DM3 series
- NT_DCB – Dialogic dcb module (no interface)
- NT_MSI – Dialogic msi board
- NT_GL – GammaLink fax board
- NT_BKT – Brooktrout TR-series
- NT_EDS – Eicon Diva Server board
- NT_PKLSINL – Pika analogue - loop start (InLine)
- NT_PKLSDT – Pika analogue - loop start (Daytona)
- NT_PKISDN – Pika E1/T1 board (PrimeNet)

ModuleNum

Specifies the device specific board index as assigned by the underlying hardware driver.

bVoiceSupported

Specifies whether the board has voice resources. If this member is *TRUE*, the board supports voice capabilities.

bFaxSupported

Specifies whether the board has voice resources. If this member is *TRUE*, the board supports fax capabilities.

bConfSupported

Specifies whether the board has conferencing resources. If this member is *TRUE*, the board supports conferencing.

SerialId

Pointer to a null-terminated string that contains the board serial Id as defined by the underlying hardware driver. Uniquely identifies each board.

ChannelsCount

The total number of channels on the board. CM library internally assigns unique IDs to all board's channels from 0 to *ChannelsCount-1*. If this member is 0 then the board has no channels (resource board).

ExtsCount

The total number of station channels (extensions) on the board. CM library internally assigns unique IDs to all board's station channels from 0 to *ExtsCount-1*. If this member is 0 then the board has no station channels.

DSPCount

Specifies the total number of DSP processors available on the board. **N.B.:** Applicable only to Dialogic and Pika conferencing boards.

Description

This structure contains information on a particular board installed on the chassis. Pointer to an array of these structures is a member of the *DEVDESC* and *CHASSIS2* structures.

See also

[DEVDESC](#), [CHASSIS2](#)

CHASSIS

Syntax

```
typedef struct _CHASSIS {
    LPDEVDESC lpIntelVoice;
    LPDEVDESC lpIntelDTi;
    LPDEVDESC lpIntelBRI;
    LPDEVDESC lpIntelMSI;
    LPDEVDESC lpIntelDCB;
    LPDEVDESC lpIntelCPI;
    LPDEVDESC lpBrktTR;
    LPDEVDESC lpEiconDS;
    LPDEVDESC lpPikaAnalog;
    LPDEVDESC lpPikaDigital;
} CHASSIS;
```

Members

lpIntelVoice

Pointer to a *DEVDESC* structure that contains information on the Dialogic voice boards installed on the chassis. If it is *NULL* then no Dialogic voice board found. **N.B.:** Dialogic driver emulates a set of 4-port voice logical devices for all Dialogic boards having the voice resources. E.g. a 12-port analog board is considered as four 4-port logical voice devices.

lpIntelDTi

Pointer to a *DEVDESC* structure that contains information on the Dialogic E1/T1 boards (digital interface boards) installed on the chassis. If it is *NULL* then no Dialogic E1/T1 board found.

lpIntelBRI

Pointer to a *DEVDESC* structure that contains information on the Dialogic BRI boards installed on the chassis. If it is *NULL* then no Dialogic BRI board found.

lpIntelMSI

Pointer to a *DEVDESC* structure that contains information on the Dialogic msi-boards installed on the chassis. If it is *NULL* then no Dialogic msi-board found.

lpIntelDCB

Pointer to a *DEVDESC* structure that contains information on the Dialogic dcb-boards installed on the chassis. If it is *NULL* then no Dialogic dcb-board found.

lpIntelCPI

Pointer to a *DEVDESC* structure that contains information on the GammaLink fax boards installed on the chassis. If it is *NULL* then no GammaLink fax board found.

lpBrktTR

Pointer to a *DEVDESC* structure that contains information on the Brooktrout tr-boards installed on the chassis. If it is *NULL* then no Brooktrout tr-board found.

lpEiconDS

Pointer to a *DEVDESC* structure that contains information on the Eicon Diva Server boards installed on the chassis. If it is *NULL* then no Eicon Diva Server board found.

lpPikaAnalog

Pointer to a *DEVDESC* structure that contains information on the Pika analog boards installed on the chassis. If it is *NULL* then no Pika analog board found.

lpPikaDigital

Pointer to a *DEVDESC* structure that contains information on the Pika digital boards installed on the chassis. If it is *NULL* then no Pika E1/T1 board found.

Description

This structure contains device specific information on all supported telephony boards currently installed on the chassis. This information is returned by the *CMGetChassisInfo()* function. This function also returns device independent information stored in the *CHASSIS2* structure.

See also

[DEVDESC](#), [CMGetChassisInfo\(\)](#), [CHASSIS2](#)

CHASSIS2

Syntax

```
typedef struct _CHASSIS2 {
    int LinesCount;
    int BoardsCount;
    LPBOARDDESC lpBoards;
} CHASSIS2;
```

Members

LinesCount

The total number of lines for all boards of supported kinds that are installed and running on the chassis.

BoardsCount

The total number of boards (logical devices) of supported kinds that are installed and running on the chassis. CM library internally assigns unique IDs to all boards from 0 to *BoardsCount*-1. If it is 0 then no supported board found and the *lpBoards* member is *NULL*.

lpBoards

Pointer to an array of *BOARDDESC* structures consisting of *BoardsCount* entries. Each entry contains complete information on the board with the corresponding ID starting from 0. If it is *NULL* then no supported board found.

Description

This structure contains device independent information on all supported telephony boards currently installed on the chassis. This information is returned by the *CMGetChassisInfo()* function. This function also returns device specific information stored in the *CHASSIS* structure.

See also

[CMGetChassisInfo\(\)](#), [CHASSIS](#), [BOARDDESC](#)

DEVDESC

Syntax

```
typedef struct _DEVDESC {
    int BoardsCount;
    char szDriverVer[40];
    char szCompiledVer[40];
    LPBOARDDESC lpBoardDesc;
} DEVDESC, *LPDEVDESC;
```

Members

BoardsCount

The number of boards (logical devices) of a particular kind (say, the number of Dialogic E1/T1 boards, or Pika analog Daytona boards) that are installed and running on the chassis. If it is 0 then no board of that kind is found and the *lpBoardDesc* member is *NULL*.

szDriverVer

Pointer to a null-terminated string that specifies the current version of the underlying hardware driver.

szCompiledVer

Pointer to a null-terminated string that specifies the hardware SDK version used to compile the library.

lpBoardDesc

Pointer to an array of *BOARDDESC* structures consisting of *BoardsCount* entries. Each entry contains complete information on a board of same kind. If it is *NULL* then no board of a particular is found.

Description

This structure contains information on all boards of same particular kind that are currently installed on the chassis. Pointers to arrays of these structures are members of the *CHASSIS* structure.

See also

[CHASSIS](#), [BOARDDESC](#)

FaxParams

Syntax

```
#define CM_MAX_ERR_LENGTH      130

typedef struct faxparams {
    char *szFaxHeader;
    char szLocalId[21];
    char szRemoteId[21];
    int BaudRate;
    int Resolution;
    int NumOfPagesSent;
    int NumOfBadPages;
    char ErrorMessage[CM_MAX_ERR_LENGTH];
    BOOL CombinePages;
    int RetryCNT;
    int RTN;
    int RTP;
} FaxParams;
```

Members

szFaxHeader

[in] Pointer to a null-terminated string that specifies the outbound fax header. If set to *NULL* then the default fax header will be used which may differ for various hardware platforms. If it points to an empty string then no fax header will be printed.

szLocalId

[in] Pointer to a null-terminated string that specifies the local fax Id that will be transmitted to the remote fax device. Its maximum allowed length is 21 character.

szRemoteId

[out] Pointer to a null-terminated string that receives the remote fax device Id. Its maximum allowed length is 21 character.

BaudRate

[in/out] On input specifies the maximum allowable baud rate for the fax transmission. On output specifies the factual fax transmission baud rate. This can be one of the following bit-flags defined in the *CmLib.h*:

```
CM_FAX_RATE_2400
CM_FAX_RATE_4800
CM_FAX_RATE_7200
CM_FAX_RATE_9600
CM_FAX_RATE_12000
CM_FAX_RATE_14400
CM_FAX_RATE_16800
CM_FAX_RATE_21600
CM_FAX_RATE_24000
CM_FAX_RATE_26400
CM_FAX_RATE_28800
CM_FAX_RATE_31200
```

CM_FAX_RATE_33600
CM_FAX_RATE_19200

If the rate specified on input exceeds that of supported by the underlying hardware then the maximum supported rate will be used instead.

Resolution

[in/out] On input specifies the desired fax resolution for the fax transmission. On output specifies the factual fax resolution after transmission. This can be one of the following values defined in the *CmLib.h*:

CM_FAX_RESOLUTION_LOW – standard fax resolution (200x100)

CM_FAX_RESOLUTION_HIGH – fine fax resolution (200x200)

CM_FAX_RESOLUTION_SUPERHIGH – superfine fax resolution (200x400 or 300x300), not currently supported

CM_FAX_RESOLUTION_ULTRAHIGH – ultrafine fax resolution (400x400), not currently supported

If the resolution specified on input exceeds that of supported by the underlying hardware then the maximum supported resolution will be used instead.

NumOfPagesSent

[out] Specifies the number of fax pages successfully transmitted.

NumOfBadPages

[out] Specifies the number of pages that had bad lines. This member is returned for Dialogic and Brooktrout platforms only.

ErrorMsg

[out] Pointer to a null-terminated string that specifies the hardware device error message in case the fax transmission failed.

CombinePages

[in] Set to *TRUE* to combine all fax pages to a single page fax.

N.B.: this feature is supported by Dialogic and Brooktrout only. Ignored for Pika and Eicon Diva Server.

RetryCNT

[in] Dialogic specific, ignored by the other platforms. Defines the maximum allowable number of fax connection retries for a session. Allowed values are:

0 – no retry (default)

1 – retry once

2 – retry twice

3 – retry three times

RTN

[in] Dialogic specific, ignored by the other platforms. Specifies the percent of bad scan lines accepted for a fax page before an RTN (Retrain Negative) message is returned to the transmitter. Valid values are integers from 1 to 100. Default value is 15.

RTP

[in] Dialogic specific, ignored by the other platforms. Specifies the percent of bad scan lines accepted for a fax page before an RTP (Retrain Positive) message is returned to the transmitter. Valid values are integers from 1 to 100. Default value is 5.

Description

This structure is used by outbound/inbound fax transmission functions. On input this structure specifies fax parameters such as maximum baud rate, local Id, fax header and desired fax resolution. On output it receives factual parameters of the fax transmission performed and, if any, an error message passed by the hardware device.

See also

[CMSendFaxFiles\(\)](#), [CMReceiveFaxFile\(\)](#)

Appendix A (Helper function index)

This chapter gives a description of the CM API helper functions exported for the developer's convenience. Currently this includes audio processing functions that allow to do one step conversion of audio data stored in various formats.

Audio conversion functions

All functions are listed alphabetically for ease of use.

CMConvertMp3ToRawFile()

Syntax

```
int CMConvertMp3ToRawFile(LPSTR pSource, LPSTR pDest, int iFormat);
```

Parameters

pSource

[in] Pointer to a null-terminated string that specifies the full path of an MP3 file to be converted.

pDest

[in] Pointer to a null-terminated string that specifies the full path of the destination file where the converted raw data will be stored.

iFormat

[in] Specifies the audio format of the destination file. For complete list of possible format values see *Appendix B (Audio formats)*.

Returns

If this function succeeds the return value is zero, otherwise the return value is (-1).

Description

This is a convenience function that may be used to convert MPEG Layer 3 audio (MP3-files) to headerless raw audio files using a specific audio format (see *iFormat* parameter description).

See also

[CMConvertMp3ToWaveFile\(\)](#), [Appendix B](#)

CMConvertMp3ToWaveFile()

Syntax

```
int CMConvertMp3ToWaveFile(LPSTR pSource, LPSTR pDest, int iFormat);
```

Parameters

pSource

[in] Pointer to a null-terminated string that specifies the full path of an MP3 file to be converted.

pDest

[in] Pointer to a null-terminated string that specifies the full path of the destination WAVE file where the converted data will be stored.

iFormat

[in] Specifies the audio format of the destination file. For complete list of possible format values see *Appendix B (Audio formats)*.

Returns

If this function succeeds the return value is zero, otherwise the return value is (-1).

Description

This is a convenience function that may be used to convert MPEG Layer 3 audio (MP3-files) to WAVE audio files using a specific audio format (see *iFormat* parameter description).

See also

[CMConvertMp3ToRawFile\(\)](#), [Appendix B](#)

CMConvertRawToRawFile()

Syntax

```
int CMConvertRawToRawFile (LPSTR pSource, LPSTR pDest, int iSrcFormat, int iDstFormat);
```

Parameters

pSource

[in] Pointer to a null-terminated string that specifies the full path of a raw audio file to be converted.

pDest

[in] Pointer to a null-terminated string that specifies the full path of the destination file where the converted raw data will be stored.

iSrcFormat

[in] Specifies the audio format of the source file. For complete list of possible format values see *Appendix B (Audio formats)*.

iDstFormat

[in] Specifies the audio format of the destination file. For complete list of possible format values see *Appendix B (Audio formats)*.

Returns

If this function succeeds the return value is zero, otherwise the return value is (-1).

Description

This is a convenience function that may be used to convert headerless raw audio files from one format to another.

See also

[CMConvertRawToWaveFile\(\)](#), [Appendix B](#), [CMConvertWaveToRawFile\(\)](#)

CMConvertRawToWaveFile()

Syntax

```
int CMConvertRawToWaveFile (LPSTR pSource, LPSTR pDest, int iSrcFormat, int iDstFormat);
```

Parameters

pSource

[in] Pointer to a null-terminated string that specifies the full path of a raw audio file to be converted.

pDest

[in] Pointer to a null-terminated string that specifies the full path of the destination WAVE file where the converted data will be stored.

iSrcFormat

[in] Specifies the audio format of the source file. For complete list of possible format values see *Appendix B (Audio formats)*.

iDstFormat

[in] Specifies the audio format of the destination file. For complete list of possible format values see *Appendix B (Audio formats)*.

Returns

If this function succeeds the return value is zero, otherwise the return value is (-1).

Description

This is a convenience function that may be used to convert headerless raw audio files to WAVE files. The resulting file may contain audio data in another or same format as the source file. In the latter case no real conversion is done just the required WAVE header is added.

See also

[CMConvertRawToRawFile\(\)](#), [Appendix B](#), [CMConvertWaveToRawFile\(\)](#)

CMConvertWaveToRawFile()

Syntax

```
int CMConvertRawToRawFile (LPSTR pSource, LPSTR pDest, int iSrcFormat, int iDstFormat);
```

Parameters

pSource

[in] Pointer to a null-terminated string that specifies the full path of a WAVE file to be converted.

pDest

[in] Pointer to a null-terminated string that specifies the full path of the destination file where the converted raw data will be stored.

iFormat

[in] Specifies the audio format of the destination file. For complete list of possible format values see *Appendix B (Audio formats)*.

Returns

If this function succeeds the return value is zero, otherwise the return value is (-1).

Description

This is a convenience function that may be used to convert WAVE files to headerless raw data files. The resulting file may contain audio data in another or same format as the source file. In the latter case no real conversion is done just the WAVE header is removed.

See also

[CMConvertWaveToWaveFile\(\)](#), [Appendix B](#), [CMConvertRawToWaveFile\(\)](#)

CMConvertWaveToWaveFile()

Syntax

```
int CMConvertWaveToWaveFile (LPSTR pSource, LPSTR pDest, int iSrcFormat, int iDstFormat);
```

Parameters

pSource

[in] Pointer to a null-terminated string that specifies the full path of a WAVE file to be converted.

pDest

[in] Pointer to a null-terminated string that specifies the full path of the destination WAVE file where the converted data will be stored.

iFormat

[in] Specifies the audio format of the destination file. For complete list of possible format values see *Appendix B (Audio formats)*.

Returns

If this function succeeds the return value is zero, otherwise the return value is (-1).

Description

This is a convenience function that may be used to convert WAVE files from one format to another.

See also

[CMConvertWaveToRawFile\(\)](#), [Appendix B](#), [CMConvertRawToWaveFile\(\)](#)

Appendix B (Audio formats)

This appendix describes audio formats supported by the CM API library for voice streaming and recording. All format values are defined in the *CmLib.h* header file which is an essential part of the CM API release.

Pika proprietary ADPCM formats.

This section lists audio formats that can be used for playback and recording on Pika hardware only and *they cannot be converted to other non-Pika formats*. They are supported “as is” for full Pika compatibility:

```
CM_AUDIO_3bit_4kHz_PIKA_ADPCM
CM_AUDIO_3bit_6kHz_PIKA_ADPCM
CM_AUDIO_3bit_8kHz_PIKA_ADPCM
CM_AUDIO_3bit_11kHz_PIKA_ADPCM
CM_AUDIO_4bit_4kHz_PIKA_ADPCM
CM_AUDIO_4bit_6kHz_PIKA_ADPCM
CM_AUDIO_4bit_8kHz_PIKA_ADPCM
CM_AUDIO_4bit_11kHz_PIKA_ADPCM
```

Next formats shown below may be freely played and recorded on all hardware platforms supported by the library. Any audio data conversion, if needed, is performed internally. These formats are interchangeable in the library framework and can be converted to each other with audio processing helper functions exported for developer's convenience.

Oki-Dialogic ADPCM formats.

```
CM_AUDIO_4bit_6kHz_OKI_ADPCM
CM_AUDIO_4bit_8kHz_OKI_ADPCM
CM_AUDIO_4bit_11kHz_OKI_ADPCM
```

A-law/u-law PCM audio. :

```
CM_AUDIO_8bit_4kHz_aLaw_PCM
CM_AUDIO_8bit_6kHz_aLaw_PCM
CM_AUDIO_8bit_8kHz_aLaw_PCM
CM_AUDIO_8bit_11kHz_aLaw_PCM
CM_AUDIO_8bit_4kHz_uLaw_PCM
CM_AUDIO_8bit_6kHz_uLaw_PCM
CM_AUDIO_8bit_8kHz_uLaw_PCM
CM_AUDIO_8bit_11kHz_uLaw_PCM
```

Linear audio formats. :

```
CM_AUDIO_8bit_4kHz_PCM
CM_AUDIO_8bit_6kHz_PCM
CM_AUDIO_8bit_8kHz_PCM
CM_AUDIO_8bit_11kHz_PCM
CM_AUDIO_16bit_4kHz_PCM
CM_AUDIO_16bit_6kHz_PCM
CM_AUDIO_16bit_8kHz_PCM
CM_AUDIO_16bit_11kHz_PCM
```

Besides the formats listed above the CM library allows to play MPEG Layer 3 audio (MP3-files) on any kind of board, this is done by internal conversion to an appropriate format that is supported by the underlying hardware. Recording of MP3-files are not supported.