Dialogic®

Dialogic® Diva® API

Developer's Reference Guide

Part of the Dialogic® Diva® Software Development Kit

Fifth Edition (October 2007) **206-444-05**

Copyright and Legal Disclaimer

Dialogic Corporation License Agreement for Use of Software

This is an Agreement between you, the Company, and your Affiliates (referred to in some instances as "You" and in other instances as "Company") and all your Authorized Users and Dialogic Corporation ("Dialogic").

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE INSTALLING OR DOWNLOADING THE SOFTWARE. IF YOU AGREE WITH THESE TERMS YOU MAY PROCEED WITH THE DOWNLOAD OR INSTALLATION OF THE SOFTWARE. IF YOU DO NOT AGREE WITH THESE TERMS, PLEASE RETURN THE PACKAGE IN "AS NEW" CONDITION (INCLUDING DOCUMENTATION AND BINDERS OR OTHER CONTAINERS) AND YOUR MONEY WILL BE REFUNDED. DOWNLOADING OR INSTALLING THE SOFTWARE CONSTITUTES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. YOU ASSUME RESPONSIBILITY FOR THE SELECTION OF THE PROGRAM TO ACHIEVE YOUR INTENDED RESULTS, AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM THE PACKAGE.

Copyright

The enclosed Software ("Program") and documents are owned by Dialogic Corporation ("Dialogic") and its suppliers and are protected by copyright laws and international treaty provisions. Therefore, You and your Authorized Users must treat the Program and documentation like any other copyrighted material except as expressly permitted in this License Agreement.

License

Under the terms and conditions of this License Agreement:

- You may install and use one copy of the Program on a single-user computer, file server, or on a workstation of a local area network, and only in conjunction with a legally acquired Dialogic hardware or software product;
- The primary Authorized User on the computer on which the "Program" is installed may make a second copy for his/her exclusive use on either a home or portable computer;
- You may copy the Program into any machine readable or printed form for backup or modification purposes in support of your use of one copy of the Program;
- You may make one copy of Dialogic's documentation provided that all copyright notices contained within the documentation are retained;
- You may modify the Program and/or merge it into another Program for your use in one computer; (any portion of this Program will continue to be subject to the terms and conditions of this Agreement);
- You may transfer the Program, documentation and the license to another eligible party within your Company if the other party agrees to accept the terms and conditions of this Agreement. If You transfer the Program and documentation, You must at the same time either transfer all copies whether in printed or machine readable form to the same party or destroy any copies not transferred; this includes all modifications and portions of the Program contained in or merged into other Programs;
- You must reproduce and include the copyright notice on any copy, modification or portion of the Program merged into another Program;
- You may not rent or lease the Program. You may not reverse engineer, decompile or disassemble the Program. You may not use, copy, modify or transfer the Program and documentation, or any copy, modification or merged portion, in whole or in part, except as expressly provided for in this License Agreement;
- If You transfer possession of any copy, modification or merged portion of the Program or documentation to another party in any way other than as expressly permitted in this License Agreement, this license is automatically terminated.

Upgrades

If the Program is provided as an upgrade and the upgrade is an upgrade from another software product licensed to You and Your Authorized Users by Dialogic, the upgrade is governed by the License Agreement earlier provided with that software product package and the present License Agreement does not grant you additional license(s).

Term

The license is effective until terminated. You may terminate it at any time by destroying the Program and documentation together with all copies, modifications and merged portions in any form. It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any terms or conditions of this Agreement. You agree upon such termination to destroy the Program and documentation together with all copies, modifications and merged portions in any form.

Limited Warranty

The only warranty Dialogic makes is that the medium on which the Program is recorded will be replaced without charge if Dialogic, in good faith, determines that it was defective in materials or workmanship and if returned to your supplier with a copy of your receipt within ninety (90) days from the date you received it. Dialogic offers no warranty for your reproduction of the Program. This Limited Warranty is void if failure of the Program has resulted from accident, misuse, abuse, or misapplication.

Customer Remedies

Dialogic's entire liability and You and Your Authorized Users exclusive remedy shall be, at Dialogic's option, either (a) return of the price paid or (b) repair or replacement of the Program that does not meet the above Limited Warranty. Any replacement Program will be warranted for the remainder of the original Warranty period.

No Other Warranties

Dialogic disclaims all other warranties, either expressed or implied, including but not limited to implied warranties or merchantability and fitness for a particular purpose and the warranty against latent defects, with respect to the Program and the accompanying documentation. This limited warranty gives You specific legal rights. You may have others, which may vary from jurisdiction to jurisdiction.

No Liability for Consequential Damage

In no event shall Dialogic or its suppliers be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of information, or other pecuniary loss and indirect, consequential, incidental, economic, or punitive damages) arising out of the use of or inability to use this Program, even if Dialogic has been advised of the possibility of such damages. As some jurisdictions do not allow the exclusion or limitation for certain damages, some of the above limitations may not apply to You or your Authorized Users.

Limit of Liability

Dialogic's entire aggregate liability under any provision of this agreement shall be limited to the amount actually paid by You for the affected Program.

Right to Audit

If this Program is licensed for use in a Company, your Company agrees to keep all usual and proper records and books of accounts and all usual proper entries relating to each reproduction and Authorized User of the Program during the term of this Agreement and for a period of three (3) years thereafter. During this period, Dialogic may cause an audit to be made of the applicable records in order to verify Your compliance with this Agreement and prompt adjustment shall be made to compensate for any errors or omissions disclosed by such audit. Any such audit shall be conducted by an independent certified public accountant selected by Dialogic and shall be conducted during the regular business hours at Your offices and in such a manner as not to interfere with Your normal business activities. Any such audit shall be paid for by Dialogic unless material discrepancies are disclosed. For such purposes, "material discrepancies" shall mean three percent (3%) or more of the Authorized Users within the Company. If material discrepancies are disclosed, Your Company agrees to pay Dialogic for the costs associated with the audit as well as the license fees for the additional Authorized Users. In no event shall audits be made more frequently than semi-annually unless the immediately preceding audit disclosed a material discrepancy.

Supplementary Software

Any Supplementary Software provided with the Dialogic Program referred to in this License Agreement is provided "as is" with no warranty of any kind.

U.S. Government Restricted Rights

The Program and documentation are provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph c) 1) ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraph c) 1) and 2) of the Commercial Computer Software-Restricted Rights at 48 CFR52.227-19, as applicable.

Governing Law

This Agreement shall be construed and controlled by the laws in force in the Province of Quebec, Canada.

Contractor/ manufacturer is:

Dialogic CORPORATION.

9800 Cavendish Blvd., Montreal, Quebec, Canada H4M 2V9

This Agreement has been drafted in English at the express wish of the parties. Ce contrat a été rédigé en anglais à la demande expresse des parties.

Contents

Copyright and Legal Disclaimer **2**

Dialogic Corporation License Agreement for Use of Software **3**

About this Online Guide **12**

How to use this online guide 12

Structure of this guide 12

Dialogic® Diva® SDK Overview **13**

Requirements for installation 13

Installation 13

Diva SDK application programming interfaces 14

Dialogic® Diva® API Overview **17**

Prerequisites 17

Dialogic® Diva® API objectives 17

Samples 18

Dialogic® Diva® API function call interface 19

Getting started 20

Dialogic® Diva® API Functions **33**

DivaInitialize 33

DivaTerminate 33

DivaGetVersion 34

DivaGetVersionEx 34

DivaGetNumLineDevices 35

DivaGetLineDeviceInfo 35

DivaCheckDeviceCapabilities 36

DivaRegister 36

DivaUnregister 38

DivaSetLineDeviceParamsFax 38

DivaSetLineDeviceParamsVoice 39

DivaGetLineDeviceConfiguration 39

DivaGetLineDeviceStatus 40

DivaSetLineDeviceStatusEvents 41

DivaEnableExtensions 41

DivaDisableExtensions 42

DivaGetDeviceName 42

DivaDeviceMgmtGetValue 43

DivaDeviceMgmtSetValue 44

DivaDeviceMgmtExecute 45

DivaGetChannelStatus 45

DivaSetChannelStatus 46

DivaGetSystemConfiguration 46

DivaSystemConfigurationActive 47

DivaSetAnalogHookState 48

DivaCreateCall 49

DivaDial 49

DivaListen 50
DivaProceeding 51
DivaAlert 51
DivaAttachToCall 52
DivaAnswer 52
DivaAnswerFax 53
DivaAnswerVoice 54
DivaAnswerVoIP (RTP) 55
DivaAnswerModem 55
DivaAnswerSMS 56
DivaReject 57
DivaConnect 58
DivaConnectFax 59
DivaConnectVoice 60
DivaConnectVoIP 61
DivaConnectModem 62
DivaConnectSMS 63
DivaSetCallType 64
DivaSetCallTypeFax 65
DivaSetCallTypeVoice 66
DivaSetCallTypeVoIP 66
DivaDisconnect 67
DivaGetCallInfo 67
DivaCloseCall 68
DivaEnableAnsweringMachineDetector 68
DivaDisableAnsweringMachineDetector 69
DivaSendData 70
DivaReceiveData 70
DivaSendFrame 71
DivaReceiveFrame 72
DivaSendFax 73
DivaSendMultipleFaxFiles 73
DivaReceiveFax 74
DivaAppendFax 75
DivaAppendFaxFiles 76
DivaReceiveFaxToMemory 77
DivaReadFaxData 77
DivaSendVoiceFile 79
DivaSendMultipleVoiceFiles 79
DivaSendVoiceEx 80
DivaAppendVoice 81
DivaStopSending 82
DivaPauseSend 82
DivaContinueSend 82
DivaForwardSend 83
DivaRewindSend 83
DivaGetSendPosition 84
DivaPauseRecording 84
DivaContinueRecording 85
DivaGetRecordPosition 85
DivaSetVolume 86

DivaEnableEchoCanceller 86
DivaRecordVoiceFile 87
DivaReceiveAudio 88
DivaStopRecording 89
DivaGetVoiceFileLength 89
DivaSetVoiceFileLength 90
DivaRecordAppendVoiceFile 90
DivaReportDTMF 92
DivaSendDTMF 93
DivaReportTones 93
DivaSendTone 94
DivaSendContinuousTone 94
DivaStopContinuousTone 95
DivaGenerateSingleTone 95
DivaGenerateDualTone 96
DivaStopToneGeneration 97
DivaDetectSingleTone 98
DivaDetectDualTone 99
DivaGetToneDetectorResult 100
DivaSendGenericToneRequest 100
DivaGetGenericToneInfo 101
DivaSetDTMFProcessingRules 102
DivaGetDTMFBuffer 103
DivaClearDTMFBuffer 104
DivaSetupCallTransfer 106
DivaCompleteCallTransfer 107
DivaBlindCallTransfer 107
DivaLIConnect 108
DivaLIDisconnect 109
DivaLIEnableRxData 109
DivaHold 110
DivaRetrieve 110
DivaSendInfo 111
DivaSendFlash 111
DivaCreateConference 113
DivaDestroyConference 113
DivaConferenceSetProperties 114
DivaAddToConference 114
DivaRemoveFromConference 115
DivaGetConferenceInfo 116
DivaConferenceEnableRxData 116
DivaConferenceGetProperties 117
DivaMWIActivate 118
DivaMWIDeactivate 118
DivaSetCallProperties 120
DivaGetCallProperties 121
DivaDefaultCallProperties 121
Callback function 123
CallbackEx function 123
CallbackSignal function 124
DivaGetEvent 125

DivaCreateMonitor 126
DivaDestroyMonitor 127
DivaMonitorGetCallInfo 127
DivaMonitorGetCallProperties 128
DivaMonitorGetSetupMessage 129
DivaMonitorCloseCallHandle 129
DivaMonitorRecordAudio 130
DivaMonitorStopAudio 131
DivaMonitorSetVolume 131
DivaMonitorEnableAudioData 132
DivaMonitorDisableAudioData 133
DivaMonitorReceiveAudio 133
DivaRegisterAudioProvider 135
DivaReleaseAudioProvider 135
DivaConnectAudioProvider 136
DivaDisconnectAudioProvider 136
DivaAPSendAudio 137
DivaAPStopSendAudio 138
DivaAPSetRecordFormat 138
DivaAPSetVolume 139
DivaAPCloseAudio 140
APNotifyCall 140
APNotifyCallClose 141
APNotifyReceiveAudio 141
APConfirmAudioSend 142
DivaStartCallTimer 143
DivaStopCallTimer 143
DivaStartApplicationTimer 144
DivaStopApplicationTimer 144
DivaEnableTrace 145
DivaSetTraceFile 145
DivaLogPrintf 146

Dialogic® Diva® API Events........ 147

DivaEventIncomingCall 147
DivaEventCallInfo 147
DivaEventCallProgress 147
DivaEventCallConnected 147
DivaEventCallDisconnected 147
DivaEventEarlyDataChannelConnected 148
DivaEventSendVoiceDone 148
DivaEventSendVoiceEnded 148
DivaEventSendVoiceRestarted 148
DivaEventSendVoiceCanceled 148
DivaEventRecordVoiceEnded 148
DivaEventFaxPageSent 148
DivaEventFaxSent 148
DivaEventFaxPageReceived 148
DivaEventFaxReceived 148
DivaEventFaxPageSent 149
DivaEventFaxPageReceived 149

DivaEventDataAvailable 149
DivaEventDataSent 149
DivaEventDTMFReceived 149
DivaEventHoldCompleted 149
DivaEventRetrieveCompleted 149
DivaEventSetupTransferCompleted 149
DivaEventTransferCompleted 150
DivaEventSendToneEnded 150
DivaEventSendDTMFToneEnded 150
DivaEventToneDetected 150
DivaEventConferenceInfo 150
DivaEventLIConnectCompleted 150
DivaEventLIDisconnected 151
DivaEventMonitorStatus 151
DivaEventMonitorCallInitiated 151
DivaEventMonitorCallConnected 151
DivaEventMonitorCallDisconnected 151
DivaEventMonitorCallInfo 151
DivaEventMonitorRecordEnded 151
DivaEventDeviceStatusChanged 152
DivaEventGenericToneEnded 152
DivaEventGenericToneDetected 152
DivaEventGenericToneInfo 152
DivaEventSms1MsgReceived 152
DivaEventSmsError 152
DivaEventMWICompleted 152
DivaEventAnsweringMachineDetector 152
DivaEventCallTimer 152
DivaEventApplicationTimer 153
DivaEventDTMFMaxDigits 153
DivaEventDTMFTerminationDigit 153
DivaEventDTMFInterDigitTimeout 153
DivaEventDTMFInitialDigitTimeout 153
DivaEventCallHoldNotify 153
DivaEventCallRetrievedNotify 153
DivaEventCallTransferredNotify 153
DivaEventFaxDocumentSent 153
DivaEventFlashCompleted 153
DivaEventDTMFMaxTimeout 154
DivaEventMonitorAudioData 154

Dialogic® Diva® API Data Structures and Defines 155
DivaCallType 155
DivaListenType 156
DivaLineDeviceInfo 156
DivaLineDeviceParamsFax 158
DivaLineDeviceParamsVoice 159
DivaEventModes 159
DivaCallState 160
DivaCallInfo 161
DivaDisconnectReasons 163

DivaRedirectReason 165
DivaSignalledCallType 166
DivaReturnCodes 167
DivaFaxFormat 168
DivaExtensions 169
DivaLineCodec 169
DivaAudioFormat 169
DivaFaxOptions 170
Dialog®DivaFaxResolution 171
DivaVoiceOptions 171
DivaVoIPParams 172
DivaPayloadProtocol 173
DivaPayloadOptions 173
DivaModemOptions 174
DivaFaxMaxSpeed 176
DivaTransferOptions 176
DivaContinuousTones 177
DivaMultiFrequencyTones 177
DivaVoiceDataSource 178
DivaVoicePositionFormat 178
DivaVoiceDescriptor 178
DivaConferencePropertyType 179
DivaConferenceRights 180
DivaConferenceMemberInfo 181
DivaConferenceMemberRights 181
DivaConferenceSupervisor 181
DivaConferenceOptions 182
DivaConferenceState 182
DivaConferenceInfo 183
DivaCallPropertyType 184
DivaFaxPageQuality 195
DivaFaxPageEnd 195
DivaModulationClass 195
Extended modem parameters 196
V18 Properties 196
Plain Protocol parameter setting 196
DivaBinaryData 197
DivaPlainNumber 197
DivaNumberInformation 197
DivaCallPropertyValue 198
DivaV18DefProbings 198
DivaV18Framing 198
DivaConnectedNorm 199
DivaMonitorSource 199
DivaMonitorStatus 200
DivaTime 200
DivaCallTimeStatistics 201
DivaRecordEndReasons 201
DivaIdFormat 202
DivaIdDescriptor 202
DivaAPNotifyCallInParams 202

DivaAPNotifyCallOutParams 203
DivaVolume 204
DivaVoicePosition 204
DivaDirection 204
DivaSignalService 205
DivaDeviceConfigType 206
DivaDeviceConfigValue 207
DivaDeviceStatusType 207
DivaDeviceStatusValue 208
DivaSwitchType 208
DivaLayer2Mode 209
DivaLayer1Status 209
DivaPotsLineStatus 209
DivaLayer2Status 210
DivaDSPState 210
DivaDSPStateArray 211
DivaDeviceStatusEvents 211
DivaGenericToneFunction 211
DivaSingleToneReport 212
DivaDualToneReport 212
DivaGenericToneResultType 212
DivaGenericToneResult 212
DivaToneDetectorResults 213
DivaGenericToneInfo 213
DivaActiveDiscReasons 214
DivaSMSProtocol 214
DivaMessageStatus 214
DivaMessageNumberInfo 214
DivaMessageInvokeMode 215
DivaMWIActivateParams 215
DivaMWIDeactivateParams 216
DivaResultAnsweringMachineDetector 216
DivaTerminationDigits 217
DivaProcessingGroup 217
DivaSendVoiceEndReasons 217
DivaSysConfCallDirection 218
DivaSysConfType 219
DivaSysConfValue 219
DivaDeviceCapabilities 219
DivaTraceLevel 220
DivaChannelStatus 220
DivaDataOptions 221
DivaFaxScanLineMax 221
DivaFaxStoreModes 222

CHAPTER 1

About this Online Guide

How to use this online guide

- To view a section, click the corresponding bookmark located on the left.
- To view a topic that contains further information, click the corresponding blue underlined phrase.
- You may wish to print out the pages required for developing your communication application.

Structure of this guide

This guide presents implementation details and functional descriptions of all commands in the Dialogic® Diva® API Library interface. Examples are provided where needed. Constants, data structures, and return codes are also provided.

This guide is structured as follows:

Section	Contents
Dialogic® Diva® SDK Overview	Introduction to the Dialogic® Diva® software development kit and its application programming interfaces: the Dialogic® Diva® API and the Extended CAPI 2.0
Dialogic® Diva® API Overview	Introduction to the components provided with the Dialogic® Diva® API and prerequisites for using the Diva API
Dialogic® Diva® API Functions	Description of all functions provided with the Diva API
Dialogic® Diva® API Events	Description of all events provided with the Diva API
Dialogic® Diva® API Data Structures and Defines	Description of the structures and defines used in the Diva API

Note: As of December 2015, Dialogic no longer supports TAPI on the Dialogic® Diva® platform.

CHAPTER 2

Dialogic® Diva® SDK Overview

The Dialogic Diva SDK can be used in combination with Dialogic® Diva® Media Boards and provides two application programming interfaces (APIs), the extended CAPI 2.0 and the Dialogic® Diva® API, the latter allows for developing communication applications for all Dialogic® communication platforms.

It is planned that new versions of the Diva SDK will be released periodically, and it is intended that such new versions will be backwards compatible so as to allow applications developed on the basis of earlier versions of the Diva SDK to be used with the new versions.

The Diva SDK includes the following components:

- Libraries providing functions to access the Dialogic® communication platforms
- Programming samples in source code
- Documentation explaining all functions of the Diva SDK

Requirements for installation

- Installed and running Dialogic® communication platform
- Installed CAPI
- Installed GNU C/C++ compiler version 2.xx or 3.xx
- Installed threading library (pthread)

Installation

The Dialogic® Diva® SDK is provided as a RPM package that contains the documentation with the binaries, header files, and the samples. Use a package tool or the command line version of RPM to install the Diva SDK.

To install use: `rpm -i dssdk-<version>-1.i386.rpm`
To upgrade use: `rpm -U dssdk-<version>-1.i386.rpm`

Due to binary incompatibilities between version 2.xx and 3.xx of the GNU C/C++ compiler, the SDK is provided for both mainstream versions of this compiler as separate RPM packages. Please use the correct package suitable to the installed compiler on your system. The installation of the wrong package on your system will be faulty.

After installation, the following files have available:

File(s)	Description
/usr/include/dssdk.h	Diva SDK interface specification and constants
/usr/libDivaS.a	Diva SDK static library
/usr/libDivaS.so, *.so.1, *.so.1.<version>	Diva SDK shared library and symbolic links
/usr/share/doc/packages/dssdk/CxDtmf.pdf	Documentation about proprietary DTMF extensions to CAPI interface
/usr/share/doc/packages/dssdk/CxEcho.pdf	Documentation about proprietary echo cancelling extensions to CAPI interface
/usr/share/doc/packages/dssdk/CxFax.pdf	Documentation about proprietary FAX extensions to CAPI interface
/usr/share/doc/packages/dssdk/CxModem.pdf	Documentation about proprietary Modem extensions to CAPI interface
/usr/share/doc/packages/dssdk/CxTone.pdf	Documentation about proprietary tone generation and recognition extensions to CAPI interface
/usr/share/doc/packages/dssdk/DivaSAPI.pdf	The main Diva SDK documentation
/usr/share/doc/packages/dssdk/readme.html	Documentation about installation of Diva SDK
/usr/share/doc/packages/dssdk/examples.tgz	Archive that contains the samples

If you unpack the file "examples.tgz" into the current directory, one directory for each sample is created. The created directories with a short description of the sample are listed in the table below. A more detailed description of each sample including the sample subdirectory tree and file structure is provided by the "readme.html" file in each sample directory.

Directory	**Sample Description**
audiomonitor	Monitoring calls and record audio streams
audiomonitorex	Interactively monitoring calls and record audio streams
faxdial	Sample for processing outgoing fax calls
faxinsimple	Mainstream sample for fax reception
faxoutsimple	Mainstream sample for sending fax
faxserver	Simple faxserver
smsservicecenter	SMS service center
voiceext1	Simple answering machine or processing incoming voice calls
voiceext2	CTI-sample for voice processing and call transfer
voiceinsetvolume	Streaming audio with volume contol
voiceinsimple	Answering machine
voiceonleasedline	Streaming audio data on leased lines
voiceoutsimple	Announcement machine

To use the Dialogic® Diva® SDK in your application, you have to include the header "dssdk.h" in the source files and "smssdk.h" if you use SMS functionality, and link your application with either the static library "libDivaS.a" or the shared library "libDivaS.so". Because the Diva SDK uses threads internally, the "pthread" library needs to be linked additionally.

The samples that are provided with the Diva SDK contain details on compiling and linking your application. In every sample subdirectory you will find a makefile that describes the compilation and link process of that application.

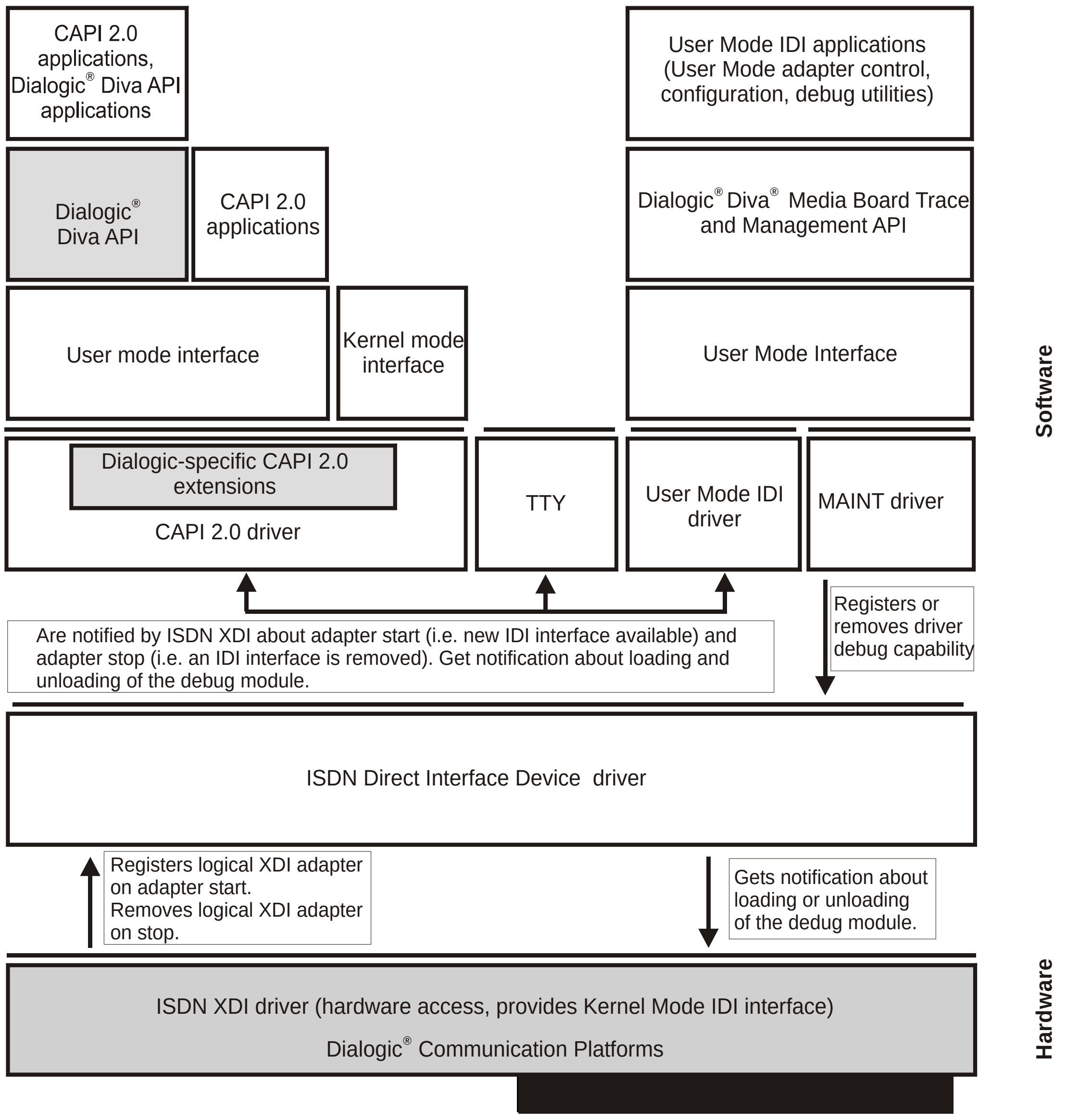
The Diva SDK is freely available for download and distributed with the Dialogic® communication platforms. No licences are needed for developing applications based on the software development kit.

Diva SDK application programming interfaces

The two application programming interfaces (APIs) of the Dialogic® Diva® SDK represent different layers for the management and development of applications for Dialogic® communication platforms.

- Dialogic® Diva® API: It provides a high-level interface into the communication platforms that allows developers to implement communication applications. It also provides an additional library for data conversion like TIFF to SFF for fax applications.
- Extended CAPI 2.0: It provides Dialogic-specific CAPI extensions that are fully CAPI 2.0 compliant.

See the position of the application programming interfaces within the Dialogic® software structure on the next page. The interfaces are highlighted in gray.


CAPI 2.0 applications, Dialogic® Diva API applications
Dialogic® Diva API
CAPI 2.0 applications
User mode interface
Kernel mode interface
User Mode IDI applications (User Mode adapter control, configuration, debug utilities)
Dialogic® Diva® Media Board Trace and Management API
User Mode Interface
Dialogic-specific CAPI 2.0 extensions
CAPI 2.0 driver
TTY
User Mode IDI driver
MAINT driver
Software
Are notified by ISDN XDI about adapter start (i.e. new IDI interface available) and adapter stop (i.e. an IDI interface is removed). Get notification about loading and unloading of the debug module.
Registers or removes driver debug capability
ISDN Direct Interface Device driver
Registers logical XDI adapter on adapter start. Removes logical XDI adapter on stop.
Gets notification about loading or unloading of the dedug module.
ISDN XDI driver (hardware access, provides Kernel Mode IDI interface)
Dialogic® Communication Platforms
Hardware

Dialogic® Diva® API

The Diva API is a high-level interface into the Dialogic® communication platforms via a library of "C" function calls. This interface can allow developers to implement various communication applications faster and easier than in the traditional CAPI 2.0 application development.

The Diva API contains modules that can be used as basis for communication applications, such as fax and voice transfer or call control, and in the development of applications for these areas. The modules are intended to be updated so as to offer development bases for additional communication applications.

Even if the Diva API abstracts functions and provides a high level interface, access to low level functions is optionally available. Applications that require access to low level operations, e.g., control over signaling messages, can be performed on the Diva API. This allows existing applications to be extended using the same API, even if the requirements change. The CAPI 2.0 extensions are also available on the Diva API.

The Diva API also allows for access to the management interface of the Dialogic® Diva® Media Board for status and statistic information.

Extended CAPI 2.0

The Extended CAPI 2.0 is only available for Dialogic® Diva® Media Boards and provides Dialogic-specific extensions for CAPI 2.0. The extensions are fully CAPI 2.0 compatible, and thus can be used with CAPI 2.0 applications. The following Dialogic-specific CAPI extensions are available:

- Echo canceller support for voice applications: This extension allows the voice application to place an echo canceller unit in the front end of a connection to suppress acoustical echo and signal return. The Dialogic extension and the new CAPI standard for echo canceller are supported.
- Extension for fax paper formats and resolutions: This extension enables fax transmission and reception with an extended range of paper formats and resolutions.
- Tone detection and generation extension for DTMF facility: This extension enables fax and voice applications to detect in-band signals such as busy tone, to report events like modem CNG or fax flag detection, to detect human speech, to report the unidentified tones, and to report that no signal is present on the line.
- Extensions for modem configuration: This extension enables to specify certain modulation and protocol-related parameters. Modulations can be removed from the auto moding list or specific modulations can be selected. The results of the modulation and the protocol negotiation are signaled to the application.
- Generic tone generator and detector support for voice applications: This extension provides built-in generic tone detector and generator facilities.The generic tone services include sine generators with programmable frequency and amplitude modulation, function generators with programmable signal shape, frequency, and amplitude modulation, noise generators with programmable crest factor and amplitude modulation, single tone detection, and dual tone detection.

Descriptions of the Dialogic-specific CAPI 2.0 extensions are available under `SDK/DOC`. The CAPI 2.0 specification can be downloaded from the web site www.capi.org.

CHAPTER 3

Dialogic® Diva® API Overview

The Dialogic® Diva® API is a high-level interface that provides a basis for developing communication applications for specific tasks, e.g., fax applications. It provides applications to access the communication resources of Dialogic® communication platforms.The abstraction level is very high, without reducing the required flexibility. Connection management is reduced to a simple function call. In general, the Diva API provides functions that combine several steps of the CAPI in one function.

The Diva API is available as a function-oriented C-call interface.

The Diva API abstracts all communication details and concentrates on the global tasks of connection management and data transfer. In addition, it provides functions to access certain supplementary services. This interface allows developers to implement various communication applications faster and easier than the traditional CAPI 2.0 application development.

The Diva API Library relies on the Dialogic® Diva® System Release for Dialogic® Diva® Media Boards.

With the above software, the Diva API allows an application to control the ISDN features of a Dialogic® Diva® BRI, 4BRI, PRI, 2PRI, 4PRI, or Analog Media Boards.

Thus the Diva API Library can help in the development of communication applications using a Dialogic communication platform.

Prerequisites

This manual assumes that the developer has knowledge of C or C++ programming with the GNU C/C++ compiler collection.

Dialogic® Diva® API objectives

Since the Diva API can facilitate development of communication applications, it must fulfill the requirements of various types of applications.

Call setup

Call setup on signaling platforms can be very different. Starting with a simple analog call, where only the phone number to dial is needed, up to connections using ISDN-specific messages (user/user data) and high-level protocols that require negotiation in the B-channel (like X.25) or IP-based protocols.

The Diva API provides a set of simple call setup functions to establish calls.

In general, applications support one specific set of services, e.g., voice or fax. For these applications, the call setup should not contain specific parameters of other services, even if they are optional. In general, only one function call is necessary to create or answer a call. Event reporting of the call progress is optional.

Event reporting

In general, call establishment is an asynchronous process. Depending on the used protocol or service, e.g., fax, information is exchanged or negotiated between the two peers. This information is available and can be signaled to the application. The application can choose if events should be signaled and how they are signaled.

Events can be signaled in the following ways:

- Callback function including the event information
- Callback function that only notifies that an event is available

Depending on the mechanism, the information about the event may be provided directly, e.g., as parameter to the callback; or has to be retrieved from the Diva API by a function call.

Implementation dependencies

As already stated earlier, connection establishment is an asynchronous process that may cover several steps.

The Dialogic® Diva® API is an asynchronous API. Applications that require blocking operations should use the Dialogic® Diva® Component API.

How applications use ports or channels

The communication channels provided by the installed Dialogic® Diva® Media Boards can be seen as a pool of resources shared between several applications or from a port oriented view. CAPI-based applications see the channels as a pool of resources. Applications basically designed for serial ports see each available channel as a dedicated resource.

The Dialogic® Diva® API does not reserve resources and does not block resources against access by other applications like serial interfaces do.

Samples

To demonstate the basic design of an SDK-related application, some samples are provided with this SDK. These samples cover fax server, fax client and fax polling, voice sending/recording and monitoring applications. Some samples are designed just to demonstrate basic functionality without any error handling (single source files). All samples are designed to be portable between operating systems.

- audiomonitor - Monitoring calls and recording audio streams:
  This sample shows the monitoring of signaling information and the recording of audio streams between the NT-side and the attached TE-side. The objective is to show the main task of monitoring or audio tapping. This is done without any error handling, so this sample must not be used in productive environments. It is realized as a simple command line program.
- audiomonitorex - Interactively monitoring calls and recording audio streams:
  This sample shows the monitoring of signaling information and the recording of audio streams between the NT-side and the attached TE-side. The objective is to show the main task of interacitvely monitoring or audio tapping. This is done without any error handling, so this sample must not be used in productive environments. It is realized as a simple command line program.
- faxdial - Sample for processing outgoing fax calls:
  This sample shows the processing of multiple outgoing fax calls. The calls can either initiate sending a fax or fax polling. When fax polling is used, the direction is reversed and a fax is received. The sample includes a command line user interface to configure a few parameters and to show active connections and status messages.
- faxinsimple - Mainstream sample for fax reception:
  This sample shows the processing of incoming fax calls and storing the received faxes in a single file. The objective is to show the main tasks of fax reception. This is done without any error handling, so this sample must not be used in productive environments. It is realized as a simple command line program.
- faxoutsimple - Mainstream sample for sending fax:
  This sample shows how to send a fax. The fax data must be available (for this sample) as a file in TIFF Class F Format. The objective is to show the main tasks for sending a fax. This is done without any error handling. Therefore, this sample must not be used in productive environments. It is realized as a simple command line program.
- faxserver - Simple faxserver:
  This sample shows the processing of multiple incoming voice or fax calls and streaming of audio data in outgoing direction. In addition, detection and processing of DTMF and fax calling tones are shown. The sample is designed as a simple command line program, that displays informational output on the terminal.
- voiceext1 - Simple answering machine or processing incoming voice calls:
  This sample shows the processing of multiple incoming voice calls and streaming of audio data in both directions. In addition, detection and processing of DTMF is shown. The sample includes a command line user interface to configure a few parameters and to show active connections and status messages.

- voiceext2 - CTI-sample for voice processing and call transfer:
  This sample shows the processing of multiple incoming voice calls, streaming of audio data in both directions and call transfer to fixed or detected numbers. In addition, detection and processing of DTMF is shown. The sample includes a command line user interface to configure a few parameters and to show active connections and status messages.
- voiceinsetvolume - Streaming audio with volume contol:
  This sample shows the streaming and recording of audio data. The volume of the outgoing and recorded audio stream can be adjusted online. The recorded audio stream is stored in a file. The objective is to show the main tasks of audio streaming and volume control. This is done without any error handling, so this sample must not be used in productive environments. It is realized as a simple command line program.
- voiceinsimple - Answering machine:
  This sample shows the streaming and recording of audio data. The recorded audio stream is stored in a file. The objective is to show the main tasks of audio streaming and recording. This is done without any error handling, so this sample must not be used in productive environments. It is realized as a simple command line program.
- voiceonleasedline - Streaming audio data on leased lines:
  This sample shows the streaming and recording of audio data on leased lines. The sample can stream audio data from a wavefile into the leased line and record voice data out of the leased line into a wavefile. The objective is to show the main tasks of audio streaming on a leased line. This is done without any error handling, so this sample must not be used in productive environments. It is realized as a simple command line program.
- voiceoutsimple - Announcement machine:
  This sample shows the streaming of audio data. The objective is to show the main tasks of making a call and sending an audio stream. This is done without any error handling, so this sample must not be used in productive environments. It is realized as a simple command line program.
- smsservicecenter - a simple SMS service center:
  This sample shows the processing of short messages in the role of an SMS service center. It receives messages and forwards them to their destination. It is command line based and has a simple menu to configure one option (automatic forwarding on/off), trigger an action (forward last received message), and to quit the program.
  Please note that this sample is far from feature-complete and error handling is sparse, so it must not be used in productive environments.

Note: Please see legal notice at the front of this document.

Dialogic® Diva® API function call interface

The function call interface is implemented as a library and provides standard C-function calls. A static and dynamic library and the Dialogic® Diva® API header files are available for the application developer. The following groups of functions are available:

- Retrieve capabilities of installed Dialogic® Diva® Media Boards
- Registration
- Set common parameters for all calls
- Connection management, Connect / Disconnect / Get Status
- Data transfer
  - Voice streaming
  - Fax functions
  - VoIP functions
  - Analog data transfer
  - Digital data transfer
- Supplementary services
- Blind and supervised call transfer
- Conferencing

- Passive Monitoring
- ASR / TTS Integration

Instances

Each application of the Dialogic® Diva® API and each call represents an instance. A call must be identified by the Diva API and by the application. The Diva API identifies different applications and calls based on virtual handles. The handles are valid in the context of a process. Different threads of one process are allowed to share handles. Any application that uses the Diva API interface has to register with the Diva API.

Registration instance

When a process registers with the Diva API, a handle is assigned. This handle must be used in subsequent calls to other Diva API functions. With the registration, the application sets some parameters, e.g., buffer sizes and number of buffers that are valid for all calls done on this registration.

The handle is also used for de-registration when the application terminates or stops all communication services. The Diva API handles a cleanup for all pending actions on this instance. Calls are automatically disconnected and any thread waiting for events is signaled.

Call instance

Each call has a unique call handle at the Diva API level and at the application level. The application handle is optional and only used by the application, the Diva API does not interpret this value. The application handle is signaled with each call-related event and can be used by the application to assign the event to a particular call.

When an outgoing call is initiated, the application provides its call handle to the Diva API and a location where the Diva API places its own handle. An incoming call is signaled with the Diva API handle, the application gives its own handle to the Diva API when accepting the call.

Lifetime of a call instance

A call instance at Diva API level has a defined life time. When an outgoing call is established or an incoming call is signaled, the call instance is created and the handle is reported to the application. The instance is valid until the application calls the functions *DivaCloseCall* or *DivaReject*.

Getting started

This section provides basic information about the functionality of the Dialogic® Diva® API. In general, an application can be handled as described below:

Initialization

1. *Detect installed resources.* (optional)
2. *Get board-specific information, channels, etc.* (optional)
3. *If specific board handling, identify board by serial number.* (optional)
4. Register with the Dialogic® Diva® SDK, either with default parameter or with application-specific parameters.

Monitor system integrity

The Dialogic® communication platforms support various interfaces for different types of applications. Some of these interfaces allow remote access and in some scenarios these interfaces should be disabled for security reasons. The Dialogic® Diva® configuration ensures that these interfaces are securely disabled. However, applications may want to monitor the system configuration to ensure that the configuration is not changed during runtime. The Dialogic® Diva® API provides the functions DivaGetSystemConfiguration and DivaSystemConfigurationActive to retrieve the installed environment.

Event processing

1. Build callback function or event thread.
2. Process necessary events in the callback or event thread.

Initiate call processing

1. For incoming call processing, place a Listen on either all line devices or specific line devices.
2. For outgoing calls, call the suitable connect function.

Call establishment

The call establishment for incoming and outgoing calls can be handled by high-level functions. The Dialogic® Diva® API handles the different media as call types. Common functions for all call types are available as well as functions for specific call types, e.g., for fax and voice.

Outgoing calls

Outgoing calls can be established using one of the *DivaConnect...* functions. Based on a registration handle, the call to *DivaConnect* initiates the connection. The function *DivaConnect* can be used to create calls for any call type using the default parameters. For the call types fax, voice, modem, and VoIP, separate functions exist to set media-specific parameters, e.g., the local identifier for fax calls.

Several events report the progress, e.g., call progress changes and call information changes. The application may process them to display information or ignore them. Two events need to be processed for outgoing calls, the connected and the disconnected event. See the basic frame below for an application that makes an outgoing call and processes the connect and disconnected event.

```
void CallbackHandler ( DivaAppHandle hApp, DivaEvent Event,
                       PVOID Param1, PVOID Param2 )
{
   switch (Event)
   {
   case DivaEventCallConnected:
      printf("Call connected.\n");
      break;

   case DivaEventCallDisconnected:
      printf("Call disconnected.\n");
      DivaCloseCall ( Param2 );
      break;
   }
}

int main(int argc, char* argv[])
{
   hMyCall = (void *) 0x11223344;
   if ( DivaInitialize () != DivaSuccess )return -1;
   if ( DivaRegister (&hApp, DivaEventModeCallback,
                      (void *) CallbackHandler, 0, 1, 7, 2048 ) != DivaSuccess )return -1;

   if ( DivaConnect (hApp, hMyCall, &hSdkCall, "99999",
                    DivaCallTypeVoice, LINEDEV_ALL ) != DivaSuccess )return -1;

   // Call initiated. Add any synchronization to wait for call completion.
   DivaUnregister ( hApp );
   DivaTerminate ();
   return ( 0 );
};
```

Incoming calls

If enabled by *DivaListen*, incoming calls are signaled via events. The application may specify the services to listen for. Optionally, listen can be restricted to a specific called party number, a list of numbers, or a range. By default, the listen is done for all services on all devices.

An incoming call is signaled by the event *DivaEventIncomingCall*. The application may answer the call right away, alert the call to get more time, or monitor the call. Answering the call is handled by *DivaAnswer* or one of the call type specific functions, e.g., *DivaAnswerFax*. The following process is the same as for outgoing calls. Below you can see the basic frame for an application that handles a single incoming call.

```
void CallbackHandler ( DivaAppHandle hApp, DivaEvent Event,
                       PVOID Param1, PVOID Param2 )
{
   switch (Event)
   {
   case  DivaEventIncomingCall:
       printf("Incomimg call.\n");
       hSdkCall = Param1;
       DivaAnswer (hSdkCall, hMyCall, DivaCallTypeVoice );
       break;

   case  DivaEventCallConnected:
       printf("Call connected.\n");
       break;

   case  DivaEventCallDisconnected:
       printf("Call disconnected.\n");
       DivaCloseCall ( Param2 );
       break;
   }
}

int main(int argc, char* argv[])
{
   hMyCall = (void *) 0x11223344;
   if ( DivaInitialize () != DivaSuccess )return -1;
   if ( DivaRegister (&hApp, DivaEventModeCallback,
                      (void *) CallbackHandler, 0, 1, 7, 2048 )
                      != DivaSuccess )
        return -1;

   if ( DivaListen (hApp, DivaListenAll, LINEDEV_ALL, "" )
                      != DivaSuccess)
        return -1;

   // Listen initiated. Add any synchronization to wait for call completion.
   DivaUnregister ( hApp );
   DivaTerminate ();
   return ( 0 );
};
```

Extended call properties

The Dialogic® Diva® API provides high level functions for making and answering calls. The functions allow for specifying necessary parameters and for hiding specific parameters that require knowledge of the underlying protocols. In some cases additional parameters have to be set, or parameters negotiated during connection establishment have to be known and processed by the application. The extended functions for setting and retrieving extended parameters provide this functionality. Those parameters are often specific to the underlying protocol and therefore may require detailed knowledge of the protocol or modulation.

Extended call properties enable applications to set certain parameters during the call setup that are specific to the environment, i.e. signal the call with different bearer capabilities than the standard function would use. For further processing, i.e. voice streaming, the powerful high level functions can be used.

The setting of the extended properties requires a call handle. For incoming calls, the call handle is already available and provided with the event *DivaEventIncomingCall*. The application may set any property using *DivaSetCallProperties*. For outgoing calls, the function *DivaConnect* and the call type specific connect functions return the call type when the call is initiated, which is too late to set the properties. Set the extended call properties for outgoing calls by creating a call object using *DivaCreateCall*. Set the call properties via *DivaSetCallProperties* and initiate the connect by calling *DivaDial* with the destination number.

Fax processing

The Dialogic® Diva® SDK supports high level functions for fax transmission and reception. Conversion from line format to TIFF or SFF format (and vice versa) is handled without any interaction of the application. For details refer to the following chapters.

Fax sending and receiving

Sending and receiving a fax is handled by a single function call. The fax document format is either TIFF class F or SFF. The Diva API processes single- or multi-page documents automatically and signals the progress per page via events. Sending or receiving a fax is initiated once the event *DivaEventCallConnected* is signaled. The application may call the function *DivaSendFax* or *DivaReceiveFax* directly from the event handler. If the application could not call the function right away, the Dialogic® Diva® API will prevent data from being lost.

Upon successfully receiving or sending a fax, the application receives a confirmation event, *DivaEventFaxSent* or *DivaEventFaxReceived*. If this event does not occur and the application receives the disconnect event, the fax transmission has failed and the call information contains the reason.

The standard resolution of the fax format has different resolution for horizontal and vertical orientation. The conversion routines of the Dialogic® Diva® SDK align the resolution upon request by the application.

Below is a sample to start sending a fax from the connect event of the callback function:

```
void CallbackHandler ( DivaAppHandle hApp, DivaEvent Event,
                          PVOID Param1, PVOID Param2 )
 {
     switch (Event)
     {
     case DivaEventCallConnected:
        DivaSendFax ( hSdkCall, "myfax.tif", DivaFaxFormatTIFF_ClassF );
        break;
     }
}
```

Fax polling

The application may allow fax polling for an incoming call or request polling for an outgoing call. The result of the negotiation is available when the connection is established. The call information, retrieved by *DivaGetCallInfo*, contains the state of the polling.

An application calls *DivaSendFax* to process an incoming call that has been negotiated for polling. To receive the polled fax, an application calls *DivaReceiveFax* to process an outgoing call that has requested polling and successfully negotiated polling.

Fax multi-document handling

The application provides the fax documents for sending. They can be sent as one or various SFF or TIFF files to the Dialogic® Diva® SDK. Each file may contain one page or multiple pages.

The fax protocol allows sending several fax documents on one fax connection in order to save an additional connect establishment time. The Diva SDK supports both the sending of multiple pages in different files as one single fax document and the sending of each file as one single fax document. With the append function the application can control the sending mode on a page base.

The behavior of *DivaSendMultipleFaxFiles* is controlled with the options set during connect establishment. If the option *DivaFaxOptionMultipleDocument* is set, the pages included in each file are sent as a separate document. If the option is not set, all pages of all files are sent as one document.

With the function *DivaAppendFax* the application can add faxes during a running transmission. For each file added with *DivaAppendFax*, the application can specify if the pages included in the file as belonging to the same document or to a new document. This provides more flexibility to the application.

Fax resolution and document formats

The Dialogic® Diva® SDK supports the fax resolutions standard, fine, super fine, and ultra fine. See the below list of formats, ISO and T.30, and the corresponding number of pixels per line.

Format		Pixels per line
ISO A4 at:	R8 x 3.85 R8 x 7.7 R8 x 15.4 200 x 200	1728
ISO B4 at:	R8 x 3.85 R8 x 7.7 R8 x 15.4 200 x 200	2048
ISO A3 at:	R8 x 3.85 R8 x 7.7 R8 x 15.4 200 x 200	2432
ISO A4 at:	300 x 300	2592
ISO B4 at:	300 x 300	3072
ISO A3 at:	300 x 300	3648
ISO A4 at:	R16 x 15.4 400 x 400	3456
ISO B4 at:	R16 x 15.4 400 x 400	4096
ISO A3 at:	R16 x 15.4 400 x 400	4864

The Diva SDK supports the formats listed above. Documents in SFF format must match one of these formats in horizontal resolution. TIFF documents that have a different horizontal pixel count are centered on the next matching format. By default, no stretching is done. If the call property *DivaCPT_FaxAllowDocumentStretching* is enabled, the document is doubled to the next matching pixels per line, e.g., a document with 800 pixels per line is converted to 1600 pixels and centered on the 1728 pixel fax format.

With the event *DivaEventCallConnected*, the capabilities of the receiving side are available. The application may retrieve either the full DIS frame or the resolution and maximum speed capabilities of the receiving side via the call properties *DivaCPT_FaxRemoteFeatures*, *DivaCPT_FaxRemoteMaxResolution*, and *DivaCPT_FaxRemoteMaxSpeed*. Based on this information, the application can pass the data in a valid format.

Color fax

When the application wants to send a color fax, the call property *DivaCPT_FaxEnableColor* needs to be enabled. When the call is connected, the result of the negotiation must be retrieved by the application. If the call property *DivaCPT_FaxColorSelected* is true, color fax has been selected and the application must send a color fax document. If the remote fax machine does not support color fax, the application must either send a corresponding black and white document or disconnect. The SDK does not handle any color conversion.

If the application wants to receive color faxes, the call property *DivaCPT_FaxEnableColor* must be enabled. If this property is enabled, the application must check the property *DivaCPT_FaxColorSelected* before calling *DivaReceiveFax*. The format given to DivaReceiveFax must match the negotiated formats. If the format does not match, the function will return DivaErrorInvalidParameter.

By default, all received pages of a color fax document are stored in a single file. The functions *DivaSendFax* and *DivaReceiveFax* support the format *DivaFaxFormatColorJPEG*. All other fax sending and receiving related functions return the error *DivaErrorUnsupportedFormat*.

Sending and receiving Non-Standard Facilities

The fax protocol allows the exchange of so called Non-Standard Facilities (NSF). These frames are used by some fax machines to exchange a symbolic station name. The Dialogic® Diva® SDK provides received frames to the application and allows to send frames. The call property *DivaCPT_FaxLocalNSF* is used to specify the NSF to be send to the remote peer. The call property *DivaCPT_FaxRemoteNSF* provides a received NSF frame. The data is not interpreted by the SDK.

Voice processing

The Dialogic® Diva® API supports audio streaming in several ways. Simple functions to stream one or more audio files are available as well as functions to stream from memory. Different audio formats are supported, either wave file based or as raw format without any header information. Audio can be streamed continuously or up to a certain amount of time.

Received audio can be recorded to a file or saved to the memory in a specified audio format.

The volume of the audio can be set by the application in the range of -18 to +18 db. The setting can be done separately for inbound and outbound streaming.

Control inbound streaming

Received audio data, which is recorded to an audio file, can be controlled by the application. The recording can be paused and continued at any time. The application may retrieve the position from the start of the recording either as recorded bytes or as recorded milliseconds.

Start and duration of the recording can be controlled by several properties. With the property *DivaCPT_VoiceRecordSilenceTimeout* the application may specify a timeout of silence when the recording should be terminated. If the stream is terminated, the reason is delivered with the event that sends a notification about the terminated streaming.

The start of the recording can be delayed until a specific tone is detected. The property *DivaCPT_VoiceRecordStartTones* allows specifying a list of start tones.

Control outbound streaming

Applications can stream and control audio. Playing can be paused and continued at any time. The application may position the streaming by forward and rewind operations. The current position of the streaming can be retrieved at any time. The position is reset to zero when a new streaming is activated. Note that control of the audio streaming can be done only within one active streaming. Appended streaming is handled separately.

Streaming during connection establishment

In some cases, audio streaming is already available before the connection is confirmed. For example, an announcement if a wrong number is dialed. The Dialogic® Diva® SDK supports the establishment of an audio channel for the call type "voice" during the connection establishment by setting the *DivaVoiceOptionEarlyDataChannel* (as) in the voice options. This option allows the receiving of audio streams and the detection of tones.

DTMF tone generation and detection

The Dialogic® Diva® SDK supports DTMF detection and reporting on all Dialogic® communication platforms. This includes the detection of fax calling tones and fax and modem answer tones. The tone support and the generic tone support are only available if the Dialogic communication platform is equipped with DSP resources per channel. The application gets the information if a line device is able to do tone support by calling *DivaGetLineDeviceInfo* and checking the parameter *bExtVoiceSupported* in the returned information.

Detection of DTMF must be enabled before the Diva SDK will signal detected DTMF digits. Digits are signaled via the Event *DivaEventDTMFReceived*. The received digit is directly passed with the event. DTMF detection is enabled via the function *DivaReportDTMF*. The detection parameter for duration and pause of the signal can be set by the call properties *DivaCPT_VoiceDTMF_SendDuration* and *DivaCPT_VoiceDTMF_SendPause* prior to calling *DivaReportDTMF*.

Digits can be sent via the function *DivaSendDTMF*. The event *DivaEventSendDTMFToneEnded* is signaled when the tone has been sent on the physical link.

Termination rules for DTMF events

In addition to the events that are signaled when a DTMF tone is received the Dialogic® Diva® SDK can handle rules to combine several received DTMF tones to signal a specific event or even to terminate a streaming action. The following rules can be defined using DivaSetDTMFProcessingRules:

- Termination digits, defined by a digit mask
- Maximum number of received digits
- Maximum inter digit delay (time between digits)
- Maximum initial digit delay (time for receiving the first digit)
- Maximum timeout (if no other rule expires before)

The termination rules can be combined and initiate the following actions:

- Signal an event, e.g., *DivaEventDTMFTerminationDigit*, *DivaEventDTMFMaxDigits*, *DivaEventDTMFMaxInterDigitDelay*, *DivaEventDTMFInitialDigitTimeout, DivaEventDTMFMaxTimeout*
- Stop an ongoing outbound streaming
- Stop an ongoing recording to a file

Note: Different rules can be set for each action. The Diva SDK stores the DTMF digits internally in a buffer, maximum 128 digits. The application can retrieve the content of the buffer and clear the buffer. When the rule is set and every time a DTMF digit is detected, the termination rules are checked and the corresponding action is done. If a streaming operation is terminated by one of these rules, the event that signals the end of the streaming contains the reason. Refer to DivaRecordEndReasons and DivaSendVoiceEndReasons.

Human Talker and Voice Activity Detections

The Dialogic® Diva® SDK supports Human Talker and Voice Activity Detection (VAD) as part of the "tone support". These functionalities require DSP resources. In general, the VAD and tone support is handled in the same way as DTMF detection. The reporting of VAD must be enabled via *DivaReportTones*. Any changes are signaled via the event *DivaEventToneDetected*. The available tones are defined in *DivaContinuousTones*. The end of a human talker, VAD, or any other tone is signaled via *DivaEndOfTones*. Note that not all Dialogic® communication platforms support talker detection.

Generic tone detector and generator

Besides the generation and detection of predefined tones, the Diva SDK supports the generation and detection of generic tones. The application may specify certain parameters, like the frequency and amplitude for the generation of a single or dual tone, or the range for a tone detection. Note that not all Dialogic® communication platforms support tone detection.

The SDK supports high level functions to generate and detect single and dual tones as well as low level functions for extended functionality.

The high level functions allow for setting basic parameters like frequency and amplitude. The functions take integer values for the parameter, and are easy to handle.

The extended functionality allows access to the generator and detector setup for filter curves and filtering points. The format of the frames is described in CxTone.pdf. The usage of the low level functions requires knowledge of digital signal processing.

Plain data processing

Once the data channel is established, data can be exchanged. For the call types voice and fax, a set of specific functions is available. For applications that run a proprietary protocol, the raw data the functions *DivaSendData* and *DivaReceiveData* are available. For voice and fax applications, raw data processing is not recommended.

Received data is signaled via the event *DivaEventDataAvailable*. The application may read the data using *DivaReceiveData*. The Dialogic® Diva® SDK will only signal new received data if the application has read the data.

Applications send raw data using *DivaSendData*. The Diva SDK confirms that the data has been sent by the event *DivaEventDataSent*. During this time the SDK owns the data buffer.

For the call type voice, the data format is the raw bit transparent stream on the line. Applications using bit transparent data exchange, e.g., 3G applications, may use the call type voice. For the call type fax, the data is coded in the SFF format. All other formats contain proprietary data.

Changing media

The Dialogic® Diva® SDK supports changing the media type for an established call. An application may answer a call in voice mode, detect a fax tone, and switch to the call type fax. This is done by the *DivaSetCallType* function or one of the call type specific functions.

When the media or call type change is initiated by the application, the current data channel is disconnected and the new data channel with the new call type is established. The application may set call properties for the new call type before calling *DivaSetCallType*. The establishment of the new data channel is signaled by the event *DivaEventCallConnected*. The establishment may take some seconds depending on the used call type.

Supplementary services

This section describes the supplementary services: Call Transfer, Conference, and Line Interconnect.

Call Transfer

A call transfer can be handled with or without a consultation call. The Dialogic® Diva® API supports both. A call transfer without consultation call, termed a blind transfer, is handled by *DivaBlindTranfer*. Based on an existing call, the Dialogic® Diva® SDK fully handles the call transfer, including the transition of the original call to the hold state, if necessary.

The application may establish two separate calls, either incoming or outgoing, and transfer them later by using *DivaCompleteTransfer*.

In some environments, the second call may be required to complete a call transfer, e.g., establishment of the outgoing call on a specific channel. If the application uses *DivaSetupCallTransfer* to establish or create a call, the Diva API will address this requirement.

Conference

Conferences are sessions that include more than two parties simultaneously. They can be created using either an external server-based bridge/mixer, e.g., the Dialogic® Diva® Media Board, or a switch-based conference bridge inside a PBX. The Dialogic® Diva® API supports the feature rich conference sessions provided by the Dialogic® communication platform.

Conference setup and management

Dialogic® Diva® Media Boards have mixing capabilities and handle conferences on their own. This can be useful in environments dealing with large numbers of conferences with multiple members. In such an environment, members of a conference can be added and removed simultaneously without putting the conference on hold or interrupting the data stream.

The Dialogic® Diva® SDK creates a conference handle for each conference. The conference handle has the same capabilities for voice functions as a call handle. Using the conference handle for data streaming, audio data from conference participants can be received from and sent to all. The single call objects remain.

Voice streaming

The Dialogic® communication platforms mix the received data of participating members into the common data stream. The application that manages and monitors the conference may also want to participate in this conference. In addition, a supervisor may want to send information only to specific members.

Application may record a conference or stream an announcement to all conference members. Depending on the capabilities of the underlying platform, applications may also stream audio to specific members only.

Conference design and definitions

The Dialogic® Diva® API implements conferences handled by Dialogic® Diva® Media Boards because this provides flexibility to the application.

Conference implementation details

Each conference consists of one conference object and several call objects. An application that wants to set up a conference creates a conference object based on an already existing call by calling *DivaCreateConference*.

After the conference object is created, the properties of the conference may be set, e.g., the maximum number of members. By default, the number of members is not limited. Properties are set by calling *DivaConferenceSetProperties* indicating the type of property and the value. Properties should be set before the first call is added to the conference. The properties will be extended with support for a switch-based functionality. Existing applications will run without any changes.

In general, any call can be added to a conference. However, due to the installed hardware or the switch environment, there might be some limitations.

Calls are added to and removed from a conference using *DivaAddToConference* and *DivaRemoveFromConference*.

When the information on the conference changes, applications will receive the event *DivaEventConferenceInfo*. The application may retrieve information on the conference such as state and members at any time by calling *DivaGetConferenceInfo*.

Data streaming

For data streaming, standard streaming functions for sending and recording voice data are available. Data channels are switched internally; the application does not need to consider this.

The conference handle is a valid handle for all voice streaming functions. Using the conference handle to receive voice data, e.g., using *DivaRecordVoiceFile*, provides a mixed audio signal from all conference members. Using the conference handle for sending voice, e.g., *DivaSendVoiceFile*, can also be used for streaming. Using a single call handle will only address the particular call.

Only applications that use direct data reception via *DivaReceiveData* have to enable this on the conference object of the call. Enabling and disabling is done by calling *DivaConferenceEnableRxData*.

Tromboning and Line Interconnect

The Tromboning or Line Interconnect of the Dialogic® Diva® SDK is based on two existing voice calls. The application has to establish or answer these calls using standard functions of the Dialogic® Diva® API.

The Diva SDK provides two functions, one to initiate Line Interconnect and another to release Line Interconnect. A pair of interconnected calls has one main call and a participating call.

By default, no data traffic between the application and the interconnected calls is enabled. Data is only streamed between the two endpoints. The application may enable the data streaming by calling one of the data-related functions.

To receive native voice streaming via *DivaReceiveData* the streaming must be enabled. On the main call object, data streaming is always done for both calls, also known as transaction recording. In the receiving direction, the application gets the mixed stream. On the participating call, the data stream contains only the information of this call.

Features

The following Line Interconnect features are supported:

- Create Line Interconnect between two existing calls.
- Receive data on one or both RX-channels optionally.
- Get mixed data streams of both RX-channels on one call object optionally.
- Have data streaming on one or both lines optionally.

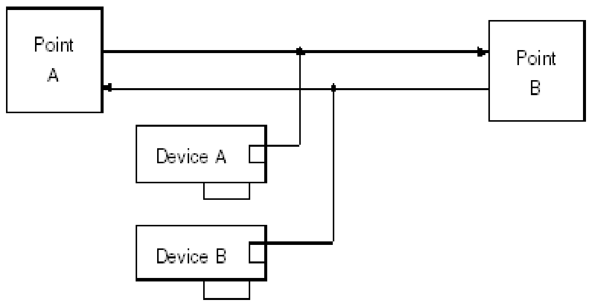
Monitoring and tapping

In general, Dialogic® Diva® Media Boards are used as one endpoint in the communication, and they actively initiate or answer calls. A Diva Media Board could also be used to monitor a line, using a special configuration of the drivers and a specific cable.

Two line devices are needed to monitor one line and to get the information of both directions. The line devices must be configured in "Monitor Mode" by the Dialogic® Diva® Configuration Manager. The line devices are attached to the line to monitor by a so called Y-cable. For monitoring in the PRI environment, a small box is available to align the impedances. The information to monitor is signaling information and data channel audio.

Note: The monitoring of signaling information is limited to protocols that have a dedicated timeslot for signaling. For protocols using in-band signaling only the data channel audio can be monitored.

When actively involved in a call, the call direction is clearly specified. In case of monitoring a line from point A to point B, it depends on the installation which board monitors the data sent by point A to point B and which board monitors the data from B to A. During initialization of the monitoring mode, the application passes this information to the Dialogic® Diva® API.



The figure above shows that device A records the signaling and audio streaming from point A to point B while device B records the signaling and audio streaming from point B to point A. The audio recording functions identify the direction to be recorded either by call direction or physical direction. The physical direction depends only on the initialization parameter. The call direction may change with any call. For a call made from point A to B, A is the originator and B the answers. For a call made from B to A, B is the originator and A the answerer. If the application specifies the recording based on endpoints, e.g., A to B, this is independent from the call origination.

The monitoring functions provide the information about the side that initiates the call in the parameter "LineDevice" of DivaCallInfo belonging to a monitored call.

Signaling channel monitoring

Monitoring of the signaling channel can be done at different layers depending on the application requirements. The Dialogic® Diva® API combines the information monitored on the two lines and assigns them to calls. This allows the application to retrieve high level events that signal the state change of the call and provide the call information as well as parameters like called and calling party number. The information is retrieved using standard data structures on the monitoring functions. High level events signal the calls and their progress. The monitoring events are:

- DivaEventMonitorCallInitiated
- DivaEventMonitorCallConnected
- DivaEventMonitorCallDisconnected
- DivaEventMonitorCallInfo

All high level monitoring events provide the handle of the monitored call as EventSpecific2 parameter. The application may use the *DivaMonitorGetCallInfo* or *DivaMonitorGetCallProperties* functions to retrieve the information. The parameter LineDevice of *DivaCallInfo* always contains the device that initiates the call. The channel referring to this call is available in the parameter "AssignedBChannel" of *DivaCallInfo*.

The plain setup frame is provided to the application in addition to the decoded information that is delivered by the high level functions. The setup frame contains the raw frame. The application may retrieve the switch-dependent information that is needed.

Data channel monitoring

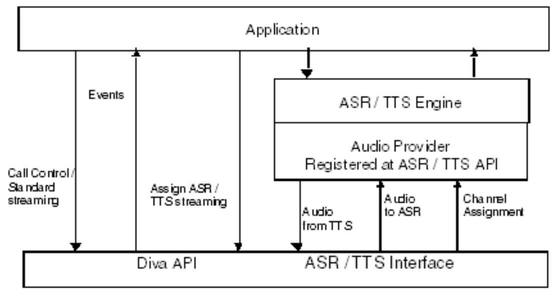
The data channel monitoring is restricted to audio. The Dialogic® Diva® API provides functions to record the received audio stream to a file. There are two independent audio streams, one for each direction. The Diva API has the ability to combine these two streams into a single audio file with two channels (stereo). The application can also record each direction into a single file. The recording can start once the data channel is known. The data channel is either assigned with the call initiation or during call progress. If assigned during call initiation, the data channel is already available with the event *DivaEventMonitorCallInitiated*. If the channel is not known at that time, the event *DivaEventMonitorCallInfo* signals the change.

With the high level monitoring interface, the audio signal can only be recorded to a file. Applications that need the audio data in memory or require DTMF and tone detection may use high level functions to monitor the signaling. The data channel monitoring is then done by creating "calls" on each line device using the "leased line mode". Thus simulated calls are created. They are handled like normal calls with the restriction that no outbound streaming functions are available.

The Dialogic® Diva® SDK contains several samples for monitoring that show the modes described above. Please refer to the samples for more information.

Audio provider (ASR/TTS) interface

Generally, a voice application using the Dialogic® Diva® API controls the call setup and the audio streaming directly. This is also done by audio providers of speech recognition and text to speech engines that have telephony support. If audio providers for ASR / TTS handle only audio streaming, the audio streaming and the call control need to be split. In addition, the received audio signal may be processed by two instances, the application for recording purposes and the speech recognition for detection or words.



The above figure shows the overall architecture. The streaming part of the ASR / TTS engine registers to the Dialogic® Diva® SDK via the so called "Audio Provider Interface". The application uses the standard registration at the Diva API.

Note: The application and the audio provider(s) must run under the same process context.

During registration at the Audio Provider Interface function, entry points for notification of established calls and streaming commands are exchanged. When the application decides to connect the streaming of an established call to the audio provider interface, it calls *DivaConnectAudioProvider*. The Diva SDK now creates a link between the call at the Diva SDK and the audio provider and routes streaming directly to / from the audio provider. The application can still use the audio-related functions of the Diva API, e.g., to record the audio signal in parallel.

The interface allows the selection of the streaming direction. This implies that separate providers, one for ASR and another for TTS, can be assigned to one call.

The assignment of the audio provider is done by a symbolic name. When the audio provider registers at the Audio Provider Interface, it passes a symbolic name to the Diva API. When the application calls *DivaConnectAudioProvider* it uses the same name to identify the provider. Assignment of an instance or channel of the ASR / TTS engine is done by a device identifier. The identifier depends on the ASR / TTS engine and can be a symbolic name or a binary ID. When the application connects the audio provider to a call, it passes the information on how to identify the channel to the Diva SDK. The Diva SDK notifies the audio provider about this connect request and passes the identifier. The audio provider compares the given identifier with the identifier assigned by the ASR / TTS engine and connects the corresponding audio channels. Once the assignment is done, function pointers are exchanged and the audio is streamed without any further interaction of the application. The streaming is automatically handled between the Diva SDK and the audio provider.

The audio provider must implement the following functions:

- APNotifyCall
- APNotifyCallClose
- APNotifyReceiveAudio (only if ASR support)
- APConfirmAudioSend (only if TTS support)

Device management

The Dialogic® Diva® API virtualizes the available communication resources by different line devices. The line devices are numbered from 1 to the number of installed lines. The application can retrieve information about installed devices at any time. There are three categories of information:

- Device Information, e.g., amount of channels and capabilities
- Device Configuration, e.g., signaling protocol
- Device Status, e.g., Layer 1 status

All information can be retrieved without registration at the Diva API. The functions *DivaGetNumLineDeviceInfo*, *DivaGetLineDeviceInfo*, *DivaGetLineDeviceConfiguration*, and *DivaGetLineDeviceStatus* do not require a registration.

The device status information may change during runtime. Those changes can be signaled to the applications via the standard event reporting mechanism. To get these notifications an application must register via *DivaRegister* and enable the status events via *DivaEnableLineStatusEvents*.

Answering machine detector

The answering machine detection is done based on the length of the initial announcement after an outgoing call is connected. In general, the answering machine detector can be done with the functions of the Dialogic® Diva® SDK; however, it requires several steps for the application and requires timer handling by the application. In order to have a high level function, the Diva SDK combines this under a single function call and an event.

The application starts the detector based on an outgoing call. This can be done at any time from initiating the call up to the event *DivaEventCallConnected*. The application provides two parameters, the maximum initial silence and the minimum time of speech that is interpreted as automatic announcement.

The detection process starts when the call is connected. Typically the called person will answer the call with either "Hello" or "John Smith speaking" and wait for the caller to respond. The SDK will note the time when the talker starts and ends. If the human speech stops before the minimum time of speech expires, the event *DivaEventAnsweringMachineDetector* is signaled and the detection is set to *DivaResultAMD_Human*. If the minimum time of speech is reached, the detection result *DivaResultAMD_Machine* is signaled. The maximum initial silence is an optional parameter to terminate the detector if no signal is received from the remote end. The detector will also terminate if a fax or modem tone is detected.

Note: The detection process is based on the human talker and silence detector. Call properties that modify the parameter of these detectors are also valid for the answering machine detector.

Timer events

The Dialogic® Diva® SDK notifies the application whenever an event occurs. Applications can register for a timer event. When the timer expires, an event is signaled to the application. The event can be based on a call or on an application registration. The application calls *DivaStartCallTimer* or *DivaStartApplicationTimer* to enable the timer. When the timer expires, the corresponding event *DivaEventCallTimer* or *DivaEventApplicationTimer* is signaled to the application. The timer resolution is 100 milliseconds. All timers are single shot timers.

Tracing

The Dialogic® Diva® SDK supports tracing into a text-based file. By default, the tracing is disabled. Applications may enable tracing on demand at any time by calling the function *DivaEnableTrace* to set a new trace level. Enabling the tracing also requires a valid trace file name. The application may overwrite the default name (`C:\Temp\dssdk.log`) by any valid file name. The path for this file must exist, the file will be automatically created.

Once enabled, the Diva SDK will write trace messages to the specified file. The application may use the SDK trace interface to write own messages into the same file. This can be done by the function *DivaLogPrintf*.

CHAPTER 4

Dialogic® Diva® API Functions

The Dialogic® Diva® API offers common functions for all call types as well as functions for specific call types, e.g., for fax and voice.

Startup and version

To ensure that the resources for the Dialogic® Diva® API are properly allocated during start and released when the application terminates, the Diva API must be initialized. In addition, the application may query and verify the version number of the Diva API.

DivaInitialize

The *DivaInitialize* function initializes the internal core of the Dialogic® Diva® API.

```
DWORD          DivaInitialize (        )
```

Parameters

None.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). In case of an error, the function returns *DivaErrorNoCapi*.

Remarks

This function must be called before any other Diva API function is called. This is a synchronous function. When DivaInitialize returned the initialization is completed.

See also

DivaTerminate

DivaTerminate

The *DivaTerminate* function frees all internal resources allocated by the core of the Dialogic® Diva® API.

```
void        DivaTerminate (          )
```

Parameters

None.

Return values

None.

Remarks

This function must be called before the application exits. All internal resources are returned to the system. Any pending call is terminated.

See also

DivaInitialize

DivaGetVersion

The *DivaGetVersion* function returns the current version of the Dialogic® Diva® API.

```
DWORD          DivaGetVersion (          )
```

Parameters

None.

Return values

The high word contains the major version and the low word contains the minor version.

Remarks

The application can query the version number to verify that the correct Diva API is available.

See also

DivaGetVersionEx

DivaGetVersionEx

The *DivaGetVersionEx* function returns the current version of the Dialogic® Diva® API.

```
DWORD      DivaGetVersionEx (   DWORD      *pProductVersionMajor,
                                DWORD      *pProductVersionMinor,
                                DWORD      *pRevision,
                                DWORD      *pBuildVersionMinor );
```

Parameters

pProductVersionMajor

[out] This parameter points to the location where the Dialogic® Diva® SDK places the major product version number.

pProductVersionMinor

[out] This parameter points to the location where the Diva SDK places the minor product version number.

pRevision

[out] This parameter points to the location where the Diva SDK places the major revision number. The revision is reserved for future use and is currently set to zero.

pBuildVersionMinor

[out] This parameter points to the location where the Diva SDK places the minor build product version number.

Return values

The function returns *DivaSuccess (0)* if the version information are correctly returned. Another possible return value is *DivaErrorInvalidParameter.*

Remarks

The application can query the version and build number to verify that the correct Diva API is available.

See also

DivaGetVersion

Capabilities, registration, and information

The Dialogic® Diva® API virtualizes the available communication resources by different line devices. The line devices are numbered from 1 to the number of installed lines. Please note that an installed Dialogic® Diva® Media Board may have more than one line device.

A line device has a defined number of channels. Line devices based on a Dialogic® Diva® BRI Media Board have two channels. For line devices that depend on a Dialogic® Diva® PRI Media Board, the number of channels depends on the used protocol. For fractional T1 or E1 lines, the line device's number of channels depends on the capabilities of the line. For IP-based line devices, the amount of channel depends on the software license. The application can obtain the number of channels for each line device at any time.

DivaGetNumLineDevices

The *DivaGetNumLineDevices* function gets the number of available line devices. All line devices are numbered from one to the maximum number of devices.

```
DWORD      DivaGetNumLineDevices (      DWORD       *pNumLine );
```

Parameters

pNumLine

[out] This parameter is a pointer to a location that receives the number of available physical lines.

Return values

If the function succeeds, the return value is *DivaSuccess* (0).

Remarks

The function returns the number of available lines that can be accessed via the Dialogic® Diva® API. To obtain more information on the lines, the application may call *DivaGetLineDeviceInfo* and *DivaCheckDeviceCapabilities* for each line.

See also

DivaGetLineDeviceInfo

DivaGetLineDeviceInfo

The *DivaGetLineDeviceInfo* function gets information on the capabilities of the line device, e.g., voice, fax, and supplementary services.

```
DWORD     DivaGetLineDeviceInfo (     DWORD                   LineDeviceId,
                                      DivaLineDeviceInfo      *pInfo );
```

Parameters

LineDeviceId

[in] The *LineDeviceId* parameter identifies the line device by an index starting with one.

pInfo

[out] This parameter is a pointer to a buffer that receives the information on the given line device. Note that the buffer must be of the type *DivaLineDeviceInfo* and the length field of the data structure must be set to the size of the structure.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorLineDevice* and *DivaErrorInvalidParameter*.

Remarks

The function retrieves specific information about the given line device.

See also

DivaLineDeviceInfo

DivaCheckDeviceCapabilities

DivaCheckDeviceCapabilities returns information if the device has the specified capability.

```
BOOL      DivaCheckDeviceCapabilities (      DWORD      LineDevice,
                                             DWORD      Capability );
```

Parameters

LineDevice

[in] The parameter *LineDevice* identifies the line device by an index starting with one.

Capability

[in] The parameter *Capability* specifies the capability to be validated. For valid capabilities see DivaDeviceCapabilities.

Return values

If the line device supports the capability, the function returns TRUE. If the capability is not supported, the function returns FALSE.

Remarks

The function returns the information if a specified capability is supported by the line device. The capabilities are defined in DivaDeviceCapabilities. The function is a synchronous function and returns right away.

See also

No references.

DivaRegister

The *DivaRegister* function registers with the Dialogic® Diva® API and sets global parameters and event reporting.

```
DWORD     DivaRegister (     DivaAppHandle     *pHandle,
                             DWORD             EventMode,
                             void              *EventModeSpecific1,
                             void              *EventModeSpecific2,
                             DWORD             MaxConn,
                             DWORD             RxBuffers,
                             DWORD             MaxBufferSize );
```

Parameters

pHandle

[out] This parameter is a pointer to a location that receives the handle for all further access to the Diva API.

EventMode

[in] This parameter specifies how the application handles events. The event mode must be one of the modes specified by DivaEventModes.

EventModeSpecific1

[in] This parameter depends on the event mode. For details refer to remarks and to DivaEventModes.

EventModeSpecific2

[in] This parameter depends on the event mode. For details refer to remarks and to DivaEventModes.

MaxConn

[in] This parameter specifies the maximum number of connections to be used by the application. If this parameter is set to zero, a maximum of one connection per available physical channel can be used.

RxBuffers

[in] This parameter specifies the number of data blocks that should be reserved for each connection in receive direction.

MaxBufferSize

[in] This parameter specifies the maximum buffer size to be used. See remarks below. The default registration uses the value 256.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorLineDevice*, *DivaErrorInvalidHandle*, and *DivaErrorInvalidParameter*.

Remarks

This function must be called by the application before any connection-oriented function can be called. The application registers the parameters to be used for the amount of concurrent connections and the data buffer management.

The *MaxBufferSize* and *RxBuffers* parameters are valid for all connections done on this registration at the Diva API. The defaults are selected to have an optimal situation for fax and voice connections. The amount of buffers should be between 4 and 8. This amount of buffers is used for receiving data and also for sending data. The *MaxBufferSize* depends on the application requirements. If only fax or data calls are handled, the maximum of 2048 should be used. For voice applications the delay may be important. If the application is delay sensitive, a buffer size or 256 should not be exceeded. If the application is not delay sensitive, a buffer set to 1024 or even 2048 is recommended. The buffer size should not be below 128.

The registration is a synchronous process; however, most of the function calls following the registration are asynchronous and the result is reported via an event. The application selects the event mode with the parameter EventMode. The event mode specific parameters are:

EventMode	**EventModeSpecific1**	**EventModeSpecific2**
DivaEventModeCallback	Callback function entry	NULL
DivaEventModeCallbackEx	Callback function entry	Application context. This context is not interpreted by the Dialogic® Diva® SDK and passed to the callback function.
DivaEventModeCallbackSignal	Callback function entry	Application context. This context is not interpreted by the Diva SDK and passed to the callback function.

See also

DivaUnregister

DivaUnregister

The *DivaUnregister* function releases all allocated resources at the Dialogic® Diva® API.

```
DWORD        DivaUnregister (   DivaAppHandle        Handle );
```

Parameters

Handle

[in] The application handle that was returned by a call to *DivaRegister*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

This function is called to release the resources allocated to this instance. The application typically calls this function before its termination. Note that all pending calls are cleared implicitly. It may take some time until the final cleanup is confirmed. This is a synchronous function and the execution is blocked until cleanup is done.

See also

DivaRegister

DivaSetLineDeviceParamsFax

The *DivaSetLineDeviceParamsFax* function sets the fax parameters that should be used for all calls on this line device, e.g., the calling number to be signaled to the remote side.

```
DWORD   DivaSetLineDeviceParamsFax (      DivaAppHandle              hApp,
                                          DWORD                      LineDeviceId,
                                          DivaLineDeviceParamsFax    *Params );
```

Parameters

hApp

[in] *hApp* is the application handle that was returned by a call to *DivaRegister*.

LineDeviceId

[in] This parameter identifies the line device. Line devices are continuously numbered by an index starting with one.

Params

[in] This parameter is a pointer to a buffer that contains the default parameter settings. The buffer is of the type DivaLineDeviceParamsFax.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorLineDevice* and *DivaErrorInvalidParameter*.

Remarks

The function sets defaults for the line device. These defaults may be overwritten for specific calls. This is a synchronous function that returns control right away.

See also

DivaConnect, DivaAnswer, DivaConnectFax, DivaAnswerFax

DivaSetLineDeviceParamsVoice

The *DivaSetLineDeviceParamsVoice* function sets the voice parameters for all calls on this line device, e.g., the calling number to be signaled to the remote side.

```
DWORD    DivaSetLineDeviceParamsVoice (     DivaAppHandle                 hApp,
                                            DWORD                         LineDeviceId,
                                            DivaLineDeviceParamsVoice     *Params );
```

Parameters

hApp

[in] *hApp* is the application handle that was returned by a call to *DivaRegister*.

LineDeviceId

[in] This parameter identifies the line device. Line devices are continuously numbered by an index starting with one.

Params

[in] This parameter is a pointer to a buffer that contains the default parameter settings. The buffer is of the type DivaLineDeviceParamsVoice.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorLineDevice* and *DivaErrorInvalidParameter*.

Remarks

The function sets defaults for the line device. These defaults may be overwritten for specific calls. This is a synchronous function that returns control right away.

See also

DivaConnect, DivaAnswer, DivaConnectVoice, DivaAnswerVoice

DivaGetLineDeviceConfiguration

DivaGetLineDeviceConfiguration returns information about the current line device configuration.

```
DWORD    DivaGetLineDeviceConfiguration (     DWORD                      LineDeviceId,
                                              DivaDeviceConfigType       Type,
                                              DivaDeviceConfigValue      *pValue,
                                              DWORD                      ValueSize );
```

Parameters

LineDeviceId

[in] This parameter identifies the line device by an index starting with one.

Type

[in] This parameter specifies which configuration parameter should be read. Valid types are defined in *DivaDeviceConfigType*.

pValue

[out] This parameter points to a location where the value of the requested type is placed.

ValueSize

[out] This parameter specifies the length in bytes of buffer specified by pValue.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter*, *DivaErrorDataSize*, and *DivaErrorLineDevice*.

Remarks

The function reads the specified configuration property of the specified line device. The available configuration parameters are defined in *DivaDeviceConfigType*.

The configuration data is written to the location pointed to by *pValue*. The application provides the buffer and also the length of the buffer. The configuration data has different lengths and the application does not always need to provide the maximum space defined by the size of *DivaDeviceConfigValue*. The Dialogic® Diva® API will validate the provided length compared to the required length for the specific configuration type.

The function is a synchronous function and returns right away.

See also

No references.

DivaGetLineDeviceStatus

DivaGetLineDeviceStatus returns information about the current line device status.

```
DWORD   DivaGetLineDeviceStatus (    DWORD                      LineDeviceId,
                                     DivaDeviceStatusType       Type,
                                     DivaDeviceStatusValue      *pValue,
                                     DWORD                      ValueSize );
```

Parameters

LineDeviceId

[in] This parameter identifies the line device by an index starting with one.

Type

[in] This parameter specifies which configuration parameter should be read. Valid types are defined in *DivaDeviceStatusType*.

pValue

[out] This parameter points to a location where the value of the requested type is placed.

ValueSize

[out] This parameter specifies the length in bytes of buffer specified by *pValue*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter*, *DivaErrorDataSize*, and *DivaErrorLineDevice*.

Remarks

The function reads the specified status type of the specified line device. The available status options are defined in *DivaDeviceStatusType*.

The status information is written to the location pointed to by *pValue*. The application provides the buffer and also the length of the buffer. The status information has different lengths and the application does not always need to provide the maximum space defined by the size of *DivaDeviceStatusValue*. The Dialogic® Diva® API will validate the provided length compared to the required length for the specific status type.

The function is a synchronous function and returns right away.

See also

No references.

DivaSetLineDeviceStatusEvents

DivaSetLineDeviceStatusEvents specifies the status changes that should be reported to the application.

```
DWORD    DivaSetLineDeviceStatusEvents (    DivaAppHandle             hApp,
                                            DWORD                     LineDeviceId,
                                            DivaDeviceStatusEvents    EventMask );
```

Parameters

hApp

[in] The *hApp* parameter specifies the application handle returned by *DivaRegister*.

LineDeviceId

[in] The *LineDeviceId* parameter identifies the line device by an index starting with one.

EventMask

[in] The *EventMask* parameter specifies which status changes should be reported to the application. Refer to the description of DivaDeviceStatusEvents for options.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorLineDevice*.

Remarks

The function enables or disables the event reporting to the application. The application must register via *DivaRegister*. The status changes are reported via the event *DivaEventDeviceStatusChanged* using the standard event handling. The function is a synchronous function and returns right away.

See also

DivaDeviceStatusEvents, DivaEventDeviceStatusChanged, DivaRegister

DivaEnableExtensions

DivaEnableExtensions activates the given extension.

```
DWORD    DivaEnableExtensions (             DivaAppHandle             hApp,
                                            DWORD                     LineDeviceId,
                                            DivaExtensions            Type );
```

Parameters

hApp

[in] *hApp* is the application handle that was returned by a call to *DivaRegister*.

LineDeviceId

[in] This parameter identifies the line device. Line devices are continuously numbered by an index starting with one.

Type

[in] This parameter specifies the extensions to be enabled. For valid extensions refer to DivaExtensions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function enables the given extensions if supported by the underlying Dialogic® communication platform. The extensions are defined in *DivaExtensions*. A sample for an extension is the support of fax resolutions higher than the fine format. The function is a synchronous function and returns right away.

See also

DivaDisableExtensions

DivaDisableExtensions

DivaDisableExtensions deactivates the given extension.

```
DWORD    DivaDisableExtensions (        DivaAppHandle        hApp,
                                        DWORD                LineDeviceId,
                                        DivaExtensions       Type );
```

Parameters

hApp

[in] *hApp* is the application handle that was returned by a call to *DivaRegister*.

LineDeviceId

[in] This parameter identifies the line device. Line devices are continuously numbered by an index starting with one.

Type

[in] This parameter specifies the extensions to be enabled. For valid extensions refer to DivaExtensions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function disables the given extensions. The extensions are defined in *DivaExtensions*. A sample for an extension is the support of fax resolutions higher than the fine format. If disabled, the system will not signal this capabilities. The function is a synchronous function and returns right away.

See also

DivaEnableExtensions

DivaGetDeviceName

DivaGetDeviceName retrieves the name of the underlying device.

```
DWORD     DivaGetDeviceName (         DWORD                LineDevice,
                                      unsigned char *      pBuffer,
                                      DWORD                BufferSize );
```

Parameters

LineDevice

[in] This parameter identifies the line device by an index starting with one.

pBuffer

[in] The parameter *pBuffer* specifies the location where the name of the board is placed.

BufferSize

[in] The parameter BufferSize specifies the length of the buffer specified by pBuffer.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

Each Dialogic® communication platform has a symbolic name. The function *DivaGetDeviceName* provides the name of the board, for example "Diva PRI CTI" or "Diva softIP v2.2". The function is a synchronous function and returns right away.

See also

No references.

DivaDeviceMgmtGetValue

DivaDeviceMgmtGetValue reads information from the management interface of an underlying Dialogic® communication platform.

```
DWORD     DivaDeviceMgmtGetValue (          DWORD                LineDevice,
                                             char *               pValueName,
                                             unsigned char *      pResultBuffer,
                                             DWORD                BufferSize,
                                             DWORD                *pBytesRead );
```

Parameters

LineDevice

[in] This parameter identifies the line device by an index starting with one.

pValueName

[in] The parameter *pValueName* specifies the parameter to be read.

pResultBuffer

[out] The parameter *pResultBuffer* specifies the location where the result or the read request is placed.

BufferSize

[in] The parameter *BufferSize* specifies the length of the buffer specified by pResultBuffer

pBytesRead

[out] The parameter *pBytesRead* specifies the location where to place the amount of bytes read from the management interface and written to the result buffer.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorNotSupported,* and *DivaErrorInvalidHandle*.

Remarks

The Dialogic® Diva® Media Boards provide an interface to retrieve certain information on the system or a specific call. The interface is structured like a directory tree and the values are addressed by a path and a value name, e.g., "Config\FAX\Options".

This is a synchronous function. The result is placed in the buffer provided by the application. The format is depending on the parameter, options are zero terminated string, DWORD, BOOLEAN etc.

Working with the management interface requires specific knowledge on the interface and the parameter.

Note: Not all Dialogic® communication platforms provide a management interface. If the line device does not support a management interface, the function will fail with the result *DivaErrorNotSupported*.

See also

DivaDeviceMgmtSetValue, DivaDeviceMgmtExecute

DivaDeviceMgmtSetValue

DivaDeviceMgmtSetValue writes information to the management interface of an underlying Dialogic® communication platform.

```
DWORD    DivaDeviceMgmtSetValue (         DWORD              LineDevice,
                                          char *             pValueName,
                                          unsigned char *    pValue,
                                          DWORD              ValueSize,
                                          DWORD *            pBytesWritten );
```

Parameters

LineDevice

[in] This parameter identifies the line device by an index starting with one.

pValueName

[in] The parameter *pValueName* specifies the parameter to be written.

pValue

[in] The parameter *pValue* specifies where the data to be written is located.

ValueSize

[in] The parameter *ValueSize* specifies the length of the date to be written.

pBytesWritten

[out] The parameter *pBytesWritten* specifies the location where to place the amount of bytes written to the management interface.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorNotSupported*, and *DivaErrorInvalidHandle*.

Remarks

The Dialogic® Diva® Media Boards provide an interface to write information, e.g., configuration parameter, to the board. The interface is structured like a directory tree and the values are addressed by a path and value name, e.g., "Config\FAX\Options".

This is a synchronous function. The application must provide the data in the format expected for the specified parameter. Working with the management interface requires specific knowledge on the interface and the parameter.

Note: Not all Dialogic® communication platforms provide a management interface. If the line device does not support a management interface, the function will fail with the result *DivaErrorNotSupported*.

See also

DivaDeviceMgmtGetValue, DivaDeviceMgmtExecute

DivaDeviceMgmtExecute

DivaDeviceMgmtExecute executes a specific function on the management interface of an underlying Dialogic® communication platform.

```
DWORD     DivaDeviceMgmtExecute (        DWORD              LineDevice,
                                         char *             pValueName );
```

Parameters

LineDevice

[in] This parameter identifies the line device by an index starting with one.

pValueName

[in] The parameter *pValueName* specifies the parameter to be executed.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorNotSupported*, and *DivaErrorInvalidHandle*.

Remarks

The Dialogic® Diva® Media Boards provide an interface to read, write, and execute parameters. The interface is structured like a directory tree and the values are addressed by a path and value name, e.g., "Config\FAX\Options". The execute option can be used to instruct the board to flash the LED.

This is a synchronous function. The application must provide the data in the format expected for the specified parameter. Working with the management interface requires specific knowledge on the interface and the parameter.

Note: Not all communication platforms provide a management interface. If the line device does not support a management interface, the function will fail with the result *DivaErrorNotSupported*.

See also

DivaDeviceMgmtGetValue, DivaDeviceMgmtSetValue

DivaGetChannelStatus

DivaGetChannelStatus retrieves the status of a specific channel.

```
DWORD     DivaGetChannelStatus (         DWORD              LineDevice,
                                         DWORD              Channel,
                                         DivaChannelStatus* pResult );
```

Parameters

LineDevice

[in] This parameter identifies the line device by an index starting with one.

Channel

[in] This parameter identifies the channel by an index starting with one. See remarks section for more information.

pResult

[out] The parameter *pResult* specifies the location of type *DivaChannelStatus* where the status is placed. Possible values are defined in DivaChannelStatus.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

Some protocols allow to enable or disable a specific channel of a trunk. The function retrieves the status for the given channel. The channels are continuously numbered from 1 up to the maximum number of channels. If the function returns *DivaSuccess*, the result is written to the location pointed to by *pResult*. If the parameter Channel is set to zero, the status of all channels is returned. The application is responsible for providing a result buffer large enough to cover the status of all channels. On all protocols that do not support channel blocking, the result is always *DivaSuccess* and the status is *DivaChannelStatusUnblocked*.

See also

DivaSetChannelStatus

DivaSetChannelStatus

DivaSetChannelStatus modifies the status of a specific channel.

```
DWORD     DivaSetChannelStatus (          DWORD               LineDevice,
                                          DWORD               Channel,
                                          DivaChannelStatus   Status );
```

Parameters

LineDevice

[in] This parameter identifies the line device by an index starting with one.

Channel

[in] This parameter identifies the channel by an index starting with one.

Status

[in] The parameter *Status* specifies the new state of the channel. Possible options are defined in *DivaChannelStatus*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

Some protocols allow to enabled or disable a specific channel of a trunk. This function sets the channel status. The channels are continuously numbered from 1 up to the maximum number of channels.

If the function returns *DivaSuccess*, the new status has been set. If the channel has already been blocked, the function returns *DivaErrorInvalidState*. If the underlying protocol does not support channel blocking, the return value is *DivaErrorNotSupported*. The function is a synchronous function and returns right away.

See also

DivaGetChannelStatus

DivaGetSystemConfiguration

DivaGetSystemConfiguration retrieves the information on installed interfaces.

```
DWORD     DivaGetSystemConfiguration (    DivaSysConfType     Type,
                                          DivaSysConfValue*   pValue
                                          DWORD               Size );
```

Parameters

Type

[in] The parameter *Type* identifies the type of information to be retrieved. See DivaSysConfType for available types.

pValue

[out] The parameter *pValue* specifies the location of type *DivaSysConfValues* where the configuration information is stored.

Size

[in] The parameter *Size* specifies the size of the buffer specified by *pValue*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorReadFile*, *DivaErrorOpenFile,* and *DivaErrorUnsupportedFormat*.

Remarks

The function accesses the configuration system of the underlying Dialogic® communication platform. If the function returns *DivaErrorUnsupportedFormat,* the version of the platform does not match the Dialogic® Diva® SDK version.

The application calls the function with a specific type of the configuration to retrieve. Valid types are defined in *DivaSysConfTypes*. The function returns the requested information in the provided buffer. The function is a synchronous function and returns right away.

See also

DivaSystemConfigurationActive

DivaSystemConfigurationActive

DivaSystemConfigurationActive returns information if a configuration update is pending.

```
DWORD     DivaSystemConfigurationActive (     BOOL*        bNeedsReboot );
```

Parameters

bNeedsReboot

[out] The parameter *bNeedsReboot* specifies a memory location of type BOOL. The information if a reboot is necessary to active the reported configuration is placed at this location.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorNotSupported*.

Remarks

Some configuration changes require a restart of the system to make them active. The function *DivaSystemConfigurationActive* checks if a restart is pending to activate the configuration reported by *DivaGetSystemConfiguration*.

Older Dialogic® Diva® platforms do not provide the information about a pending restart. In this case, the function returns *DivaErrorNotSupported*. The function is a synchronous function and returns right away.

See also

DivaGetSystemConfiguration

DivaSetAnalogHookState

DivaSetAnalogHookState changes the hook state of an analog line.

```
DWORD     DivaSetAnalogHookState (     DWORD     LineDevice,
                                       DWORD     Line,
                                       BOOL      bOffHook );
```

Parameters

LineDevice

[in] The parameter *LineDevice* identifies the line device by an index starting with 1.

Line

[in] The parameter *Line* specifies the analog line of the line device by an index starting with 1.

bOffHook

[in] The parameter *bOffHook* specifies if the state should be set to on-hook or off-hook.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorNotSupported* or *DivaErrorInvalidParameter*.

Remarks

The function changes the hook state of the Dialogic® Diva® Analog Media Board. If the selected line device is not a Diva Analog Media Board, the function returns *DivaErrorNotSupported*. If the parameter *bOffHook* is true, the line is set to off-hook. There is no timer running in this mode. The application must place the line on-hook before any incoming call will be signaled. Note that outgoing calls can still be done. If the application places an outgoing call on a line that is set to off-hook, the hook state is switched to on-hook and back to off-hook before the dialing will start.

Note that this function is used to busy out the line at the switch. For call establishment use *DivaConnect* or *DivalDial*.

This function may change the configuration of the parameter "Call Direction". This parameter should only be used for systems that do not use the "Call Direction" configuration.

See also

DivaSetChannelStatus

Connection-oriented functions

This chapter contains various connection-oriented functions.

DivaCreateCall

The *DivaCreateCall* function creates a call object without initiating the dialing.

```
DWORD    DivaCreateCall (            DivaAppHandle        hApp,
                                     AppCallHandle        haCall
                                     DivaCallHandle       *phdCall );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application instance. The handle has been returned by *DivaRegister*.

haCall

[in] The *haCall* parameter specifies the application call handle. This handle is not interpreted by the Dialogic® Diva® SDK and is only used for event reporting.

phdCall

[out] The *phdCall* parameter points to a location where the Diva SDK call handle is placed. This handle must be used in all following calls to the Diva SDK for this call.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function creates a call object for outgoing calls with standard call parameters for voice connections. The application can modify the properties of the call by calling *DivaSetCallProperties*. The connection is initiated by calling *DivaDial*.

The function returns right away, independent from the event mode.

See also

DivaSetCallProperties, DivaCompleteCallTransfer, DivaGetCallProperties

DivaDial

DivaDial initiates dialing the given call object.

```
DWORD     DivaDial (       DivaCallHandle      hdCall,
                           Char                *DestinationNumber );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

DestinationNumber

[in] This parameter specifies the number to dial.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

If dialing the object has not yet started, a new call is set up with the stored information and the given number. If call setup has already started, the dialing information is sent in so-called overlap mode. This requires that the underlying switch supports overlap sending.

The function returns right away, and the call progress is reported via events.

See also

DivaSetupCallTransfer, DivaCompleteCallTransfer, DivaBlindCallTransfer

DivaListen

The *DivaListen* function registers for incoming calls on one or all line devices. The function is only valid if event reporting is selected.

```
DWORD       DivaListen (      DivaAppHandle        hApp,
                              DivaListenType       Services,
                              DWORD                LineDevice,
                              char                 *pCalledNumber );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application. The handle is assigned by a call to *DivaRegister*.

Services

[in] This parameter specifies which services should be signaled. Possible values are defined in DivaListenType. Multiple services can be combined.

LineDevice

[in] The *LineDevice* parameter specifies if incoming calls from all line devices or only from a specific line device should be signaled. To listen to all devices set this value to LINEDEV_ALL. Specific line defines are numbered from one to the maximum installed.

pCalledNumber

[in] Specification of the called number is optional. If it is specified, the signaled called number is compared to the given number for all incoming calls.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorLineDevice*, *DivaErrorInvalidHandle*, and *DivaErrorInvalidParameter*.

Remarks

Call this function to receive incoming calls. The function is a synchronous function and returns right away. Incoming calls are signaled by the event *DivaEventIncomingCall* to the event mechanism selected during registration.

The application can specify which calls it wants to receive, and from which line devices. To listen for all types of incoming calls, the *Services* parameter must be set to *DivaListenAll*. To listen for calls on all line devices, the *LineDevice* parameter must be set to LINEDEV_ALL.

If the *DivaListen* function differs for the various line devices or only incoming calls on certain line devices should be signaled, the application must specify the line device. In this case, a call to *DivaListen* is necessary for each line device.

Please note that whether the service of a call can be detected is dependent on the underlying switch environment. In particular, fax and voice calls may be signaled with the same service.

If the called number is specified, the Dialogic® Diva® API signals only calls that match this number. The numbers are compared from right to left up to the end of the shortest number. The question mark is allowed as wildcard and represents one digit. The Diva API also handles overlap receiving.

In addition to a single number, the application may set a list of numbers, e.g., several MSNs or a range of numbers, that the signaled number must match. Several numbers are separated by semicolons (1234;234;2345). A range is defined by the keywords "low" and "high" (LOW:1234000 HIGH:1234999).

See also

DivaRegister, DivaEventIncomingCall, DivaAlert, DivaAnswer, DivaListenType

DivaProceeding

DivaProceeding sends a proceeding message to the remote side.

```
DWORD       DivaProceeding (    DivaCallHandle       hdCall
                                AppCallHandle        haCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic$^{®}$ Diva$^{®}$ API. The handle is either returned by *DivaCreateCall*, *DivaConnect*, or is signaled with the event *DivaEventIncomingCall*.

haCall

[in] The *haCall* parameter identifies the call at application level. This handle is not interpreted by the Diva API, and it is only used for event reporting.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The function sends a proceeding message to the remote side. If UUI or Facility Data are specified, they are send with the message.

See also

DivaAlert

DivaAlert

DivaAlert sends an alert to the remote side.

```
DWORD       DivaAlert (     DivaCallHandle       hdCall
                            AppCallHandle        haCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic$^{®}$ Diva$^{®}$ API. The handle is signaled with the event *DivaEventIncomingCall*.

haCall

[in] The *haCall* parameter identifies the call at application level. This handle is not interpreted by the Diva API and it is only used for event reporting.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

None.

See also

DivaEventIncomingCall, DivaAnswer

DivaAttachToCall

The *DivaAttachToCall* function assigns an application-specific call handle to an incoming call in order to get additional event reporting for the call.

```
DWORD        DivaAttachToCall (        DivaCallHandle        hdCall
                                       AppCallHandle         haCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is provided with the event *DivaEventIncomingCall*.

haCall

[in] The *haCall* parameter identifies the call at application level. This handle is not interpreted by the Diva API, and only used for event reporting.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a call comes in, the application must call one of the following functions: DivaAnswer, DivaAlert, DivaAttachToCall, DivaReject.

If the application answers the call right away by calling *DivaAnswer* or proceeds with the call by *DivaAlert*, there is no need to call *DivaAttachToCall*. The application handle is then assigned by one of these functions.

Not all parameters of an incoming call may be available when the call is signaled for the first time. In direct dial-in environments, for example, the called number might be signaled digit by digit. If the application does not have enough information to decide whether a call should be accepted or rejected, it assigns its own call handle which is necessary for event reporting.

If the application wants to reject the call, it must call *DivaReject*.

See also

DivaEventIncomingCall

DivaAnswer

DivaAnswer answers an incoming call using the default settings set by the application with a call to *DivaSetLineDeviceParamsFax* or *DivaSetLineDeviceParamsVoice*.

```
DWORD        DivaAnswer (        DivaCallHandle        hdCall,
                                 AppCallHandle         haCall,
                                 DivaCallType          CallType );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is signaled with the event *DivaEventIncomingCall*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

CallType

[in] This parameter selects the call type to use, e.g., voice or fax. The values are defined in DivaCallType.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a call comes in, the application must call one of the following functions: DivaAnswer, DivaAlert, DivaAttachToCall, or DivaReject.

This function answers the call right away. It may take some time until the call is actually established.

The function returns right away, and the call progress is reported via events. The event *DivaEventCallConnected* signals that the data channel is ready for communication.

See also

DivaEventIncomingCall, DivaAnswerFax, DivaAnswerVoice, Call instance

DivaAnswerFax

DivaAnswerFax answers an incoming call with call type *DivaCallTypeFaxG3*.

```
DWORD      DivaAnswerFax (      DivaCallHandle      hdCall,
                                AppCallHandle       haCall,
                                char                *pLocalFaxId,
                                char                *pHeadLine,
                                DWORD               MaxSpeed,
                                DivaFaxOptions      OptionFlags );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Diva API. The handle is signaled with the event *DivaEventIncomingCall*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic$^{®}$ Diva$^{®}$ SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

pLocalFaxId

[in] This parameter specifies the fax station identification to be used as local identification. If *pLocalFaxId* is zero, the identification set by the application with a call to *DivaSetLineDeviceParamsFax* is used.

pHeadLine

[in] This parameter specifies a text that will be printed on top of every page. In addition to this information, the current date and time as well as the station identification and the current page are printed. If *pHeadLine* is zero, the headline set by the application with a call to *DivaSetLineDeviceParamsFax* is used.

MaxSpeed

[in] The *MaxSpeed* parameter specifies the maximum speed to be negotiated for the fax transmission.

OptionFlags

[in] The *OptionFlags* parameter specifies the fax options to be used, e.g., transmission mode as defined by DivaFaxOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a call comes in, the application must call one of the following functions: DivaAnswer, DivaAlert, DivaAttachToCall, DivaReject.

This function answers an incoming fax call right away without using the default parameters. It may take some seconds until the connection is established.

The function returns right away, and the call progress is reported via events. The event *DivaEventCallConnected* signals that the fax reception or sending can be started.

See also

DivaEventIncomingCall, DivaAnswer, DivaAnswerVoice, Call instance

DivaAnswerVoice

DivaAnswerVoice answers an incoming call with the call type *DivaCallTypeVoice*.

```
DWORD     DivaAnswerVoice (     DivaCallHandle      hdCall,
                                AppCallHandle       haCall,
                                DivaVoiceOptions    Options );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is signaled with the event *DivaEventIncomingCall*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

Options

[in] This parameter specifies the options to be used for this call. Valid options are defined in DivaVoiceOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a call comes in, the application must call one of the following functions: DivaAnswer, DivaAlert, DivaAttachToCall, or DivaReject.

The function answers an incoming voice call right away without using the default parameters. It may take several seconds until the connection is established.

The function returns right away, and the call progress is reported via events. The event *DivaEventCallConnected* signals that the data channel is available.

See also

DivaEventIncomingCall, DivaAnswer, DivaAnswerFax, Call instance

DivaAnswerVoIP (RTP)

DivaAnswerVoIP answers an incoming call using the call type *DivaCallTypeVoIP* for RTP streaming.

```
DWORD     DivaAnswerVoIP (      DivaCallHandle      hdCall,
                                AppCallHandle       haCall,
                                DivaVoIPParams      *pVoIPParams );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or signaled with the event *DivaEventIncomingCall*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

pVoIPParams

[in] The *pVoIPParams* parameter is a pointer to a user-supplied buffer of the type *DivaVoIPParams* that defines VoIP-specific parameters. For detailed information on the parameters, see DivaVoIPParams.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a call comes in, the application must call one of the following functions: DivaAnswer, DivaAlert, DivaAttachToCall, or DivaReject.

The function answers an incoming voice call right away without using the default parameters defined by *DivaVoIPParams*.

Note that the call type *DivaCallTypeVoIP* specifies that RTP packets should be used in the data channel. This call type does not initiate a call on the IP network via SIP or H.323. This call type is only available on Dialogic® Diva® Media Boards.

The function returns right away, and the call progress is reported via events. The event *DivaEventCallConnected* signals that the data channel is available.

See also

DivaEventIncomingCall, DivaAnswer, DivaAnswerFax, DivaAnswerVoice, DivaAnswerModem, Call instance

DivaAnswerModem

DivaAnswerModem answers an incoming call using the call type *DivaCallTypeModem*.

```
DWORD     DivaAnswerModem (     DivaCallHandle      hdCall,
                                AppCallHandle       haCall,
                                DWORD               MaxSpeed,
                                DWORD               ModemOptions );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or signaled with the event *DivaEventIncomingCall*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

MaxSpeed

[in] The *MaxSpeed* parameter defines the maximum speed that should be negotiated.

ModemOptions

[in] The *ModemOptions* parameter defines the modem options to be used for connection establishment. Valid options are defined in DivaModemOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a call comes in, the application must call one of the following functions: DivaAnswer, DivaAlert, DivaAttachToCall, or DivaReject.

The function answers an incoming analog modem call right away. The modem speed and protocol, e.g., compression, are negotiated according to the *MaxSpeed* and *ModemOptions* parameters.

Applications that want to reduce the call setup time and only need a very low speed may set *MaxSpeed* to a low value, e.g., 2400. Additional modem parameters can be set using call properties. Please refer to DivaSetCallProperties for more information.

The function returns right away, and the *DivaEventCallConnected* event is sent when connection establishment is completed.

See also

DivaEventIncomingCall, DivaAnswer, DivaAnswerFax, DivaAnswerVoice, DivaAnswerModem, Call instance

DivaAnswerSMS

DivaAnswerSMS answers an incoming call using the call type *DivaCallTypeSMS*.

```
DWORD     DivaAnswerSMS (      DivaCallHandle       hdCall,
                               AppCallHandle        haCall,
                               DivaSMSProtocol      Protocol );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is signaled with the event *DivaEventIncomingCall*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

Protocol

[in] This parameter selects the protocol to use. The values are defined in DivaCallType.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a call comes in, the application must call one of the following functions: DivaAnswer, DivaAlert, DivaAttachToCall, or DivaReject.

This function answers the call right away. It may take some time until the call is actually established.

The function returns right away, and the call progress is reported via events. The event *DivaEventCallConnected* signals that the data channel is ready for communication.

See also

DivaEventIncomingCall, DivaAnswer, DivaAnswerFax, Call instance

DivaReject

The *DivaReject* function tells the Dialogic® Diva® API that the application is not interested in a call.

```
DWORD      DivaReject (     DivaCallHandle      hdCall
                            BOOL                bOtherMayTakeIt );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Diva API. The handle is signaled with the event *DivaEventIncomingCall*.

bOthersMayTakeIt

[in] The *bOthersMayTakeIt* parameter defines if other applications may answer this call. See remarks below.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a call comes in, the application must call one of the following functions: DivaAnswer, DivaAlert, DivaAttachToCall, or DivaReject.

An application may decide that an incoming call should not be taken. The application calls *DivaReject* in order to tell the Diva API that the call is not serviced and how to proceed with the call.

Incoming calls are signaled to all applications that have an active listen matching the call type and the optional called number. If the application wants to allow other applications to service this call, it sets the *bOtherMayTakeIt* parameter to TRUE. In this case, the Diva API does not reject the physical call right away, and other applications have the chance to answer this call.

If the application decides that the call should be rejected and disconnected right away, the *bOtherMayTakeIt* parameter must be set to FALSE.

The application may set a specific reason for rejecting the call via the call property *DivaCPT_RejectReason*.

See also

DivaEventIncomingCall, DivaAttachToCall

DivaConnect

DivaConnect initiates the establishment of an outgoing call using the default parameters set by the application with a call to *DivaSetLineDeviceParamsVoice* or *DivaSetLineDeviceParamsFax*.

```
DWORD    DivaConnect (   DivaAppHandle    hApp,
                         AppCallHandle    haCall,
                         DivaCallHandle   *phdCall,
                         char             *DestinationNumber,
                         DivaCallType     CallType,
                         DWORD            LineDevice );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application. The handle is assigned by a call to *DivaRegister*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

phdCall

[out] The *phdCall* parameter points to a location of the type *DivaCallHandle* that receives the call handle on successful return.

DestinationNumber

[in] This parameter specifies the number to dial.

CallType

[in] This parameter selects the call type to use, e.g., voice or fax. Possible values are defined in DivaCallType.

LineDevice

[in] This parameter specifies the line device that should be used for the call. If the parameter is set to LINEDEV_ALL, the Dialogic® Diva® API searches for a free resource on all installed line devices.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorDestBusy*, *DivaErrorNoAnswer*, *DivaErrorNoChannel*, *DivaErrorLineDevice*, *DivaErrorInvalidHandle*, and *DivaErrorInvalidParameter*.

Remarks

The function initiates the connection using the default parameter for the specified call type. The function returns right away. If the result is *DivaSuccess*, the connection has been initiated and the progress is reported via events. Once the event *DivaEventCallConnected* is signaled, the data channel is available.

See also

DivaRegister, DivaEventCallConnected, DivaEventCallProgress, DivaDisconnect, DivaConnectFax, DivaConnectVoice, Call instance

DivaConnectFax

DivaConnectFax initiates the establishment of an outgoing call with call type *DivaCallTypeFaxG3*.

```
DWORD     DivaConnectFax (     DivaAppHandle        hApp,
                               AppCallHandle        haCall,
                               DivaCallHandle       *phdCall,
                               char                 *DestinationNumber,
                               DWORD                LineDevice,
                               char                 *LocalNumber,
                               char                 *LocalSubAddress,
                               char                 *pLocalFaxId,
                               char                 *pHeadLine,
                               DWORD                MaxSpeed,
                               DivaFaxOptions       OptionFlags );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application. The handle is assigned by a call to *DivaRegister*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

phdCall

[out] This parameter points to a location of the type *DivaCallHandle* that receives the call handle on successful return.

DestinationNumber

[in] This parameter specifies the number to dial.

LineDevice

[in] The *LineDevice* parameter specifies which line device should be used for the call. If the parameter is set to LINEDEV_ALL, the Diva API searches for a free resource on all installed line devices.

LocalNumber

[in] The *LocalNumber* parameter specifies which number should be signaled as the calling number. If *LocalNumber* is zero, the number set by the application with a call to *DivaSetLineDeviceParamsFax* or *DivaSetLineDeviceVoice* is used.

LocalSubAddress

[in] This parameter specifies which number should be signaled as the calling subaddress. If *LocalSubAddress* is zero, the number set by the application with a call to *DivaSetLineDeviceParamsFax* or *DivaSetLineDeviceParamsVoice* is used.

pLocalFaxId

[in] This parameter specifies the fax station identification to be used as the local identification. If *pLocalFaxId* is zero, the identification set by the application with a call to *DivaSetLineDeviceParamsFax* is used.

pHeadLine

[in] The *pHeadLine* parameter specifies a text that is printed on top of every page. In addition to this information, the current date and time as well as the station identification and the current page are given. If *pHeadLine* is zero, the headline set by the application with a call to *DivaSetLineDeviceParamsFax* is used.

MaxSpeed

[in] This parameter specifies the maximum speed to be negotiated for the fax transmission.

OptionFlags

[in] This parameter specifies the fax options to be used, e.g., transmission mode as defined by DivaFaxOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorDestBusy*, *DivaErrorNoAnswer*, *DivaErrorNoChannel*, *DivaErrorLineDevice*, *DivaErrorInvalidHandle*, and *DivaErrorInvalidParameter*.

Remarks

The function initiates a fax connection using the given fax parameter. The function returns right away. If the result is *DivaSuccess*, the connection has been initiated and the progress is reported via events. Once the event *DivaEventCallConnected* is signaled, the fax exchange must be initiated via *DivaSendFax* or polling mode is negotiated via *DivaReceiveFax*.

See also

DivaRegister, DivaEventCallConnected, DivaEventCallProgress, DivaDisconnect, DivaConnect, DivaConnectVoice, Call instance

DivaConnectVoice

DivaConnectVoice initiates the establishment of an outgoing call with the call type *DivaCallTypeVoice*.

```
DWORD      DivaConnectVoice (    DivaAppHandle        hApp,
                                 AppCallHandle        haCall,
                                 DivaCallHandle       *phdCall,
                                 char                 *DestinationNumber,
                                 DWORD                LineDevice,
                                 char                 *LocalNumber,
                                 char                 *LocalSubAddress,
                                 DivaVoiceOptions     Options );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application. The handle is assigned by a call to *DivaRegister*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

phdCall

[out] This parameter points to a location of the type *DivaCallHandle* that receives the call handle on successful return.

DestinationNumber

[in] This parameter specifies the number to dial.

LineDevice

[in] The *LineDevice* parameter specifies which line device should be used for the call. If the parameter is set to LINEDEV_ALL, the Dialogic® Diva® API searches for a free resource on all installed line devices.

LocalNumber

[in] The *LocalNumber* parameter specifies the number that should be signaled as the calling number. If *LocalNumber* is (0) zero, the number set by the application with a call to *DivaSetLineDeviceParamsVoice* or *DivaSetLineDeviceParamsFax* is used.

LocalSubAddress

[in] This parameter specifies the number that should be signaled as the calling subaddress. If *LocalSubAddress* is zero, the number set by the application with a call to *DivaSetLineDeviceParamsVoice* or *DivaSetLineDeviceParmsFax* is used.

Options

[in] This parameter specifies the voice options that should be used for a call. For valid options see DivaVoiceOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorDestBusy*, *DivaErrorNoAnswer*, *DivaErrorNoChannel*, *DivaErrorLineDevice*, *DivaErrorInvalidHandle*, and *DivaErrorInvalidParameter*.

Remarks

The function initiates the connection for a voice call using the given parameters. The function returns right away. If the result is *DivaSuccess*, the connection has been initiated and the progress is reported via events.

The availability of the data channel is reported via the event *DivaEventCallConnected* or in early data channel mode via *DivaEventEarlyDataChannelConnected*.

See also

DivaRegister, DivaEventCallConnected, DivaEventCallProgress, DivaDisconnect, DivaConnectFax, DivaConnect, Call instance

DivaConnectVoIP

DivaConnectVoIP initiates the establishment of an outgoing call with the call type *DivaCallTypeVoIP*.

```
DWORD     DivaConnectVoIP (     DivaAppHandle         hApp,
                                AppCallHandle         haCall,
                                DivaCallHandle        *phdCall,
                                char                  *DestinationNumber,
                                DWORD                 LineDevice,
                                char                  *LocalNumber,
                                char                  *LocalSubAddress,
                                DivaVoIPParams        *pVoIPParams );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application. The handle is assigned by a call to *DivaRegister*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

phdCall

[out] This parameter points to a location of the type *DivaCallHandle* that receives the call handle on successful return.

DestinationNumber

[in] This parameter specifies the number to dial.

LineDevice

[in] The *LineDevice* parameter specifies which line device should be used for the call. If the parameter is set to LINEDEV_ALL, the Dialogic® Diva® API searches for a free resource on all installed line devices.

LocalNumber

[in] This parameter specifies which number should be signaled as the calling number. If *LocalNumber* is empty, the number set by the application with a call to *DivaSetLineDeviceParams* is used.

LocalSubAddress

[in] This parameter specifies which number should be signaled as the calling subaddress. If *LocalSubAddress* is empty, the subaddress set by the application with a call to *DivaSetLineDeviceParams* is used.

pVoIPParams

[in] The p*VoIPParams* parameter is a user-supplied buffer of the type *DivaVoIPParams* that defines VoIP-specific parameters. For detailed information on the parameters, see DivaVoIPParams.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function establishes an analog connection and prepares the data channel for RTP streaming. The parameters for RTP streaming and additional functions, such as silence suppression, are set by DivaVoIPParams.

The function returns right away, and the event *DivaEventCallConnected* is sent when connection establishment is completed.

See also

DivaRegister, DivaEventCallConnected, DivaEventCallProgress, DivaDisconnect, DivaConnectFax, DivaConnectVoice, Call instance

DivaConnectModem

DivaConnectModem initiates the establishment of an outgoing call with the call type *DivaCallTypeModem.*

```
DWORD      DivaConnectModem (      DivaAppHandle          hApp,
                                   AppCallHandle          haCall,
                                   DivaCallHandle         *phdCall,
                                   char                   *DestinationNumber,
                                   DWORD                  LineDevice,
                                   char                   *LocalNumber,
                                   char                   *LocalSubAddress,
                                   DWORD                  MaxSpeed,
                                   DivaModemOptions       Options );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application. The handle is assigned by a call to *DivaRegister*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

phdCall

[out] This parameter points to a location of the type *DivaCallHandle* that receives the call handle on successful return.

DestinationNumber

[in] This parameter specifies the number to dial.

LineDevice

[in] The *LineDevice* parameter specifies which line device should be used for the call. If the parameter is set to LINEDEV_ALL, the SDK searches for a free resource on all installed line devices.

LocalNumber

[in] The *LocalNumber* parameter specifies which number should be signaled as the calling number. If *LocalNumber* is empty, the number set by the application with a call to *DivaSetLineDeviceParams* is used.

LocalSubAddress

[in] This parameter specifies which number should be signaled as the calling subaddress. If *LocalSubAddress* is empty, the subaddress set by the application with a call to *DivaSetLineDeviceParams* is used.

MaxSpeed

[in] The *MaxSpeed* parameter defines the maximum speed that should be negotiated.

Options

[in] The *Options* parameter defines the modem options to be used for connection establishment. Valid options are defined in DivaModemOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function establishes an analog modem connection. The modem speed and protocol, e.g., compression, is negotiated depending on the *MaxSpeed* and *Options* parameters.

Applications that want to reduce the call setup time and only need a very low speed may set *MaxSpeed* to a low value, e.g., 2400.

The function returns right away, and the *DivaEventCallConnected* event is sent when connection establishment is completed.

See also

DivaRegister, DivaEventCallConnected, DivaEventCallProgress, DivaDisconnect, DivaConnectFax, DivaConnectVoice, Call instance

DivaConnectSMS

DivaConnectSMS initiates the establishment of an outgoing call using the default parameters set by the application with a call to *DivaSetLineDeviceParamsVoice* or *DivaSetLineDeviceParamsFax*.

```
DWORD       DivaConnectSMS (    DivaAppHandle       hApp,
                                AppCallHandle       haCall,
                                DivaCallHandle      *phdCall,
                                char                *DestinationNumber,
                                DWORD               LineDevice,
                                char                *LocalNumber,
                                DivaSMSProtocol     Protocol );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application. The handle is assigned by a call to *DivaRegister*.

haCall

[in] The value of *haCall* that you pass in here will be given back to your application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass (for example) an index or a pointer to a structure, to help you keep track of multiple calls in the same application.

phdCall

[out] The *phdCall* parameter points to a location of the type *DivaCallHandle* that receives the call handle on successful return.

DestinationNumber

[in] This parameter specifies the number to dial.

LineDevice

[in] This parameter specifies the line device that should be used for the call. If the parameter is set to LINEDEV_ALL, the Dialogic® Diva® API searches for a free resource on all installed line devices.

LocalNumber

[in] This parameter specifies which number should be signaled as the calling number. If *LocalNumber* is empty, the number set by the application with a call to *DivaSetLineDeviceParams* is used.

Protocol

[in] This parameter selects the protocol to use. The values are defined in DivaCallType.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorDestBusy*, *DivaErrorNoAnswer*, *DivaErrorNoChannel*, *DivaErrorLineDevice*, *DivaErrorInvalidHandle*, and *DivaErrorInvalidParameter*.

Remarks

The function initiates a modem connection for SMS over fixed networks. The function returns right away. If the result code is *DivaSuccess*, the connection has been initiated and the progress is reported via events. Once the event *DivaEventCallConnected* is signaled the application can start to pass layer 3 SMS messages.

See also

DivaRegister, DivaEventCallConnected, DivaEventCallProgress, DivaDisconnect, Call instance

DivaSetCallType

DivaSetCallType changes the type of an already established call.

```
DWORD        DivaSetCallType (   DivaCallHandle      hdCall,
                                 DivaCallType        CallType );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

CallType

[in] This parameter selects the call type to use, e.g., voice or fax. Possible values are defined in DivaCallType.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function changes the call type for an existing connection. The connection remains stable, only the data channel is disconnected and re-established with the new call type.

The function returns right away, and a new event *DivaEventCallConnected* is sent when the data channel is up again and data can be sent and received.

Note: This function is not available for all call types. If either the active call type or the target call type does not support changing the call type, the function returns *DivaErrorInvalidParameter*.

See also

DivaSetCallTypeFax, DivaSetCallTypeVoice, DivaSetCallTypeVoIP

DivaSetCallTypeFax

DivaSetCallTypeFax changes the type of the call to *DivaCallTypeFaxG3*.

```
DWORD     DivaSetCallTypeFax (     DivaCallHandle        hdCall,
                                   char                  *pLocalFaxId,
                                   char                  *pHeadLine,
                                   DivaFaxMaxSpeed       MaxSpeed,
                                   DivaFaxOptions        OptionFlags );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pLocalFaxId

[in] This parameter specifies the fax station identification to be used as local identification. If *pLocalFaxId* is zero, the identification set by the application with a call to *DivaSetLineDeviceParamsFax* is used.

pHeadline

[in] The *pHeadline* parameter specifies a text that is printed on top of every page. In addition to this information, the current date and time, as well as the station identification and the current page are given. If *pHeadLine* is zero, the headline set by the application with a call to *DivaSetLineDeviceParamsFax* is used.

MaxSpeed

[in] This parameter specifies the maximum speed to be negotiated for the fax transmission.

OptionFlags

[in] The *OptionFlags* parameter specifies the fax options to be used, e.g., transmission mode as defined by DivaFaxOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function changes the call type to Fax G3 for a call that is already connected. The existing connection remains stable, only the data channel is disconnected and reconnected with the call type Fax G3 using the given parameters.

The function returns right away, and a new event *DivaEventCallConnected* is sent when the data channel is up again and fax files can be sent or received.

The fax protocol depends on the call direction. In general, the fax protocol itself supports changing the call direction internally, which is called fax polling. In case of a protocol change in the data channel, the basic call direction for the fax protocol can also be changed. This is supported by the fax options.

See also

DivaSetCallType, DivaSetCallTypeVoice, DivaSetCallTypeVoIP

DivaSetCallTypeVoice

DivaSetCallTypeVoice changes the call type to voice, using the default parameters that were set by the application with a call to *DivaSetLineDeviceParamsVoice*.

```
DWORD     DivaSetCallTypeVoice (    DivaCallHandle        hdCall,
                                    DivaVoiceOptions      Options );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Options

[in] This parameter specifies the options that should be used for this call. For valid options see DivaVoiceOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function changes the call type to voice for a call that is already connected. The existing connection remains stable, only the data channel is disconnected and reconnected with the call type voice using the given voice options.

The function returns right away, and the *DivaEventCallConnected* event is sent when the data channel is up again and audio data can be sent and received.

See also

DivaSetCallType, DivaSetCallTypeFax, DivaSetCallTypeVoIP

DivaSetCallTypeVoIP

DivaSetCallTypeVoIP changes the type of the call to voice using RTP streaming.

```
DWORD     DivaSetCallTypeVoIP (     DivaCallHandle     hdCall,
                                    DivaVoIPParams     *pVoIPParams );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pVoIPParams

[in] This parameter is a pointer to a user-supplied buffer of the type *DivaVoIPParams* that defines VoIP-specific parameters. For detailed information on the parameters, see DivaVoIPParams.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function changes the call type to voice for a call that is already connected. The existing connection remains stable. Only the data channel is disconnected and reconnected for RTP streaming with the given payload protocol and options.

The function returns right away, and a new event *DivaEventCallConnected* is sent when the data channel is up again and audio data can be sent and received.

See also

DivaSetCallType, DivaSetCallTypeFax, DivaSetCallTypeVoice

DivaDisconnect

DivaDisconnect disconnects a call.

```
DWORD      DivaDisconnect (    DivaCallHandle        hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The function initiates the disconnection of a call.

The function returns right away. If the return parameter is *DivaSuccess*, the disconnect is reported via the event *DivaEventCallDisconnected*.

Note: If the function does not return *DivaSuccess*, no event is signaled.

See also

DivaConnect, DivaAnswer, DivaCloseCall

DivaGetCallInfo

DivaGetCallInfo retrieves information about the call.

```
DWORD          DivaGetCallInfo (    DivaCallHandle        hdCall,
                                    DivaCallInfo          *pCallInfo );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pCallInfo

[out] This parameter is a pointer to a user-supplied buffer of the type DivaCallInfo that receives the information on the call. Note that the application must set the *Size* field of the *DivaCallInfo* structure to the size of the structure before calling this function.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

For further information on the provided information see the description of DivaCallInfo.

See also

DivaCallInfo

DivaCloseCall

DivaCloseCall releases a call instance at the Dialogic® Diva® API.

```
DWORD          DivaCloseCall (    DivaCallHandle          hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Diva API. The handle is returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). If the call handle is not known, the return value is *DivaErrorInvalidHandle*.

Remarks

This function frees a call instance at the Diva API. In order to give the application the opportunity to retrieve information on the disconnect reason, the Diva API keeps the call after *DivaEventCallDisconnected* has been signaled. Therefore, the application must close the call by calling *DivaCloseCall*. Note that this is not necessary for calls that are rejected by calling *DivaReject*.

See also

DivaAnswer

DivaEnableAnsweringMachineDetector

DivaEnableAnsweringMachineDetector starts the detection process based on the length of prompt based answering machine detection.

```
DWORD     DivaEnableAnsweringMachineDetector (      DivaCallHandle     hdCall,
                                                    DWORD              MaxInitialsilence,
                                                    DWORD              MaxHumanSpeakerTime,
                                                    DWORD              MaxInterSpeakerTimeout );
```

Parameters

hdCall

[in] The *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect*.

MaxInitialSilence

[in] The parameter *MaxInitialSilence* specifies the time, in milliseconds, until the remote side starts speaking. When this timeout is reached without detecting a speaker, the answering machine detector terminates with the result *DivaResultSilence*.

MaxHumanSpeakerTime

[in] The parameter *MaxHumanSpeakerTime* specifies the time, in milliseconds, that is seen as the maximum time a human speaker would speak when answering the phone. If the announcement from the called party is longer, this will be interpreted as an answering machine.

MaxInterSpeakerTimeout

[in] The parameter *MaxInterSpeakerTimeout* specifies the maximum time, in milliseconds, the human speech is interrupted after it has started to be interpreted as continuous speech.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaEnableAnsweringMachineDetector* the application enables the analyses of the inbound audio stream for answering machine detection based on the length of the speech. The result is reported via the event *DivaEventAnsweringMachineDetector*.

The Dialogic® Diva® SDK compares the length of the received speech with the given parameter. If the length of the announcements is below *MaxHumanSpeakerTime,* a human has answered the phone. If the length is above *MaxHumanSpeakerTime,* an automated system has answered.

If no signal is received, the detector terminates when the *MaxInitialSilence* is reached.

Note: The answering machine detector requires detection capabilities not available on all Dialogic® communication platforms. The application may check for the extended voice capabilities of a line device. The detection must be enabled by the application using *DivaReportTones*.

See also

DivaEventAnsweringMachineDetector, DivaDisableAnsweringMachineDetector, DivaEventAnsweringMachineDetector

DivaDisableAnsweringMachineDetector

DivaDisableAnsweringMachineDetector stops the answering machine detector.

```
DWORD     DivaDisableAnsweringMachineDetector (        DivaCallHandle       hdCall );
```

Parameters

hdCall

[in] The *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaDisableAnsweringMachineDetector* the application stops a previously started detector. The stopping is confirmed with an event *DivaEventAnsweringMachineDetector* with the result set to *DivaResultUserTerminated*.

See also

DivaEnableAnsweringMachineDetector

Data transfer functions

This chapter contains the following data transfer functions.

DivaSendData

DivaSendData sends the given data to the remote side.

```
DWORD     DivaSendData (     DivaCallHandle     hdCall,
                             unsigned char      *pData,
                             DWORD              DataLength,
                             DWORD              DataHandle );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pData

[in] The *pData* parameter points to a buffer provided by the caller. This buffer contains the data to be sent.

DataLength

[in] The *DataLength* parameter specifies the amount of data in the buffer pointed to by *pData*.

DataHandle

[in] The *DataHandle* parameter is an optional value to be used for confirmation. See remarks below.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

If the length of the data buffer exceeds the maximum buffer size set by *DivaRegister*, the data is fragmented. Depending on the used protocol, the data may be received in fragments at the remote side.

In order to avoid copying data, the data buffer is owned by the Diva API until it is free. The application receives the event *DivaEventDataSent* when the data has been sent and the buffer can be reused by the application. The buffer is identified by the *DataHandle*.

See also

DivaReceiveData, DivaEventDataSent

DivaReceiveData

DivaReceiveData obtains received data from the Dialogic® Diva® API.

```
DWORD     DivaReceiveData (  DivaCallHandle     hdCall,
                             unsigned char      **pData,
                             DWORD              BufferSize
                             DWORD              *pDataLength );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Diva API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pData

[out] The *pData* parameter points to a buffer that receives the data.

BufferSize

[out] The *BufferSize* parameter specifies the length of the data buffer in bytes.

pDataLength

[out] The *pDataLength* parameter points to a location that receives the amount of bytes copied to the buffer.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

If data is available, it is copied to the buffer provided by the application. The Diva API signals the event *DivaEventDataAvailable* when data is available. The amount of data is passed with the event. The application may retrieve the data using *DivaReceiveData*.

Note: New data is only signaled if the application retrieves the data.

See also

DivaSendData

DivaSendFrame

DivaSendFrame sends the given data and data options as a frame to the remote side.

```
DWORD       DivaSendFrame (   DivaCallHandle        hdCall,
                              unsigned char         pData,
                              DWORD                 DataLength
                              DWORD                 DataHandle
                              DWORD                 DataOptions );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall*, *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pData

[in] The *pData* parameter points to a buffer provided by the caller. This buffer contains the data to be sent.

DataLength

[in] The *DataLength* parameter specifies the amount of data in the buffer pointed to *pData*.

DataHandle

[in] The *DataHandle* parameter is an optional value to be used for confirmation. See remarks below.

DataOptions

[in] The *DataOptions* parameter specifies the options to be signaled with the frame. For valid options refer to DivaDataOptions.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, or *DivaErrorInvalidHandle*.

Remarks

The function sends the given data as a frame to the remote side. To avoid fragmentation, the buffer size must not exceed the data size specified in DivaRegister.

The function is only supported on call types that allow framing, e.g., digital call types and modem call types with a layer 2 protocol.

In order to avoid copying data, the data buffer is owned by the Diva API until it is free. The application receives the event *DivaEventDataSent* when the data has been sent and the buffer can be reused by the application. The buffer is identified by the *DataHandle*.

The options for the frame can be transferred to the remote side if supported by the underlying protocol. The options are defined in DivaDataOptions.

See also

DivaSendData, DivaReceiveData, DivaReceiveFrame

DivaReceiveFrame

DivaReceiveFrame obtains the received framed data and the data options.

```
DWORD       DivaReceiveFrame (    DivaCallHandle      hdCall,
                                  unsigned char*      pData,
                                  DWORD               BufferSize,
                                  DWORD*              pDataLength,
                                  DWORD*              DataOptions );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall*, *DivaConnect*, or signaled with the event *DivaEventIncomingCall*.

pData

[out] The *pData* parameter points to a buffer provided by the caller. This buffer contains the data to be received.

BufferSize

[in] The *BufferSize* parameter specifies the length of the data buffer in bytes.

pDataLength

[out] The *pDataLength* parameter points to a location that receives the amount of bytes copied to the buffer.

pDataOptions

[out] The *pDataOptions* parameter points to a location that receives the data options for the frame.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, or *DivaErrorInvalidHandle*.

Remarks

If data is available, it is copied to the buffer provided by the application. The data options for the frame are also provided to the application.

The availability of a data frame is signaled to the application via the event *DivaEventDataAvailable*. The amount of available data is signaled with the event. If more than one frame is available, this amount of data is the complete size of all frames.

See also

DivaSendData, DivaReceiveData, DivaSendFrame

Fax transfer functions

This chapter contains various fax transfer functions.

DivaSendFax

DivaSendFax sends a fax given in a file.

```
DWORD     DivaSendFax (  DivaCallHandle    hdCall,
                         char              *pFilename,
                         DivaFaxFormat     Format = DivaFaxFormatTIFF_ClassF );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pFilename

[in] The *pFilename* parameter points to the filename of the file containing the data to be sent. It must be a complete path and file name. The process context of the caller must have read access rights for this file.

Format

[in] The *Format* parameter specifies the format in which the data is stored. For supported fax formats, see DivaFaxFormat.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorOpenFile*, *DivaErrorReadFile*, *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

The function opens the given file, converts the data if necessary, and sends them as fax stream to the remote side. For supported fax formats, see DivaFaxFormat. For each transmitted page, the event *DivaEventFaxPageSent* is signaled. When the fax is successfully sent to the remote end, the event *DivaEventFaxSent* is signaled. If the application did not receive the event *DivaEventFaxSent* before the event *DivaEventCallDisconnected* is signaled, an error is indicated. Details on the reason can be retrieved via *DivaGetCallInfo*.

The function can only be used if the *DivaFaxOptionMultipleDocuments* is not set. If the application has enabled the option before initiating the call, *DivaSendFax* returns *DivaErrorInvalidFunction*.

See also

DivaEventFaxPageSent, DivaFaxFormat, DivaReceiveFax

DivaSendMultipleFaxFiles

DivaSendMultipleFaxFiles sends multiple fax documents.

```
DWORD     DivaSendMultipleFaxFiles (   DivaCallHandle     hdCall,
                                       Int                NumFiles,
                                       char               **ppFileArray,
                                       DivaFaxFormat      Format );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

NumFiles

[in] The *NumFiles* parameter identifies the number of entries in *ppFileArray*.

ppFileArray

[in] The *ppFileArray* parameter points to an array of pointers to the documents.

Format

[in] The *Format* parameter specifies in which format the data is available in the files. Note that all files must contain data of the same format.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorOpenFile*, *DivaErrorReadFile*, *DivaErrorInvalidParameter*, *DivaErrorInvalidState,* and *DivaErrorInvalidHandle*.

Remarks

The function sends multiple files. Each file may contain several pages, depending on the option *DivaFaxOptionMultipleDocument*.

If the option has been set during connect establishment in its call to *DivaConnectFax* or *DivaAnswerFax*, each file is sent as a single document. The application receives the event *DivaEventFaxDocumentSent* after each file has been sent and the event *DivaEventFaxSent* after all files have been sent. The number information added to the headline start with one for each file in the list.

If the option is not set, which is only supported for TIFF files, all files are interpreted as one document. The application receives only one event *DivaEventFaxSent* after the last page of the last file has been sent. The number information added to the headline are consecutive for all pages of all files.

See also

DivaSendFax, DivaConnectFax, DivaAnswerFax, DivaSetCallTypeFax

DivaReceiveFax

DivaReceiveFax receives a fax and stores it in a given format in a given file.

```
DWORD    DivaReceiveFax (   DivaCallHandle    hdCall,
                            char              *pFilename,
                            DivaFaxFormat     Format = DivaFaxFormatTIFF_ClassF );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pFilename

[in] The *pFilename* parameter points to the filename of the file where the received fax will be stored. It must be a complete path and file name. The process context of the call must have create and write access to this directory and file.

Format

[in] The *Format* parameter specifies the format in which data is stored.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorCreateFile*, *DivaErrorWriteFile*, *DivaErrorInvalidParameter*, *DivaErrorInvalidState,* and *DivaErrorInvalidHandle*.

Remarks

The function converts the received fax to the requested data format and creates the file. Please note that existing files will be overwritten.

Available formats are specified by DivaFaxFormat. Text format is not available. If the application has registered for event reporting, the event *DivaEventFaxReceived* is signaled to the application.

See also

DivaEventFaxPageReceived, DivaFaxFormat, DivaSendFax, DivaReceiveFax

DivaAppendFax

DivaAppendFax appends the given fax document to an existing fax transmission.

```
DWORD     DivaAppendFax (     DivaCallHandle     hdCall,
                              char               *pFilename,
                              DivaFaxFortmat     Format,
                              BOOL               bNewDocument );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pFilename

[in] The *pFilename* parameter points to the filename of the file containing the data to be sent. It must be a complete path and file name. The process context of the caller must have read access rights for this file.

Format

[out] The *Format* parameter specifies the format in which the data is stored. For supported fax formats, see DivaFaxFormat.

bNewDocument

[in] The parameter *bNewDocument* specifies if the given files should be send as part of the current document or as a new document on the same connection.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidState*, *DivaErrorLimitExceeded*, *DivaErrorInvalidParameter*, and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaAppendFax* the application adds fax pages to a running fax transmission. Adding fax pages to a running transmission is only possible if the last page is still to be processed. If the last page is already in progress and no new pages can be added, the function returns *DivaErrorInvalidState.*

With the function *DivaSendMultipleFaxFiles* the Diva API supports the sending of one fax document per TIFF file or several TIFF files as one fax document. The sending mode depends on the FaxOptions specified by the application during call establishment. For more information, see *DivaFaxOptionMultipleDocument*.

When adding a TIFF file to an existing transmission, the application can specify if the pages in the new TIFF file should be appended to the last document or if they should be sent as a new document.

If the parameter *bNewDocument* is set to FALSE, the pages in the given document are added to the last document of the current transmission. If set to TRUE, the pages in the file are signaled as a separate fax document. In both cases, the pages are sent on the same logical connection.

DivaAppendFax can not be used for the format *DivaFaxFormatColorJPEG*.

See also

DivaSendFax, DivaSendMultipleFaxFiles, DivaAppendFaxFiles

DivaAppendFaxFiles

DivaAppendFaxFiles appends the given fax documents to an existing fax transmission.

```
DWORD    DivaAppendFaxFiles (    DivaCallHandle    hdCall,
                                 int               NumFiles,
                                 char              **ppFileArray,
                                 DivaFaxFortmat    Format,
                                 BOOL              bNewDocument );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

NumFiles

[in] The *NumFiles* parameter identifies the number of entries in *ppFileArray*.

ppFileArray

[in] The *ppFileArray* parameter points to an array of pointers to the documents.

Format

[out] The *Format* parameter specifies in which format the data is available in the files. Note that all files must contain data of the same format.

bNewDocument

[in] The *bNewDocument* parameter specifies if the given files should be sent as part of the current document or as a new document on the same connection.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidState*, *DivaErrorLimitExceeded*, *DivaErrorInvalidParameter*, and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaAppendFaxFiles* the application adds fax pages to a running fax transmission. Adding fax pages to a running transmission is only possible if the last page is still to be processed. If the last page is already in progress and no new pages can be added, the function returns *DivaErrorInvalidState.*

With the function *DivaSendMultipleFaxFiles* the Diva API supports the sending of one fax document per TIFF file or several TIFF files as one fax document. The sending mode depends on the FaxOptions specified by the application during call establishment. For more information, see *DivaFaxOptionMultipleDocument*.

When adding a TIFF file to an existing transmission, the application can specify if the pages in the new TIFF file should be appended to the last document or if they should be sent as a new document.

If the parameter *bNewDocument* is set to FALSE, the pages in the given document are added to the last document of the current transmission. If set to TRUE, the pages in the file are signaled as a separate fax document. In both cases, the pages are sent on the same logical connection.

DivaAppendFax cannot be used for the format *DivaFaxFormatColorJPEG*.

See also

DivaSendFax, DivaSendMultipleFaxFiles, DivaAppendFax

DivaReceiveFaxToMemory

DivaReceiveFaxToMemory initiates the memory based fax reception in the given format.

```
DWORD    DivaReceiveFaxToMemory (    DivaCallHandle    hdCall,
                                     DivaFaxFormat     Format );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall*, *DivaConnect*, or signaled with the event *DivaEventIncomingCall*.

Format

[in] The *Format* parameter specifies the data format in which the data is provided in the memory.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidState*, *DivaErrorInvalidParameter*, and *DivaErrorInvalidHandle*.

Remarks

The fax reception to the memory is available for TIFF documents only. The fax data is provided on a per page base, because the header of each page must be updated when the full page is received. Therefore, the system must ensure that enough memory is available. The memory management will be handled within the Dialogic® Diva® SDK. The amount of memory required to store a page depends on the image and can vary between 20 KB and 900 KB. The Diva SDK uses an intelligent memory management based on 64 KB memory pages. If the Diva SDK cannot allocate the required memory, the connection will be dropped and the disconnect reason will be set to *DivaDROutOfMemory*.

Once a page is received, the Diva SDK signals the event *DivaEventFaxPageReceived*. The application can now retrieve the data for the page and process the data.

Note: Since the amount of data to be retrieved can be very large, the application must ensure that the Diva API is not blocked if it uses the callback mode for event processing.

When the connection of an incoming fax call is reported to the application via *DivaEventCallConnected*, the application must ensure that *DivaReceiveFaxToMemory* is called in a reasonable time. The SDK will save data in the internal memory buffer depending on the registration parameter. If *DivaReceiveFaxToMemory* is called too late, data may be lost and the call will be disconnected with the reason *DivaDRBufferOverflow*.

See also

DivaReadFaxData, DivaReceiveFax

DivaReadFaxData

DivaReadFaxData retrieves the fax data buffered in the memory.

```
DWORD    DivaReadFaxData (    DivaCallHandle    hdCall,
                              unsigned char*    pBuffer,
                              DWORD             BufferSize,
                              DWORD*            pDataLength );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall*, *DivaConnect*, or signaled with the event *DivaEventIncomingCall*.

pBuffer

[out] The *pBuffer* parameter points to a buffer that receives the data.

BufferSize

[in] The *BufferSize* parameter specifies the length of the data buffer in bytes.

pDataLength

[out] The *pDataLength* parameter points to a location that receives the amount of bytes copied to the buffer.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorEndOfData*, *DivaErrorInvalidState*, *DivaErrorInvalidParameter*, and *DivaErrorInvalidHandle*.

Remarks

The function retrieves data for a fax reception initiated via *DivaReceiveFaxToMemory*. Once the availability of data is signaled via the event *DivaEventFaxPageReceived* or *DivaEventFaxReceived*, the application should call *DivaReadFaxData* in a loop until the amount of read bytes is zero. If the end of the fax document is reached, the return code is *DivaErrorEndOfData*.

For more information on receive fax to memory refer to the remarks section of *DivaReceiveFaxToMemory*.

See also

DivaReceiveFax, DivaReceiveFaxToMemory

Voice transfer functions

This chapter contains various voice transfer functions.

DivaSendVoiceFile

DivaSendVoiceFile streams a given audio file.

```
DWORD      DivaSendVoiceFile (        DivaCallHandle       hdCall,
                                      char                 *pFilename,
                                      BOOL                 bContinuous );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pFilename

[in] The *pFilename* parameter points to the filename of the file containing the audio data.

bContinuous

[in] If the *bContinuous* parameter is set to TRUE, the audio data is streamed until *DivaStopSending* is called.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorOpenFile*, *DivaErrorReadFile*, *DivaErrorUnsupportedFormat*, *DivaErrorInvalidParameter*, *DivaErrorInvalidState,* and *DivaErrorInvalidHandle*.

Remarks

The function opens the given file and converts the audio data to line format. Any other pending streaming is automatically terminated.

The standard wave formats are supported. Available codecs are specified by the wave formats listed in *DivaAudioFormat*. The function detects the format in the header of the wave audio file.

The event *DivaEventSendVoiceEnded* signals that the audio streaming is finished. If the continuous mode is selected, the event *DivaEventSendVoiceRestarted* is signaled every time the audio is restarted.

See also

DivaEventSendVoiceDone, DivaStopSending, DivaLineCodec

DivaSendMultipleVoiceFiles

DivaSendMultipleVoiceFiles streams voice data from several files.

```
DWORD      DivaSendMultipleVoiceFiles (      DivaCallHandle       hdCall,
                                             int                  nFiles,
                                             char                 **ppFileArray,
                                             BOOL                 bContinuous );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

nFiles

[in] The *nFiles* parameter specifies the number of files in *ppFileArray*.

ppFileArray

[in] The *ppFileArray* parameter is a pointer to an array of pointers to file names. These files are streamed one after the other.

bContinuous

[in] If the parameter *bContinuous* is set to TRUE, streaming of the audio data is repeated until it is explicitly stopped or the connection is terminated.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function opens the given files and converts the audio data to line format. Any other pending streaming operation is automatically terminated.

The standard wave formats are supported. Available codecs are specified by the wave formats listed in *DivaAudioFormat*. The function detects the format in the header of the wave audio file.

The *DivaEventSendVoiceDone* signals that audio streaming is finished. If the *bContinuous* flag is set, the event is signaled each time the end of the last audio file is reached.

The event *DivaEventSendVoiceEnded* signals that the audio streaming is finished. If the continuous mode is selected, the event *DivaEventSendVoiceRestarted* is signaled every time the audio is restarted.

See also

DivaEventSendVoiceDone, DivaStopSending, DivaLineCodec, DivaSendVoiceFile

DivaSendVoiceEx

DivaSendVoiceEx streams the given audio data either from a file or a memory in the given format.

```
DWORD      DivaSendVoiceEx (    DivaCallHandle        hdCall,
                                DWORD                 NumObjects,
                                DivaVoiceDescriptor   *pDesciptor,
                                BOOL                  bContinuous,
                                DWORD                 MaxSeconds );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

NumObjects

[in] The *NumObjects* parameter defines the number of members in the voice descriptor table pointed to by *pDescriptor*.

pDescriptor

[in] The *pDescriptor* parameter points to an array containing elements of the type DivaVoiceDescriptor. These elements define which kind of audio data should be streamed.

bContinuous

[in] If the *bContinuous* parameter is set to TRUE, the audio data is streamed until the maximum time is reached, *DivaStopSending* is called or the connection is disconnected.

MaxSeconds

[in] The *MaxSeconds* parameter defines the maximum period of time that the data should be streamed. If this parameter is set to zero, no limit is set.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorOpenFile*, *DivaErrorReadFile*, *DivaErrorUnsupportedFormat*, *DivaErrorInvalidParameter*, *DivaErrorInvalidState,* and *DivaErrorInvalidHandle*.

Remarks

The function allows flexible streaming of memory or file-based audio data using different codecs. Several audio fragments can be combined to a single announcement. Auto repeat as well as duration limitations are possible. The audio fragments are defined by DivaVoiceDescriptor.

The function streams the data defined in the given descriptors. Each descriptor defines either a file-based or memory-based data set in the specified voice format. The descriptors may also define start position and duration of the streaming independent from the file or memory length.

The Diva API streams all voice data from all descriptors. When all data from all descriptors is streamed or the maximum time defined by *MaxSeconds* is reached, the event *DivaEventSendVoiceEnded* is signaled to the application. If the continuous mode is selected, the event *DivaEventSendVoiceRestarted* is signaled every time the audio is restarted.

See also

DivaSendVoiceFile, DivaRecordVoiceFile, DivaStopSending, DivaEventSendVoiceDone

DivaAppendVoice

DivaAppendVoice appends the given audio data for streaming.

```
DWORD        DivaAppendVoice (        DivaCallHandle         hdCall,
                                      DivaVoiceDescriptor    *pDesc );
```

Parameters

hdCall

[in] The *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pDesc

[in] The parameter *pDesc* points to an element of type *DivaVoiceDescriptor*. This element describes which kind of audio data should be streamed.

Return values

If the function succeeds the return value is DivaSuccess (0). Possible other return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function allows flexible streaming of memory or file based audio data. The format and the type are specified by the parameter *pDesc* that points to an element of *DivaVoiceDescriptor*.

The function adds the streaming to any previously initiated streaming. If no streaming is active, the function also triggers the streaming. Once the streaming is finished, the event *DivaEventSendVoiceEnded* is signaled.

The function is only available for calls initiated with the call type *DivaCallTypeVoice*. For all other call types the function returns *DivaErrorInvalidFunction*.

See also

DivaSendVoiceFile, DivaSendMultipleVoiceFiles, DivaSendVoiceEx, DivaStopSending

DivaStopSending

DivaStopSending stops any data streaming right away.

```
DWORD          DivaStopSending (          DivaCallHandle          hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function stops any pending data streaming. It replaces the voice-specific function *DivaStopSendingVoiceFile.* The function returns right away. The Diva API confirms that the streaming has stopped and the resources are freed via the event *DivaEventVoiceCancelled*.

See also

DivaSendVoiceFile, DivaSendVoiceEx, DivaSendData

DivaPauseSend

DivaPauseSend pauses a currently active streaming.

```
DWORD          DivaPauseSend (          DivaCallHandle          hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaPauseSend* an application pauses the currently active streaming. The pausing is only defined for streaming from audio files.

See also

DivaContinueSend, DivaStopSending, DivaForwardSend, DivaRewindSend, DivaGetSendPosition

DivaContinueSend

DivaContinueSend continues a previously paused audio streaming.

```
DWORD          DivaContinueSend (          DivaCallHandle          hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaContinueSend* an application continues a previously paused streaming.

See also

DivaPauseSend, DivaStopSending, DivaForwardSend, DivaRewindSend, DivaGetSendPosition

DivaForwardSend

DivaForwardSend positions the active audio streaming.

```
DWORD        DivaForwardSend (        DivaCallHandle          hdCall,
                                      DivaVoicePosition       *pPosition );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Diva API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pPosition

[in] Pointer to a location that holds the parameter for the new position. For more information on the format see *DivaVoicePosition*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaForwardSend* the position of the audio file currently streamed is forwarded. If the streaming is currently paused, only the position is updated. The application must call *DivaContinueSend* to continue streaming. The new position is relative to the current position. The position can be specified in bytes or in milliseconds.

If the new position is larger than the available data to stream, the streaming is stopped and the corresponding event is fired.

See also

DivaPauseSend, DivaContinueSend, DivaStopSending, DivaRewindSend, DivaGetSendPosition

DivaRewindSend

DivaRewindSend positions the active audio streaming.

```
DWORD        DivaRewindSend (         DivaCallHandle          hdCall,
                                      DivaVoicePosition       *pPosition );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pPosition

[in] Pointer to a location that holds the parameter for the new position. For more information on the format, see *DivaVoicePosition*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaRewindSend* the position of the audio file currently streamed is rewound. If the streaming is currently paused, only the position is updated. The application must call *DivaContinueSend* to continue streaming. The new position is relative to the current position. The position can be specified in bytes or in milliseconds.

See also

DivaPauseSend, DivaContinueSend, DivaStopSending, DivaForwardSend, DivaGetSendPosition

DivaGetSendPosition

DivaGetSendPosition retrieves the current position of an active audio streaming.

```
DWORD     DivaGetSendPosition (     DivaCallHandle       hdCall,
                                    DivaVoicePosition    *pPosition );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pPosition

[in out] Pointer to a location that holds and returns the parameter about the positioning.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The application may retrieve the current position of the audio streaming any time while the streaming from a file is running.

The application must set the *Size* and *Format* parameter in the *DivaVoicePosition* before calling *DivaGetSendPosition*. When the function returns, the position value is set according to the requested format.

The internal position value is reset to zero with every new initiated streaming.

See also

DivaPauseSend, DivaContinueSend, DivaStopSending, DivaForwardSend, DivaRewindSend

DivaPauseRecording

DivaPauseRecording pauses a currently running voice recording.

```
DWORD     DivaPauseRecording (     DivaCallHandle       hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The application may pause an active recording at any time. A paused recording can be continued using *DivaContinueRecording* or stopped using *DivaStopRecording*.

See also

DivaContinueRecording, DivaStopRecording, DivaGetRecordPosition

DivaContinueRecording

DivaContinueRecording continues a previously paused recording.

```
DWORD     DivaContinueRecording (     DivaCallHandle        hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

This function continues a previously paused recording.

See also

DivaPauseRecording, DivaStopRecording, DivaGetRecordPosition

DivaGetRecordPosition

DivaGetRecordPosition retrieves the current recording position.

```
DWORD     DivaGetRecordPosition (     DivaCallHandle        hdCall,
                                      DivaVoicePosition     *pPosition );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pPosition

[in out] Pointer to a location that holds and returns the parameter about the positioning.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The application may retrieve the current position of the audio recording at any time while recording is active.

The application must set the *Size* and *Format* parameter in the *DivaVoicePosition* before calling *DivaGetSendPosition*. When the function returns, the position value is set according to the requested format.

The internal position value is reset to zero with every new initiated streaming.

See also

DivaContinueRecording, DivaStopRecording, DivaPauseRecording

DivaSetVolume

DivaSetVolume sets the volume for inbound and outbound streaming.

```
DWORD      DivaSetVolume (           DivaCallHandle        hdCall,
                                     DivaVolume            Volume,
                                     DivaDirection         Direction );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Volume

[in] The parameter *Volume* specifies the new volume to set. The value must be in the range *DivaVolumeMin* to *Diva VolumeMax*.

Direction

[in] The parameter *Direction* specifies the direction for which the new volume should be used. Possible values are defined in *DivaDirection*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The volume can be specified in a range of -18 to +18 db. The *Volume* parameter specifies this. The volume can be specified per direction, depending on the parameter *Direction*.

See also

No references.

DivaEnableEchoCanceller

DivaEnableEchoCanceller enables or disables the echo canceller of a voice call.

```
DWORD      DivaEnableEchoCanceller (      DivaCallHandle        hdCall,
                                          BOOL                  bEnable );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

bEnable

[in] The parameter *bEnable* specifies if the echo canceller is enabled or disabled.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The echo canceller can be switched on when the connection is established via *DivaConnectVoice* or *DivaAnswerVoice*. In addition, the application may control the echo canceller when the call is in the connected state using *DivaEnableEchoCanceller*. The state of the echo canceller can be seen in the *DivaCallInfo*.

See also

No references.

DivaRecordVoiceFile

DivaRecordVoiceFile writes the received audio stream to a file.

```
DWORD       DivaRecordVoiceFile (      DivaCallHandle      hdCall,
                                       char                *pFilename,
                                       DivaAudioFormat     Format,
                                       DWORD               MaxRecordTime );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pFilename

[in] The *pFilename* parameter points to the filename of the file where the audio data is stored.

Format

[in] The *Format* parameter specifies the audio format to be used. For supported formats, see DivaAudioFormat.

MaxRecordTime

[in] The *MaxRecordTime* parameter specifies the time, in seconds, that is allowed for recording. A value of zero allows unlimited recording.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorCreateFile*, *DivaErrorWriteFile*, *DivaErrorUnsupportedFormat*, *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

The function converts the received audio stream to the requested data format and creates the file. Note that existing files will be overwritten.

The function returns right away. Recording ends when *DivaStopRecording* is called, the maximum time is reached, or a line drop occurs. The application may specify a maximum silence via the call property *DivaCPT_VoiceRecordSilenceTimeout*. When the recording ends, the event *DivaEventRecordVoiceEnded* is signaled. The reason for the termination is signaled with the event. Refer to *DivaRecordEndReasons* for available reasons.

See also

DivaStopRecording

DivaReceiveAudio

DivaReceiveAudio retrieves received audio data in the requested audio format.

```
DWORD          DivaReceiveAudio (          DivaCallHandle     hdCall,
                                           unsigned char      *pBuffer,
                                           DWORD              BufferSize,
                                           DWORD              *pBytesWritten,
                                           DivaAudioFormat    Format );
```

Parameters

hdCall

[in] The *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pBuffer

[in] The parameter *pBuffer* specifies the location where the received audio data should be written.

BufferSize

[in] The parameter *BufferSize* specifies the length of the buffer in bytes.

pBytesWritten

[out] The parameter *pBytesWritten* points to a location of type DWORD where the amount of bytes written to the buffer is placed.

Format

[in] The parameter *Format* specifies the audio format for which the application requests the data. Possible options are the raw formats of *DivaAudioFormat*.

Return values

If the function succeeds the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function retrieves received audio information, converts it to the requested audio format and writes it to the given buffer. The supported audio formats are the raw formats defined in *DivaAudioFormat*. These formats do not write a header but contain the plain audio information. In general, the function works like *DivaReceiveData* and the data conversion is done additionally.

This function is only available for calls made with the call type *DivaCallTypeVoice*. For all other call types the function returns *DivaErrorInvalidFunction*.

Available data is signaled by the event *DivaEventDataAvailable*, if no recording is active.

Note: The length of the available data is reported in the line format. Depending on the requested audio format, the amount of data retrieved by the application may be much longer.

The function returns right away, independent from the event mode.

See also

DivaReceiveData, DivaRecordVoiceFile

DivaStopRecording

DivaStopRecording stops the recording initiated by *DivaRecordVoice*File right away.

```
DWORD      DivaStopRecording (      DivaCallHandle        hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState,* and *DivaErrorInvalidHandle*.

Remarks

The function stops any pending audio recording initiated by *DivaRecordVoiceFile*. The function returns right away. The event *DivaEventRecordVoiceEnded* is signaled when the recording has finished and the file can be accessed by the application. The record end reason for a user initiated termination is *DivaRecordEndReasonUndefined*.

See also

DivaRecordVoiceFile, DivaSendVoiceFile

DivaGetVoiceFileLength

DivaGetVoiceFileLength calculates the length of the given voice file.

```
DWORD      DivaGetVoiceFileLength (      DivaCallHandle        hdCall,
                                         char*                 pFilename,
                                         DivaAudioFormat       Format,
                                         DWORD                 pLength );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall*, *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pFilename

[in] the *pFilename* parameter points to the filename of the file where the audio data is stored.

Format

[in] The *Format* parameter specifies the audio format of the file. For supported formats, see DivaAudioFormat.

pLength

[in] The *pLength* parameter points to a location that receives the length of the voice file.

Return Values

If the function succeeds the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorOpenFile*, *DivaErrorReadFile*, *DivaErrorInvalidHandle*, *DivaErrorInvalidState*, or *DivaErrorInvalidParameter.*

Remarks

The function calculates the length of the given audio file in the requested format, either in bytes or in milliseconds.

If the format of the existing file allows reading the audio format from the file, the parameter *Format* will be ignored.

See also

DivaRecordAppendVoiceFile, DivaSetVoiceFileLength, DivaRecordVoiceFile

DivaSetVoiceFileLength

DivaSetVoiceFileLength changes the length of the voice file to the specified value.

```
DWORD     DivaGetVoiceFileLength (     DivaCallHandle      hdCall,
                                       char*               pFilename,
                                       DivaAudioFormat     Format,
                                       DWORD               Length );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall*, *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pFilename

[in] the *pFilename* parameter points to the filename of the file where the audio data is stored.

Format

[in] The *Format* parameter specifies the audio format of the file. For supported formats see DivaAudioFormat.

Length

[in] The *Length* parameter specifies the new length of the voice file.

Return Values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorOpenFile, DivaErrorReadFile, DivaErrorInvalidHandle, DivaErrorInvalidState* or *DivaErrorInvalidParameter*.

Remarks

The function changes the length of the given voice file. If the voice file contains a header containing length information, the header is updated as well. If a length is specified that is larger than the current length the function returns *DivaErrorInvalidParameter*.

If the format of the existing file allows reading the audio format from the file, the parameter *Format* will be ignored.

See also

DivaRecordVoiceFile, DivaRecordAppendVoiceFile, DivaGetVoiceFileLength,

DivaRecordAppendVoiceFile

DivaRecordAppendVoiceFile appends received audio data to the given file.

```
DWORD     DivaRecordAppendVoiceFile (     DivaCallHandle      hdCall,
                                          char*               pFilename,
                                          DivaAudioFormat     Format,
                                          DWORD               MaxRecordTime );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall, DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pFilename

[in] The *pFilename* parameter points to the filename of the file where the audio data is stored.

Format

[in] The *Format* parameter specifies the audio format to be used. For supported formats see DivaAudioFormat.

Return values

If the function succeeds the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorCreateFile*, *DivaErrorOpenFile*, *DivaErrorWriteFile*, *DivaErrorInvalidHandle*, *DivaErrorInvalidState*, or *DivaErrorInvalidParameter*.

Remarks

In general, the function works like *DivaRecordVoiceFile*. If an existing audio file is detected, the audio stream is appended to this file. If the audio file does not exist, the function behaves like *DivaRecordVoiceFile*.

The maximum length to record specifies the time that *DivaRecordAppendVoiceFile* will add to a potential existing file.

If the file already exists and the format of the existing file allows to read the audio format from the file, the parameter *Format* will be ignored.

See also

DivaRecordVoiceFile, DivaStopRecording

DTMF and tone support

The Dialogic® Diva® Diva API includes the *DivaReportDTMF* and *DivaSendDTMF* functions to support DTMF tone detection and generation. It also supports enhanced tone generation and detection if the number of DSPs of a Dialogic® Diva® Media Board corresponds to the number of available channels. Enhanced tone detection can be enabled per connection as needed. The following functions are available:

- DivaReportTones
- DivaSendTone
- DivaSendContinuousTone
- DivaStopContinuousTone

The enhanced tone support includes detection of single tones such as multi-frequency tones and continuous tones such as ring tones. Some tones, for example the human voice, can be detected but not generated. The DivaContinuousTones and DivaMultiFrequencyTones data structures describe the various tones.

Note: It is not recommended to enable and disabled detectors based on detected tones. Depending on the length of a tone, this may lead into double detection of tones on Diva boards.

DivaReportDTMF

DivaReportDTMF switches reporting of DTMF tones on or off.

```
DWORD     DivaReportDTMF (     DivaCallHandle     hdCall,
                               BOOL               bEnable );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

bEnable

[in] If *bEnable* is TRUE, detection of tones is initiated.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function enables or disables reporting of tones. Detected digits are signaled by the event *DivaEventDTMFReceived*. The criteria for the detection can be changed by setting the pause and duration of the DTMF digit using the call properties *VoiceDTMF_DetectDuration* and *VoiceDTMF_DetectPause*.

The application may use the automatic processing of DTMF digits via *DivaSetDTMFProcessingRules*.

See also

DivaSendDTMF, DivaEventDTMFReceived, DivaSetDTMFProcessingRules

DivaSendDTMF

DivaSendDTMF sends a given sequence of DTMF tones.

```
DWORD       DivaSendDTMF (      DivaCallHandle      hdCall,
                                char                *pTones );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pTones

[in] The *pTones* parameter points to a zero-terminated string containing the DTMF tones to be sent.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function sends the given sequence of DTMF tones to the remote side. It returns right away. If a voice file is being streamed, streaming is interrupted while the tones are sent. Valid DTMF tones are "0 to "9", "A" to "D","*", and "#".

The pause and duration of the DTMF digits can be specified by the call properties *DivaCPT_VoiceDTMF_SendDuration* and *DivaCPT_VoiceDTMF_SendPause*.

See also

DivaReportDTMF

DivaReportTones

DivaReportTones switches reporting of single or continuous tones on or off.

```
DWORD       DivaReportTones (      DivaCallHandle      hdCall,
                                   BOOL                bEnable );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

bEnable

[in] If *bEnable* is TRUE, tone detection is enabled.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function enables or disables reporting of tones. The detected tones are signaled with the event *DivaEventToneDetected*. The tones that can be detected are defined in DivaContinuousTones and DivaMultiFrequencyTones.

The detection of continuous tones generates two signals, one when the tone starts and *DivaEndOfTone* when the tone stops.

See also

DivaSendTone, DivaSendContinuousTone, DivaStopContinuousTone, DivaEventToneDetected

DivaSendTone

DivaSendTone sends a given sequence of multi-frequency tones.

```
DWORD        DivaSendTone (       DivaCallHandle              hdCall,
                                  DWORD                       NumTones,
                                  DivaMultiFrequencyTones     *pTones );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

NumTones

[in] The *NumTones* parameter specifies the number of tones available in the array pointed to by *pTones*.

pTones

[in] The *pTones* parameter points to an array that contains the tones to be sent.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function sends the given sequence of tones to the remote side. The function returns right away. If a voice file is being streamed, streaming is interrupted while the tones are sent.

The *DivaEventSendToneEnded* event is sent when the last tone has been streamed.

See also

DivaReportDTMF, DivaSendContinuousTone, DivaStopContinuousTone, DivaEventToneDetected

DivaSendContinuousTone

DivaSendContinuousTone sends a continuous tone for a given maximum of time.

```
DWORD      DivaSendContinuousTone (      DivaCallHandle          hdCall,
                                         DivaContinousTones      Tone,
                                         DWORD                   MaxSeconds );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Tone

[in] The *Tone* parameter defines the continuous tone to be streamed. For possible options, see DivaContinuousTones.

MaxSeconds

[in] If the *MaxSeconds* parameter is set to non-zero, it specifies the period of time after which streaming is stopped.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function initiates the streaming of the requested tone. Any previously initiated audio streaming is interrupted. The application may limit the length of the tone streaming by setting *MaxSeconds* to non-zero.

The function returns right away. The streaming of the tone ends when the maximum time is reached, if selected, or the function *DivaStopContinuousTone* is called. In both cases, the event *DivaEventSendToneEnded* is signaled when the streaming has stopped.

See also

DivaSendTone, DivaReportDTMF, DivaStopContinuousTone, DivaEventToneDetected

DivaStopContinuousTone

DivaStopContinuousTone stops the streaming of a continuous tone.

```
DWORD    DivaStopContinuousTone (    DivaCallHandle    hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function stops the streaming of a continuous tone initiated by *DivaSendContinuousTone*. Any other streaming that was active before the tone was sent is continued.

The function returns right away. The *DivaEventSendContinuousToneEnded* event is signaled when the streaming has stopped.

See also

DivaReportDTMF, DivaSendTone, DivaSendContinuousTone, DivaEventToneDetected

DivaGenerateSingleTone

DivaGenerateSingleTone generates a single tone of the given frequency and amplitude.

```
DWORD    DivaGenerateTone (    DivaCallHandle    hdCall,
                               DWORD             Frequency,
                               int               Amplitude,
                               DWORD             Duration );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Frequency

[in] The parameter *Frequency* specifies the frequency of the tone to be generated in Hz. The value must be in the range from 0 to 4000 Hz.

Amplitude

[in] The parameter *Amplitude* specifies the amplitude of the tone to be generated. The amplitude is specified in dBm in the range of 127.996 to -127.996. The value -32767 corresponds to -127.996 dBm and the value 32767 corresponds to +127.996 dBm.

Duration

[out] The parameter *Duration* specifies the duration of the tone in milliseconds. A value of zero indicates no timeout and the application must stop the tone via *DivaStopToneGeneration*. The maximum value is 65535.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function validates that the requested tone can be generated. If successful, the tone generation is started. The tone is either generated for a specific time or the application controls the duration and stops the tone using *DivaStopToneGeneration*. If the tone stopped via a timeout, the event *DivaEventGenericToneEnded* is signaled.

Only one tone can be generated at a time. If another request to generate a single or dual tone is issued, the current tone is stopped.

See also

No references.

DivaGenerateDualTone

DivaGenerateDualTone generates a dual tone of the given frequencies and amplitudes.

```
DWORD    DivaGenerateDualTone (    DivaCallHandle    hdCall,
                                   DWORD             FrequencyA,
                                   int               AmplitudeA,
                                   DWORD             FrequencyB,
                                   int               AmplitudeB,
                                   DWORD             Duration );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

FrequencyA

[in] The *FrequencyA* parameter specifies the frequency of the first tone to be generated in Hz. The value must be in the range from 0 to 4000 Hz.

AmplitudeA

[in] The *AmplitudeA* parameter specifies the amplitude of the first tone to be generated. The amplitude is specified in dBm in the range of 127.996 to -127.996. The value -32767 corresponds to -127.996 dBm and the value 32767 corresponds to +127.996 dBm.

FrequencyB

[in] The *FrequencyB* parameter specifies the first frequency of the second tone to be generated in Hz. The value must be in the range from 0 to 4000 Hz.

AmplitudeB

[in] The *AmplitudeB* parameter specifies the amplitude of the second tone to be generated. The amplitude is given in dBm. The amplitude is specified in dBm in the range of 127.996 to -127.996. The value -32767 corresponds to -127.996 dBm and the value 32767 corresponds to +127.996 dBm.

Duration

[out] The *Duration* parameter specifies the duration of the tone in milliseconds. A value of zero indicates no timeout and the application must stop the tone via *DivaStopToneGeneration*. The maximum value is 65535.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function validates that the requested tone can be generated. If successful, the tone generation is started. The tone is either generated for a specific time or the application controls the duration and stops the tone using *DivaStopToneGeneration*. If the tone stopped via a timeout, the event *DivaEventGenericToneEnded* is signaled.

Only one tone can be generated at a time. If another request to generate a single or dual tone is issued, the current tone is stopped.

See also

No references.

DivaStopToneGeneration

DivaStopToneGeneration stops the currently generated tone.

```
DWORD      DivaStopToneGeneration (         DivaCallHandle               hdCall );
```

Parameters

hdCall

[in] The *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

The function stops the generation of a tone. The tone stops when the function returns. No event *DivaEventGenericToneEnded* is signaled.

See also

No references.

DivaDetectSingleTone

DivaDetectSingleTone enables the generic tone detector for a single tone.

```
DWORD     DivaDetectSingleTone (     DivaCallHandle     hdCall,
                                     DWORD              ReportFlags,
                                     DWORD              MinDuration,
                                     int                MinSNR,
                                     int                MinLevel,
                                     DWORD              MaxAM,
                                     DWORD              MaxFM );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

ReportFlags

[in] The *ReportFlags* parameter specifies which characteristics of the detected tone should be reported. For options, refer to DivaSingleToneReport.

MinDuration

[in] The *MinDuration* parameter specifies the minimum duration of a tone before the detection is reported. The time is given in milliseconds.

MinSNR

[in] The *MinSNR* parameter specifies the minimum signal to noise ratio. The value is specified in dB in the range of 128 dB to -128 dB. The value of -32768 corresponds to -128 dB, the value 32767 corresponds to +127.996 dB.

MinLevel

[in] The *MinLevel* parameter specifies the minimum level of the detected signal. The value is specified in dB in the range of 127.996 to -127.996. The value of -1 corresponds to any level, the value -32767 corresponds to -127.996 dB and the value 32767 corresponds to +127.996 dB.

MaxAM

[in] The *MaxAM* parameter specifies the maximum allowed variation of the signal level. This corresponds to the maximum amplitude modulation. The value is given in dB in the range of 0 db (0) to 255.996 dB (65535).

MaxFM

[in] The *MaxFM* parameter specifies the maximum allowed variation of the signal frequency. This corresponds to the maximum frequency modulation. The value is given in the range of 0 to 4000 Hz.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function validates that the requested tone can be detected. If successful, the tone detection is started. Any previously enabled and still pending generic tone detection is stopped.

When a tone within the specified range is detected the event *DivaEventGenericToneDetector* is signaled. The application must retrieve the information via *DivaGetToneDetectorResult*.

See also

No references.

DivaDetectDualTone

DivaDetectDualTone enables the generic tone detector for a dual tone.

```
DWORD     DivaDetectDualTone (     DivaCallHandle     hdCall,
                                   DWORD              ReportFlags,
                                   DWORD              MinDuration,
                                   int                MinSNR,
                                   int                MinLevel,
                                   int                MaxDiffHighToLow,
                                   int                MaxDiffLowToHigh );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

ReportFlags

[in] The *ReportFlags* parameter specifies which characteristics of the detected tone should be reported. For options, refer to DivaDualToneReport.

MinDuration

[in] The *MinDuration* parameter specifies the minimum duration of a tone before the detection is reported. The time is given in milliseconds.

MinSNR

[in] The *MinSNR* parameter specifies the minimum signal to noise ratio. The value is specified in dB in the range of 128 to -128. The value of -32768 corresponds to -128 dB, the value 32767 corresponds to +127.996 dB.

MinLevel

[in] The *MinLevel* parameter specifies the minimum level of the detected signal. The value is specified in dB in the range of 127.996 to -127.996. The value of -32768 corresponds to no minimum level, the value -32767 corresponds to -127.996 dB and the value 32767 corresponds to +127.996 dB.

MaxDiffHighToLow

[in] The *MaxDiffHighToLow* parameter specifies the maximum allowed difference in levels between the higher and the lower frequency tone. The value -32767 corresponds to -127.996 dB and the value 32767 corresponds to +127.996 dB. The value -32768 is invalid. A dual tone is valid when the level of the higher frequency tone does not exceed the level of the lower frequency tone by more than *MaxDiffHighToLow* dB.

MaxDiffLowToHigh

[in] The *MaxDiffLowToHigh* parameter specifies the maximum allowed difference in levels between the lower and higher frequency tone. The value -32767 corresponds to -127.996 dB and the value 32767 corresponds to +127.996 dB. The value -32768 is invalid. A dual tone is valid when the level of the lower frequency tone does not exceed the level of the higher frequency tone by more than *MaxDiffHighToLow* dB.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function validates that the requested tone can be detected. If successful, the tone detection is started. Any pending previously enabled generic tone detection is stopped.

When a matching dual tone is detected the event *DivaEventGenericToneDetector* is signaled. The application must retrieve the information via *DivaGetToneDetectorResult*.

See also

No references.

DivaGetToneDetectorResult

DivaGetToneDetectorResult retrieves the information for a detected single or dual tone.

```
DWORD    DivaGetToneDetectorResult (    DivaCallHandle             hdCall,
                                        DivaToneDetectorResults    *pResults );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pResults

[out] The *pResult* parameter specifies a location in memory of type *DivaToneDetectorResults* where the information about the detected tone is written.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorInvalidState*.

Remarks

The function validates that tone detection is pending and detector information is available. The information depends on the enabled detector and if the tone started or stopped. For details on the information, refer to *DivaToneDetectorResults*.

See also

No references.

DivaSendGenericToneRequest

DivaSendGenericToneRequest sends a request coded by the application to the generic tone engine.

```
DWORD   DivaSendGenericToneRequest (    DivaCallHandle             hdCall,
                                        DivaGenericToneFunction    Function,
                                        BYTE                       *pRequest,
                                        DWORD                      RequestLen,
                                        Void                       *Handle );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Function

[in] The *Function* parameter specifies the requested function. The functions are Get Supported Services, Enable Tone operation and Disable Tone operation.

pRequest

[in] The *pRequest* parameter specifies a location in the memory where the generic tone request is stored. Upon return of the function, the buffer is free.

RequestLen

[in] The *RequestLen* parameter specifies the amount of data in *pRequest* in bytes.

Handle

[in] The *Handle* parameter specifies an application defined value that is signaled with the confirmation for the request.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

The given request data is passed to the generic tone engine without further interpretation. The application must ensure that the data is coded in accordance with the CAPI Extensions "Generic Tone Generator and Detector".

Each request that succeeds is answered by a confirmation. The confirmation is signaled via the event *DivaEventGenericToneInfo*. The application must retrieve the information using *DivaGetGenericToneInfo*.

The detector signals results also via the event *DivaEventGenericToneInfo*. The application retrieves the information via *DivaGetGenericToneInfo*. The returned data contains information regarding whether data should be interpreted as confirmation or indication data.

See also

No references.

DivaGetGenericToneInfo

DivaGetGenericToneInfo retrieves a confirmation or indication from the generic tone engine.

```
DWORD    DivaGetGenericToneInfo (    DivaCallHandle          hdCall,
                                     DivaGenericToneInfo     *pInfoBuffer,
                                     DWORD                   InfoBufferLen,
                                     DWORD                   *pBytesWritten );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pInfoBuffer

[in] The *pInfoBuffer* parameter specifies a location in the memory where the information is placed. For more information on the structure of the information, refer to DivaGenericToneInfo.

InfoBufferLen

[in] The *InfoBufferLen* parameter specifies the overall size in bytes of the memory specified by *pInfoBuffer*.

pBytesWritten

[in] The *pBytesWritten* parameter specifies a location in memory of type DWORD where the amount of data written to *pInfoBuffer* is placed. If the application is not interested in this information, *pBytesWritten* may be set to zero.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

The application calls *DivaGetGenericToneInfo* when the event *DivaEventGenericToneInfo* is signaled. The function provides a confirmation or indication information to the application.

The application must provide a buffer of type *DivaGenericToneInfo*. The required size can be queried by setting *pInfoBuffer* to zero. In this case, the required size is returned in the location specified by *pBytesWritten*.

See also

No references.

DivaSetDTMFProcessingRules

DivaSetDTMFProcessingRules defines the action in combination with received DTMF digits.

```
DWORD    DivaSetDTMFProcessingRules (    DivaCallHandle          hdCall,
                                         DivaProcessingGroup     Group,
                                         DWORD                   TerminationDigitMask,
                                         DWORD                   MaxDigits,
                                         DWORD                   InterDigitTimeout,
                                         DWORD                   IntialDigitTimeout,
                                         DWORD                   MaxTimeout );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Group

[in] The parameter *Group* specifies the category for which the rules are valid. The rules can be valid for event reporting, streaming, and recording. The possible options are defined in *DivaProcessingGroup*.

TerminationDigitMask

[in] The parameter *TerminationDigitMask* specifies which digits trigger an immediate action. For valid digit masks, refer to DivaTerminationDigits.

MaxDigits

[in] The parameter *MaxDigits* specifies the amount of digits that trigger an action. A value of zero disables this rule.

InterDigitTimeout

[in] The parameter *InterDigitTimeout* specifies the maximum time between two received DTMF digits. The time is given in milliseconds. The timer resolution is 100 milliseconds.

IntialDigitTimeout

[in] The parameter *IntialDigitTimeout* specifies the maximum time to receive the first DTMF digit. The time is given in milliseconds. The timer resolution is 100 milliseconds. A value of zero disables this timeout.

MaxTimeout

[in] The parameter *MaxTimeout* specifies the maximum time for the rule. If no other event terminates the rule, the maximum timeout terminates after the given time. The time is given in milliseconds. The timer resolution is 100 milliseconds. A value of zero disables this timeout.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidHandle* and *DivaErrorInvalidParameter*.

Remarks

With a call to *DivaSetDTMFProcessingRules* the application defines the processing of received DTMF digits. The processing rules can be set for each group separate. Please note that all groups process the same DTMF buffer.

All timeouts are optional and a value of zero disables them. The initial digit timeout defines the time within which the first digit is expected. The inter digit timeout is started when the first digit is received. This timeout is not valid for the first received digit. The maximum timeout can be used to have a maximum time for the whole rule if no other event terminates the rule.

Note: Calling this function has no impact on digits already in the internal buffer. If a processing rule for a termination digit or maximum digits is given and the digits in the buffer fulfill this rule, the action may be taken, depending on the group. For an event group the event would be fired right away. For the streaming group, the action would be taken when the streaming is started. A rule for streaming and recording would expire right away. If no streaming or recording is ongoing, there will be no event.

Once a rule detects one of the termination conditions, the whole rule for this group is terminated. Even if more digits are received, they are not processed for this group unless the application sets a new rule.

See also

DivaGetDTMFBuffer, DivaClearDTMFBuffer, DivaEventDTMFTerminationDigit, DivaEventDTMFMaxDigits, DivaEventDTMFInitialDigitTimeout, DivaTerminationDigits, DivaEventDTMFInterDigitTimeout

DivaGetDTMFBuffer

DivaGetDTMFBuffer retrieves the received DTMF digits.

```
DWORD    DivaGetDTMFBuffer (        DivaCallHandle        hdCall,
                                    char *                Buffer,
                                    DWORD                 BufferSize );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Buffer

[out] The parameter *Buffer* specifies a location in memory where the digits should be placed.

BufferSize

[in] The parameter *BufferSize* specifies the length of the buffer.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidHandle*, *DivaErrorInvalidParameter*, and *DivaErrorDataSize*.

Remarks

The application gets a copy of the internal DTMF buffer by calling this *DivaGetDTMFBuffer*, The digits remain in the internal buffer until the application calls *DivaClearDTMFBuffer*. The function is a synchronous function and can be called at any time.

See also

DivaSetDTMFProcessingRules, DivaClearDTMFBuffer

DivaClearDTMFBuffer

DivaClearDTMFBuffer clears the internal DTMF buffer.

```
DWORD    DivaClearDTMFBuffer (        DivaCallHandle        hdCall );
```

Parameters

hdCall

[in] The *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The function clears the internal DTMF buffer. The function is a synchronous function and can be called at any time.

See also

DivaSetDTMFProcessingRules, DivaGetDTMFBuffer

Call Transfer

Call transfer can be done in various ways. Usually, one call is on hold while the second call is created, then the transfer is completed. The second call may be created by the application or by the Dialogic® Diva® API. If the call is created inside the Diva API, the transfer is called "blind transfer". A call transfer may also be performed based on a single call, which is called Call Deflection.

To enable implementation of the call transfer functionality for application developers, the Diva API provides the framework for call transfer. In some cases, for example, if both calls must be handled on the same channel of an ISDN line, the call must be established following specific rules.

The following logical functions are available for call transfer:

- DivaSetupCallTransfer (optional)
- DivaDial (optional)
- DivaCompleteCallTransfer
- DivaBlindCallTransfer

There are different ways to complete a call transfer, depending on who is setting up the second call and how the call is created.

Transfer using consultation call

In general, the two calls that shall be transferred can be created in any way by the application. However, in certain switch environments, the Dialogic® Diva® API can only handle call transfer if it is informed on the intended transfer before the call is established. In this case, the Diva API uses a so-called consultation call object to handle the call transfer.

To tell the Diva API that a call will be handled as a consultation call for a call transfer, the application calls *DivaSetupCallTransfer*. The consultation call object is created and a handle is given to the application. When the Diva API returns control to the application, the original call is on hold.

Depending on the parameters passed to *DivaSetupCallTransfer*, the physical connection is either initiated when the consultation call object is created or not. If no destination number is given in *DivaSetupCallTransfer*, only the logical object is created and the physical connection is initiated when the application calls *DivaDial*. This ensures that the transfer also works in switch environments that only support block dialing.

The application calls *DivaCompleteCallTransfer* to complete the transfer.

Transfer using independent call objects

The application can create two independent calls, either incoming or outgoing, and transfer one to the other directly. The transfer needs to be completed using *DivaCompleteCallTransfer*. If the first call is not on hold, the Dialogic® Diva® API will put it on hold implicitly. Transfer using independent call objects is not possible in all switch environments, the application must detect whether this kind of transfer is possible or not.

Transfer on one call object

If an application just wants to forward a single call to a different destination, it uses *DivaBlindCallTransfer*. This function creates the second call and completes the transfer. In case of a transfer failure, the result code provides detailed information if the failure was related to the establishment of the second call or the transfer itself. Depending on the options passed to *DivaBlindCallTransfer*, the transfer may be handled as Call Deflection.

Transfer completion

All transfer-related function return right away, and the progress of the transfer is reported to the application via events. Implicit changes of the call state, e.g., when the active call is put on hold, are reported to the application. Once the transfer is completed, the *DivaEventTransferCompleted* event is signaled. The call objects are no longer needed and the call state changes to disconnect. The application has to free the call objects by calling *DivaCloseCall* once the event *DivaEventCallDisconnected* is received for those call objects.

DivaSetupCallTransfer

DivaSetupCallTransfer creates a consultation call object based on the given call. The original call is put on hold, if not already done.

```
DWORD     DivaSetupCallTransfer (     DivaCallHandle      hdCall,
                                      AppCallHandle       hAppConsultCall,
                                      DivaCallHandle      *phdConsultCall,
                                      char                *pDestination );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

hAppConsultCall

[in] This parameter specifies the application call handle for the consultation call. The Diva API uses this handle to report events for the consultation call.

phdConsultCall

[out] This parameter points to a location of the type *DivaCallHandle* that receives the call handle of the consultation call object on successful return.

pDestination

[in] This parameter specifies the number that should be used to establish the consultation call. It may be an empty string, see Remarks.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function initiates the transition to hold state for the call, if not already done. A consultation call object is created and filled with the *CallType* parameter and other parameters from the active call object. When the function returns, dialing the consultation call object can be started using DivaCompleteCallTransfer.

The function returns right away. The progress is reported by the *DivaEventSetupTransferCompleted* event.

The application can provide a number to be used for the consultation call. In this case, the Diva API initiates the consultation call. Depending on the options set by the call properties *DivaCPT_NoHoldBeforeTransfer* and *DivaCPT_UseSameChannelForTransfer* the primary call may be set on hold and the consultation call will be done on the same channel. If the application wants to handle dialing manually, it can set *pDestination* to an empty string and use *DivaDial* to establish the consultation call.

See also

DivaCompleteCallTransfer, DivaBlindCallTransfer, DivaCPT_NoHoldBefore Transfer, DivaCPT_UseSameChannel ForTransfer=4000,

DivaCompleteCallTransfer

DivaCompleteCallTransfer completes the transfer of the given call objects.

```
DWORD    DivaCompleteCallTransfer (    DivaCallHandle        hdCall,
                                       DivaCallHandle        hdConsultCall,
                                       DivaTransferOptions   Options );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

hdConsultCall

[in] The *hdConsultCall* parameter identifies the consultation call at the Diva API. The call has been created as consultation call via *DivaSetupCallTransfer* or as an independent call via the standard functions for call establishment.

Options

[in] This parameter specifies how the call transfer is completed.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

If the first call is not on hold, the function initiates the transition to hold state. After successful transfer, both call objects are no longer needed and the Diva API signals that the calls are disconnected. The application has to close both calls by calling *DivaCloseCall* when the event *DivaEventCallDisconencted* is signaled.

The function returns right away, and the completion of the call transfer is signaled by the event *DivaEventTransferCompleted*.

If the transfer fails, the application has to take care of both calls. The call state of the calls may have changed and the application may have to retrieve a call using *DivaRetrieve*.

See also

DivaSetupCallTransfer, DivaBlindCallTransfer

DivaBlindCallTransfer

DivaBlindCallTransfer automatically transfers the call to a given destination.

```
DWORD    DivaBlindCallTransfer (    DivaCallHandle        hdCall,
                                    Char                  *pDestination,
                                    DivaTransferOptions   Options );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

pDestination

[in] The *pDestination* parameter specifies the number to dial.

Options

[in] This parameter specifies how the transfer should be completed.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

If the first call is not on hold, the function initiates the transition hold state. Then, the second call to the given destination is created internally. Once the second call reaches the ringing state, the transfer is completed. Upon successful transfer, the call object is no longer needed and the Diva API signals that the call is disconnected. The application has to close the call by calling *DivaCloseCall*.

The function returns right away, and the success of the call transfer is signaled by the event *DivaEventTransferCompleted*.

If the transfer fails, the secondary call created by the Diva API is disconnected. The state of the primary call is restored, if possible. The application has to handle the primary call.

See also

DivaCompleteCallTransfer, DivaSetupCallTransfer

DivaLIConnect

DivaLIConnect creates a Line Interconnect between two existing voice calls.

```
DWORD     DivaLIConnect (        DivaCallHandle        hMainCall,
                                 DivaCallHandle        hCall );
```

Parameters

hMainCall

[in] The *hMainCall* parameter identifies the first call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*. On this call, Line Interconnect will be logically initiated. This call object is used for sending and receiving a mixed data stream.

hCall

[in] The *hCall* parameter identifies the second call at the Diva API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall.*

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

The function initiates Line Interconnect between the given calls. The calls are connected, and by default, no data traffic between the application and the calls takes place. Data traffic between the application and the calls can be enabled by calling any voice streaming function. On the call identified by the call handle *hMainCall*, the application can stream to both calls or receive the audio from both call, also called transaction recording. On the second call object, the streaming and recording is done as for a single call.

The function returns right away, and the event *DivaEventLIConnectCompleted* is sent when the calls are interconnected.

See also

DivaLIDisconnect, DivaEventLIConnectCompleted, DivaEventLIDisconnected, DivaLIEnableRxData

DivaLIDisconnect

DivaLIDisconnect releases a Line Interconnect between two existing calls.

```
DWORD    DivaLIDisconnect (          DivaCallHandle       hMainCall,
                                     BOOL                 bDisconnectCalls );
```

Parameters

hMainCall

[in] The *hMainCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*. This call handle must be the handle given as the main call when Line Interconnect was initiated.

bDisconnectCalls

[in] If the *bDisconnectCalls* parameter is set, the interconnected calls are released.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

The function initiates a Disconnect of the existing Line Interconnect. Both calls involved remain connected. The function returns right away, and the event *DivaEventLIDisconnected* is sent when the Line Interconnect is released.

See also

Call properties, DivaEventLIConnectCompleted, DivaEventLIDisconnected

DivaLIEnableRxData

DivaLIEnableRxData enables receive data on an interconnected call.

```
DWORD    DivaLIEnableRxData (        DivaCallHandle       hCall,
                                     BOOL                 bEnable );
```

Parameters

hCall

[in] The *hCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either the main call handle or the participating call handle used when Line Interconnect has been created.

bEnable

[in] If *bEnable* is set, receiving of data on the given call object is enabled. If the call object specifies the main call handle, the mixed data stream is used.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, and *DivaErrorInvalidHandle*.

Remarks

The function enables the indication of data received on the call object. Note that this is only for plain data streaming. Users of DivaRecordVoiceFile are not required to call this function.

See also

Call properties, DivaLIDisconnect, DivaEventLIConnectCompleted, DivaEventLIDisconnected

DivaHold

DivaHold puts the specified call on hold.

```
DWORD       DivaHold (        DivaCallHandle          hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

If the call is connected, the transition to hold state is initiated. The function returns right away. The result is reported by the *DivaEventHoldCompleted* event and the state change is reported by *DivaEventCallProgress.*

See also

DivaRetrieve, DivaCallState

DivaRetrieve

DivaRetrieve retrieves a call that has been put on hold.

```
DWORD       DivaRetrieve (      DivaCallHandle         hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState* and *DivaErrorInvalidHandle*.

Remarks

If the call is on hold, the transition to connected state is initiated using the selected call type. The function returns right away. The result is reported by the *DivaEventRetrieveCompleted* event and the state change is reported by *DivaEventCallProgress*.

If the call is not on hold, the function returns right away with the error code *DivaErrorInvalidState*.

See also

DivaEventRetrieveCompleted, DivaHold, DivaCallState

DivaSendInfo

DivaSendInfo sends an info message containing user-user information or facility information.

```
DWORD    DivaSendInfo (              DivaCallHandle        hdCall,
                                     unsigned char*        Info,
                                     DWORD                 InfoLength );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Info

[in] This parameter is for future use.

InfoLength

[in] This parameter is for future use.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function sends an info message containing an user-user information element and / or a facility information element. The information elements must be set via the call properties *DivaCPT_UserUserInfo* and *DivaCPT_FacilityDataArray*.

This function can be used to send messages to switches to initiate call transfers or other supplementary services. In general, it is recommended to use the standard call transfer methods. This function should only be used if the standard function and the underlying Dialogic® communication platform do not support the required functionality.

See also

No references.

DivaSendFlash

DivaSendFlash returns information if the device has the specified capability.

```
BOOL     DivaSendFlash (                 DivaCallHandle        hdCall,
                                         DWORD                 FlashLength );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

FlashLength

[in] The parameter *FlashLength* specifies the maximum length of the hook flash.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible values are *DivaErrorNotSupported* and *DivaErrorInvalidHandle*.

Remarks

The function initiates the sending of a hook flash. Once the flash is finished, the event *DivaEventFlashCompleted* is signaled to the application. The function is used for applications that run an own protocol using hook flash and DTMF to communicate to the switch.

See also

No references.

Conference

This chapter contains various conference functions.

DivaCreateConference

DivaCreateConference creates an internal conference object.

```
DWORD    DivaCreateConference (      DivaAppHandle         hApp,
                                     DivaCallHandle        hdCall,
                                     AppCallHandle         haConference,
                                     DivaCallHandle        *phdConference );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application. The handle is assigned by a call to *DivaRegister*.

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaWaitForCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*. Optionally, the *hdCall* parameter can be set to zero to create an empty conference object and add the members later.

haConference

[in] The *haConference* parameter identifies the application context of the conference object and is signaled with all conference-related events.

phdConference

[out] The *phdConference* parameter points to a location that receives the Diva API-related handle of the conference. This handle has to be used in all following conference-related calls.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState*, *DivaErrorLimitExceeded*, *DivaErrorInvalidParameter*, and *DivaErrorInvalidHandle*.

Remarks

With the call to *DivaCreateConference* the application creates a conference. The function creates a logical instance holding all information on the conference and the members belonging to the conference.

If the call handle is given, the conference is created based on this call handle and the call may be interpreted as master call. The call must be in proceeding, alerting, or connected state.

See also

DivaConferenceOptions, DivaEventConferenceInfo, DivaDestroyConference, DivaAddToConference, DivaRemoveFromConference, DivaGetConferenceInfo

DivaDestroyConference

DivaDestroyConference destroys a conference and optionally disconnects all calls.

```
DWORD    DivaDestroyConference (      DivaCallHandle        hdConference,
                                      BOOL                  bDisconnectCalls );
```

Parameters

hdConference

[in] The *hdConference* parameter identifies the conference previously created by DivaCreateConference.

bDisconnectCalls

[in] The *bDisconnectCalls* parameter specifies that the calls which belong to the conference are to be disconnected.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function destroys the conference and releases all corresponding resources. After the function returns, the conference handle is no longer valid.

If the parameter *bDisconnectCalls* is set to TRUE, all calls that belong to the conference are disconnected. If the parameter is set to FALSE, the calls are removed from the conference and kept the current state. This is a synchronous function. When the function returns, all calls are removed from the conference and the conference object is no longer valid. There is no event for destroying of a conference.

See also

DivaConferenceOptions, DivaEventConferenceInfo, DivaCreateConference, DivaAddToConference, DivaRemoveFromConference, DivaGetConferenceInfo

DivaConferenceSetProperties

DivaConferenceSetProperties modifies the properties of the conference*.*

```
DWORD    DivaConferenceSetProperties (    DivaCallHandle            hdConference,
                                          DWORD                     PropertyType,
                                          DivaConferenceProperty    *PropertyValue );
```

Parameters

hdConference

[in] The *hdConference* parameter identifies the conference previously created by DivaCreateConference.

PropertyType

[in] The *PropertyType* parameter specifies the property to be set. For more information see DivaConferencePropertyType.

PropertyValue

[in] The *PropertyValue* parameter depends on the type of property.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The properties of a conference are set by this function. The value type depends on the property. Refer to DivaConferencePropertyType for more information on the available properties.

See also

DivaConferenceOptions, DivaEventConferenceInfo, DivaCreateConference, DivaAddToConference, DivaRemoveFromConference, DivaGetConferenceInfo

DivaAddToConference

DivaAddToConference adds the given call to the existing conference.

```
DWORD    DivaAddToConference (    DivaCallHandle    hdConference,
                                  DivaCallHandle    hdCall );
```

Parameters

hdConference

[in] The *hdConference* parameter identifies the conference previously created by DivaCreateConference.

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidState, DivaErrorLimitExceeded, DivaErrorInvalidParameter,* and *DivaErrorInvalidHandle*.

Remarks

The function adds the call specified by the call handle to the conference. The call may be established using any connect or answer function. The state of the conference is changed to *DivaConferenceStateAdding* while the call is added.

The call must be in the proceeding, alerting, or connected state. The function returns right away. The event *DivaEventConferenceInfo* is signaled when the member is part of the conference.

See also

DivaConferenceOptions, DivaEventConferenceInfo, DivaCreateConference, DivaDestroyConference, DivaRemoveFromConference, DivaGetConferenceInfo

DivaRemoveFromConference

DivaRemoveFromConference removes the call from the conference.

```
DWORD    DivaRemoveFromConference (    DivaCallHandle      hdConference,
                                       DivaCallHandle      hdCall );
```

Parameters

hdConference

[in] The *hdConference* parameter identifies the conference previously created by DivaCreateConference.

hdCall

[in] The *hdCall* parameter identifies the call that is to be removed at the Dialogic® Diva® API. If this parameter is set to zero, the last call added to the conference is removed.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function removes the given call from the conference. The state of the conference is changed to *DivaConferenceStateRemoving* while the removal is pending.

The function returns right away, and the event *DivaEventConferenceInfo* is signaled when the removal is completed. The call state is not changed.

See also

DivaConferenceOptions, DivaEventConferenceInfo, DivaCreateConference, DivaDestroyConference, DivaAddToConference, DivaGetConferenceInfo

DivaGetConferenceInfo

DivaGetConferenceInfo retrieves the status and the members of a conference. The function is obsolete, the application should use DivaConferenceGetProperties.

```
DWORD    DivaGetConferenceInfo (       DivaCallHandle        hdConference,
                                       DivaConferenceInfo    *pConferenceInfo );
```

Parameters

hdConference

[in] The *hdConference* parameter identifies the conference previously created by DivaCreateConference.

pConferenceInfo

[in] The *pConferenceInfo* parameter is a pointer to a user supplied buffer of type DivaConferenceInfo that receives the information on the conference. Note that the application must set the size field of the *DivaConferenceInfo* structure to the size of the structure before calling the function.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The function retrieves information on the given conference. The information is copied to the buffer supplied by the caller. The function returns right away. For more information on the returned information refer to DivaConferenceInfo.

Note: The amount of members in a conference is not limited. However, the data structure *DivaConferenceInfo* limits the amount of members for retrieving information. Therefore, applications should use *DivaConferenceGetProperties* to retrieve member information.

See also

DivaConferenceInfo

DivaConferenceEnableRxData

DivaConferenceEnableRxData enables or disables data reception on the conference or call.

```
DWORD    DivaConferenceEnableRxData (       DivaCallHandle      hdObject,
                                            BOOL                bEnable );
```

Parameters

hdObject

[in] The *hdObject* parameter identifies the conference or call object.

bEnable

[in] The *bEnable* parameter defines if streaming is enabled or disabled.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

By default, no receive data is forwarded to conference members and no events for the reception of plain data are generated. With this function, the application can select that receive data is forwarded to the application.

The function only works in environments where streaming on the conference members is available. If this is not the case, *DivaErrorInvalidFunction* is returned.

Note: This is only for plain data streaming. Users of *DivaRecordVoice* are not required to call this function.

See also

No references.

DivaConferenceGetProperties

```
DWORD   DivaConferenceGetProperties (   DivaCallHandle           hdConference,
                                        DWORD                    PropertyType,
                                        DivaConferenceProperty   *PropertyValue,
                                        DWORD                    PropertyValueSize,
                                        DWORD                    *pPropertyValueSizeUsed );
```

Parameters

hdConference

[in] The *hdConference* parameter identifies the conference previously created by *DivaCreateConference*.

PropertyType

[in] The *PropertyType* parameter specifies the property to be retrieved. For more information see DivaConferencePropertyType.

PropertyValue

[out] The *PropertyValue* parameter depends on the type of property. The application must provide a buffer that is large enough to cover the parameter requested by *PropertyType*.

PropertyValueSize

[in] The *PropertyValueSize* parameter specifies the size of the buffer provided by the application.

pPropertyValueSizeUsed

[out] The *PropertyValueSizeUsed* parameter points to a location that receives the amount or bytes written to the buffer specified by *PropertyValue*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). If the buffer for the property value is not large enough, *DivaErrorOutOfMemory* is returned. Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

The properties of a conference are retrieved (get) by this function. The value type depends on the property. The application must ensure that the buffer for the property is large enough.

See also

DivaConferenceOptions, DivaEventConferenceInfo, DivaCreateConference, DivaAddToConference, DivaRemoveFromConference, DivaGetConferenceInfo, DivaConferenceSetProperties

Message Waiting Indication

This chapter contains various MWI functions.

DivaMWIActivate

DivaMWIActivate sends an message waiting activation request to the switch.

```
DWORD    DivaMWIActivate (      DivaAppHandle             hApp,
                                DWORD                     LineDeviceId,
                                DivaMWIActivateParams*    pParams );
```

Parameters

hApp

[in] *hApp* is the application handle that was returned by a call to *DivaRegister.*

LineDeviceId

[in] This parameter identifies the line device. Line devices are continuously numbered by an index starting with one.

pParams

[in] This parameter points to a data structure of the type *DivaMWIActivateParams* that contains the activation parameter.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function creates a message waiting activation request based on the given parameter. For information on the parameter refer to DivaMWIActivateParams.

The function returns right away. If the return code is *DivaSuccess*, the action is initiated. The result of the request is signaled via the event *DivaEventMWICompleted*. The handle given in the *DivaMWIActivateParams* is passed to the application with the event.

See also

DivaMWIDeactivate

DivaMWIDeactivate

DivaMWIDeactivate sends an message waiting deactivation request to the switch.

```
DWORD    DivaMWIDeactivate (    DivaAppHandle               hApp,
                                DWORD                       LineDeviceId,
                                DivaMWIDeactivateParams*    pParams );
```

Parameters

hApp

[in] *hApp* is the application handle that was returned by a call to *DivaRegister.*

LineDeviceId

[in] This parameter identifies the line device. Line devices are continuously numbered by an index starting with one.

pParams

[in] The parameter points to a data structure of the type *DivaMWIDeactivateParams* that contain the deactivation parameter.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter*, *DivaErrorInvalidHandle*, and *DivaErrorNotSupported*.

Remarks

The function creates a message waiting deactivation request based on the given parameter. For information on the parameter, refer to DivaMWIActivateParams. The handle given in the *DivaMWIDeactivateParams* is passed to the application with the event.

The function returns right away. If the return code is *DivaSuccess*, the action is initiated. The result of the request is signaled via the event *DivaEventMWICompleted*.

See also

DivaMWIActivate

Call properties

The call properties are available for applications that set specific information or retrieve specific information. All call properties are optional. The call properties enable a flexible development of applications and allow to extend the functionality of applications to specific environments. For a detailed list of call properties, please refer to DivaCallPropertyType.

DivaSetCallProperties

DivaSetCallProperties sets special properties of a call.

```
DWORD    DivaSetCallProperties (    DivaCallHandle          hdCall,
                                    DivaCallPropertyType    Type,
                                    DivaCallPropertyValue   *pPropertyValue,
                                    DWORD                   ProperyValueSize );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Type

[in] The *Type* parameter specifies the property to be set. See DivaCallPropertyType for available property types.

pPropertyValue

[in] The *pPropertyValue* parameter points to a location, where the property value is located. The value and the length depend on the property type. See Remarks.

PropValueSize

[in] The *PropValueSize* parameter specifies size in bytes provided for the property value. The required length depends on the property type.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter, DivaErrorReadOnlyParameter, DivaErrorDataSize,* and *DivaErrorInvalidHandle*.

Remarks

The function sets certain properties of a call. For an initial call, the function must be called prior to initiate or answer the call. For established calls the new properties have only an impact if one of the "set call type functions" is called.

The property data given to the Diva API depends on the property type. In general, this points to the type *DivaCallPropertyValue* and the length is given by the data type. In case shorter values of a simple type, i.e. Boolean, are needed, they can also be passed directly. The Diva API will always verify the given length compared to the required.

Note: Some properties are read only, they cannot be set.

The function returns right away, independent of the event mode.

See also

DivaCallPropertyValue, DivaGetCallProperties, Call properties, DivaCallPropertyType

DivaGetCallProperties

DivaGetCallProperties gets special properties of a call.

```
DWORD    DivaGetCallProperties (    DivaCallHandle         hdCall,
                                    DivaCallPropertyType   Type,
                                    DivaCallPropertyValue  *pPropertyValue,
                                    DWORD                  ProperyValueSize );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall*, *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Type

[in] The T*ype* parameter specifies the property to be set. See DivaCallPropertyType for available property types.

Note: Some properties are read only, they cannot be set.

pPropertyValue

[in] The *pPropertyValue* parameter points to a location, where the value for the requested property is placed.

PropValueSize

[in] The *PropValueSize* parameter specifies the length in bytes of the caller provided buffer.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidParameter, DivaErrorWriteOnlyParameter, DivaErrorDataSize,* and *DivaErrorInvalidHandle*.

Remarks

The function reads certain properties of a call. The available call properties are defined in *DivaCallPropertyType*; however, not all of them have the read attribute. For those with write only attributes, the return code is *DivaErrorWriteOnlyParameter.*

The property data is written to the location pointed to by *pPropertyValue*. The application provides the buffer and also the length of the buffer. The properties have different lengths and the application does not always need to provide the maximum space defined by the size of *DivaCallPropertyValue*. The Diva API will always verify the provided length compared to the required length for the specific property.

The function returns right away, independent of the event mode.

See also

DivaCallPropertyValue, DivaSetCallProperties, Call properties, DivaCallPropertyType

DivaDefaultCallProperties

DivaDefaultCallProperties sets the default properties for the given call type.

```
DWORD          DivaDefaultCallProperties (     DivaCallHandle     hdCall,
                                               DivaCallType       CallType );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall*, *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

CallType

[in] The *CallType* parameter specifies the type of the call for which the default should be set. Valid values are defined by *DivaCallType*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidParameter.*

Remarks

The function sets all call-related parameters to their defaults and sets the call type to the given value. The function is used to reset certain parameters changed by *DivaSetCallProperties* to their default.

See also

DivaSetCallProperties

Event reporting

The application can register for three different types of event reporting:

- Callback function
- Event object to be signaled

For callback function the event is provided directly. For signaling of an event object, the event has to be retrieved by a function call.

Callback function

If the application has registered for events via callback function, the following function needs to be provided by the application. The function name may be chosen by the application, the entry point of the function is given to the Dialogic® Diva® API with the registration via *DivaRegister*.

```
void        DivaCallback (    DivaAppHandle      hApp,
                              DivaEvent          Event,
                              void               *EventSpecific1,
                              void               *EventSpecific2 );
```

Parameters

hApp

[in] This parameter specifies the application instance. It is the handle returned by the Diva API during registration via *DivaRegister*.

Event

[in] The *Event* parameter specifies the event that causes the call to this function. For more information see the list of events.

EventSpecific1

[in] This parameter is event-specific. Except for signaling a new call, this parameter is usually the application handle passed into *DivaConnect*..., *DivaAnswer*..., or *DivaCreateCall*.

EventSpecific2

[in] This parameter is event-specific.

Return values

None.

Remarks

None.

See also

No references.

CallbackEx function

If the application has registered using the event mode *DivaEventModeCallbackEx*, the following function needs to be provided by the application. The function name may be chosen by the application, the entry point of the function is given to the Dialogic® Diva® API with the registration via *DivaRegister.* In addition, a context parameter is registered by the application that is signaled when the callback function is called.

```
void        DivaCallbackEx (    void*          *pContext,
                                DivaEvent      Event,
                                void           *EventSpecific1,
                                void           *EventSpecific2 );
```

Parameters

pContext

[in] This parameter has been provided by the application with the call to *DivaRegister.* The parameter is not interpreted by the Dialogic® Diva® SDK.

Event

[in] The *Event* parameter specifies the event that causes the call to this function. For more information see the list of events.

EventSpecific1

[in] This parameter is event-specific. For events related to a call, this parameter is usually the application handle passed into *DivaConnect*..., *DivaAnswer*..., or *DivaCreateCall*.

EventSpecific2

[in] This parameter is event-specific.

Return values

None.

Remarks

None.

See also

No references.

CallbackSignal function

If the application has registered using the event mode *DivaEventModeCallbackSignal*, the following function must be provided by the application. The function name may be chosen by the application, the entry point of the function is given to the Dialogic® Diva® API with the registration via *DivaRegister*. In addition, a context parameter is registered by the application that is signaled when the callback function is called.

```
void        DivaCallback (    void                *pContext );
```

Parameters

pContext

[in] This parameter has been provided by the application with the call to *DivaRegister.* The parameter is not interpreted by the Dialogic® Diva® SDK.

Return values

None.

Remarks

The SDK calls this function to notify the application that an event occurred. The event remains in the internal event queue. The application must retrieve the event using *DivaGetEvent*. Applications should always call *DivaGetEvent* in a loop until the return value shows that no more events are available.

See also

DivaGetEvent

DivaGetEvent

The *DivaGetEvent* function retrieves an event from the event queue.

```
BOOL        DivaGetEvent (   DivaAppHandle      hApp,
                             DivaEvent          *Event,
                             void               **EventSpecific1,
                             void               **EventSpecific2 );
```

Parameters

hApp

[in] The *hApp* parameter identifies the application instance. The handle is returned by *DivaRegister*.

Event

[out] This parameter is a pointer to a location that receives the event code. For more information see the list of events.

EventSpecific1

[out] This parameter is a pointer to a location that receives additional information on the event. The information depends on the event.

EventSpecific2

[out] This parameter is a pointer to a location that receives additional information on the event. The information depends on the event.

Return values

The function returns non-zero if an event is available.

Remarks

The function is used by event mechanisms that do not allow to provide the event information directly when signaling the event.

See also

No references.

Monitoring

This chapter contains various monitoring functions.

DivaCreateMonitor

DivaCreateMonitor creates an internal monitoring object and initiates the monitoring.

```
DWORD      DivaCreateMonitor (      DivaAppHandle           hApp,
                                    AppMonitorHandle        haMonitor,
                                    DivaMonitorHandle       *phdMonitor,
                                    DWORD                   LineDeviceA,
                                    DWORD                   LineDeviceB );
```

Parameters

hApp

[in] The parameter *hApp* identifies the application. The handle is assigned by a call to *DivaRegister*.

haMonitor

[in] The parameter *haMonitor* identifies the application context for the monitor object and is signaled with all monitor-related events. The handle is not interpreted by the Dialogic® Diva® API, the application is free to use any value.

phdMonitor

[out] The parameter *phdMonitor* points to a location that receives the Diva API-related handle of the monitor object. This handle has to be used in all succeeding monitor-related calls.

LineDeviceA

[in] The parameter *LineDeviceA* identifies the first line device connected to the line to be monitored. See remarks section.

LineDeviceB

[in] The parameter *LineDeviceB* identifies the second line device connected to the line to be monitored. See remarks section.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Possible other return values are *DivaErrorInvalidState*, *DivaErrorLimitExceeded*, *DivaErrorInvalidParameter*, and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaCreateMonitor* the application creates a monitor object and accesses the given line devices. The function returns right away. The start of the monitor is signaled by the event *DivaEventMonitorStatus* with the status set to DivaMonitorStarted. When the monitor object detects an active layer 1, it will signal another status event with the status set to DivaMonitorLayer1Up.

The monitor uses two line devices to record both directions of the call. The two line devices are defined by LineDeviceA and LineDeviceB. The Diva API does not know the environment and therefore cannot differentiate between incoming and outgoing calls. The line device member in the call information for a monitored call will always contain the line device that initiates the call.

See also

DivaDestroyMonitor, DivaEventMonitorStatus

DivaDestroyMonitor

DivaDestroyMonitor terminates monitoring for the given monitor handle.

```
DWORD    DivaDestroyMonitor (    DivaMonitorHandle     hdMonitor );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaDestroyMonitor* the application removes a monitoring object. This is a synchronous function and returns when the monitoring object is destroyed and all resources are freed.

See also

DivaCreateMonitor, DivaEventMonitorStatus

DivaMonitorGetCallInfo

DivaMonitorGetCallInfo retrieves information for a monitored call.

```
DWORD    DivaMonitorGetCallInfo (    DivaMonitorHandle    hdMonitor,
                                     DivaCallHandle       hdCall,
                                     DivaCallInfo         *pCallInfo );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with one of the events *DivaEventMonitorCallInitiated*, *DivaEventMonitorCallConnected*, or *DivaEventMonitorCallInfo*.

pCallInfo

[out] This parameter is a pointer to a user-supplied buffer of the type *DivaCallInfo* that receives the information on the call. Note that the application must set the *Size* field of the *DivaCallInfo* structure to the size of the structure before calling this function.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The function returns the information for a monitored call. Refer to *DivaCallInfo* for more information on the returned parameter. The application may also use *DivaMonitorGetCallProperties* to retrieve specific parameters like bearer capabilities.

See also

DivaEventMonitorCallInitiated, DivaEventMonitorCallConnected, DivaEventMonitorCallInfo, DivaMonitorGetCallProperties

DivaMonitorGetCallProperties

DivaMonitorGetCallProperties retrieves specific information for a monitored call.

```
DWORD    DivaMonitorGetCallProperties (    DivaMonitorHandle       hdMonitor,
                                            DivaCallHandle          hdCall,
                                            DivaCallPropertyType    Type,
                                            DivaCallPropertyValue   *pValue,
                                            DWORD                   PropertySize );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with one of the events *DivaEventMonitorCallInitiated*, *DivaEventMonitorCallConnected*, or *DivaEventMonitorCallInfo*.

Type

[in] The *Type* parameter specifies the property to be set. See DivaCallPropertyType for available property types.

pValue

[in] The *pValue* parameter points to a location, where the value for the requested property is placed.

PropertySize

[in] The *PropertySize* parameter specifies the length in bytes of the buffer provided by the caller.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The function reads certain properties of a call. The available call properties are defined in *DivaCallPropertyType*; however, not all of them have the read attribute. For those with write only attributes, the return code is *DivaErrorWriteOnlyParameter.*

The property data is written to the location pointed to by *pPropertyValue*. The application provides the buffer and also the length of the buffer. The properties have different lengths and the application does not always need to provide the maximum space defined by the size of *DivaCallPropertyValue*. The Dialogic® Diva® API always compares the provided length with the required length for the specific property.

The function returns right away, independent of the event mode.

See also

DivaEventMonitorCallInitiated, DivaEventMonitorCallConnected, DivaEventMonitorCallInfo, DivaMonitorGetCallInfo

DivaMonitorGetSetupMessage

DivaMonitorGetSetupMessage retrieves the setup message that belongs to the monitored call.

```
DWORD    DivaMonitorGetSetupMessage (    DivaMonitorHandle      hdMonitor,
                                         DivaCallHandle         hdCall,
                                         unsigned char          *pBuffer,
                                         DWORD                  BufferLength,
                                         DWORD                  *pBytesUsed );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with one of the events *DivaEventMonitorCallInitiated*, *DivaEventMonitorCallConnected*, or *DivaEventMonitorCallInfo*.

pBuffer

[out] The parameter specifies a location to which the setup message is written. The length of the buffer is described by the parameter *BufferLength*.

BufferLength

[in] The parameter specifies the length of the buffer provided by the caller.

pBytesUsed

[in] The parameter specifies a location where the amount of bytes, written to the user provided buffer, is placed.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The function provides the raw setup message (layer 3) to the caller. The setup message is available from the first event-related to this call.

See also

DivaEventMonitorCallInitiated, DivaEventMonitorCallConnected, DivaEventCallInfo, DivaMonitorGetCallProperties

DivaMonitorCloseCallHandle

DivaMonitorCloseCallHandle frees a call handle and resources bound to this handle.

```
DWORD    DivaMonitorCloseCallHandle (    DivaMonitorHandle      hdMonitor,
                                         DivaCallHandle         hdCall );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with one of the *DivaEventMonitorCallInitiated*, *DivaEventMonitorCallConnected*, or *DivaEventMonitorCallInfo*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

When a monitoring call is disconnected, the application might want to read properties from the call. Therefore, the Dialogic® Diva® API keeps the information on this call until the application calls *DivaMonitorCloseCallHandle*. If this is not called, the Dialogic® Diva®SDK will not release the call-related resources.

See also

DivaEventMonitorCallInitiated, DivaEventMonitorCallConnected, DivaEventMonitorCallInfo

DivaMonitorRecordAudio

DivaMonitorRecordAudio starts the recording on a monitored call.

```
DWORD    DivaMonitorRecordAudio (         DivaMonitorHandle       hdMonitor,
                                          DivaCallHandle          hdCall,
                                          char                    *pFilename,
                                          DivaAudioFormat         Format,
                                          DivaMonitorSource       Source );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call to monitor. The handle is passed by the event *DivaEventMonitorCallConnected*.

pFilename

[in] The parameter *pFilename* identifies the path and name of the file to store the audio data.

Format

[in] The parameter *Format* specifies the format of the audio. See DivaAudioFormat for available formats. If the monitoring source is set to *DivaMonitorSourceBoth*, only the wave file formats are valid.

Source

[in] The parameter *Source* specifies if both directions of the call should be recorded.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaMonitorRecordAudio* the recording to a file is started. The application defines the format and the directions to record. The direction is either given as physical information (from line device A to B) or related to the call direction (originator or answerer).

The parameter *Source* specifies what to record. Options are defined in *DivaMonitorSource*. If the source is specified to *DivaMonitorSourceBoth*, the data is written to a stereo wave file.

See also

DivaMonitorStopAudio

DivaMonitorStopAudio

DivaMonitorStopAudio terminates the recording on a monitored call.

```
DWORD    DivaMonitorStopAudio (          DivaMonitorHandle       hdMonitor,
                                         DivaCallHandle          hdCall,
                                         DivaMonitorSource       Source );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call to monitor. The handle is passed by the event *DivaEventMonitorCallConnected*.

Source

[in] The parameter *Source* specifies which recording should be stopped. If set to *DivaMonitorSourceBoth* any recording is stopped. Otherwise the specific recording direction is stopped. Note that this must match the previously initiated recording.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaMonitorRecordStop* the recording to a file terminated. The event *DivaEventMonitorRecordEnded* signals that the Dialogic® Diva® SDK has ended recording and the file is no longer accessed by the Diva SDK.

See also

DivaMonitorRecordAudio

DivaMonitorSetVolume

DivaMonitorSetVolume changes the volume for a monitored data channel.

```
DWORD      DivaMonitorSetVolume (        DivaMonitorHandle       hdMonitor,
                                         DivaCallHandle          hdCall,
                                         DivaVolume              Volume,
                                         DivaMonitorSource       Source );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call to monitor. The handle is passed by the event *DivaEventMonitorCallConnected*.

Volume

[in] The parameter *Volume* specifies the new volume in the range defined by *DivaVolume*.

Source

[in] The parameter *Source* specifies for which recording the volume is to be changed. See DivaMonitorRecordAudio for more information.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The Dialogic® Diva® API allows to control the volume in the range of -18 db to +18 db. The volume can be controlled individually for each direction.

Note: This is a setting of the volume, not an automatic gain control.

See also

DivaMonitorRecordAudio

DivaMonitorEnableAudioData

DivaMonitorEnableAudioData changes the signaling of audio data in passive monitoring mode.

```
DWORD    DivaMonitorEnableAudioData (    DivaMonitorHandle    hdMonitor,
                                         DivaCallHandle       hdCall,
                                         DWORD                AudioBuffers );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call to monitor. The handle is passed by the Dialogic® Diva® API with the event *DivaEventCallConnected*.

AudioBuffers

[in] The parameter *AudioBuffers* specifies the amount of buffers used for storing the recorded audio. See remarks.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidParameter* and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaMonitorEnableAudioData* the application enables audio data for the monitored call to be passed to the application. The function is independent from recording to a file and can be activated at any time.

Once audio data is available, the event *DivaMonitorAudioData* is signaled to the application. The application must retrieve the data via *DivaMonitorReceiveAudio*.

The parameter *AudioBuffers* specifies how many buffers are stored by the application until data is lost. For performance reasons, it is recommended to set this parameter to 1 and process the audio directly from the callback function.

See also

DivaMonitorDisableAudioData, DivaEventMonitorAudioData, DivaMonitorReceiveAudio

DivaMonitorDisableAudioData

DivaMonitorDisableAudioData changes the signaling of audio data in passive monitoring mode.

```
DWORD     DivaMonitorDisableAudioData (          DivaMonitorHandle          hdMonitor,
                                                 DivaCallHandle             hdCall );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call to monitor. The handle is provided by the Dialogic® Diva® API with the event *DivaEventCallConnected*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The function stops the reporting of monitored audio to the application. A pending recording to a file will continue unchanged.

See also

DivaMonitorEnableAudioData, DivaEventMonitorAudioData, DivaMonitorReceiveAudio

DivaMonitorReceiveAudio

DivaMonitorReceiveAudio retrieves monitored audio signal to a memory location given by the caller.

```
DWORD     DivaMonitorReceiveAudio (          DivaMonitorHandle          hdMonitor,
                                             DivaCallHandle             hdCall,
                                             DivaMonitorSource          Source,
                                             unsigned char              *pBuffer,
                                             DWORD                      BufferSize,
                                             DWORD                      *pBytesWritten,
                                             DivaAudioFormat            Format );
```

Parameters

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hdCall

[in] The parameter *hdCall* identifies the call to monitor. The handle is passed by the Dialogic® Diva® API with the event *DivaEventCallConnected*.

Source

[in] The parameter *Source* identifies for which direction the audio signal should be retrieved. The option *DivaMonitorSourceBoth* provides the mixed audio stream.

pBuffer

[out] The parameter *pBuffer* specifies the location where the received audio data should be written.

BufferSize

[in] The parameter *BufferSize* specifies the length of the buffer in bytes.

pBytesWritten

[out] The parameter *pBytesWritten* points to a location of type DWORD where the amount of bytes written to the buffer is placed.

Format

[in] The parameter *Format* specifies the audio format for which the application requests the data. Possible options are the raw formats of DivaAudioFormat.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidHandle* and *DivaErrorInvalidParameter*.

Remarks

The function retrieves received audio information, converts it to the requested audio format and writes it to the given buffer. The supported audio formats are the raw formats defined in DivaAudioFormat.

Every time the event *DivaMonitorEventAudioData* is signaled an audio buffer for both directions (caller to callee and vice versa) is available.

If the format *DivaMonitorSourceBoth* is specified, the audio from both directions will be mixed into one audio stream.

See also

DivaMonitorEnableAudioData, DivaEventMonitorAudioData, DivaMonitorDisableAudioData

Audio provider

This chapter contains various audio functions.

DivaRegisterAudioProvider

DivaRegisterAudioProvider registers an audio provider with the Dialogic® Diva® SDK.

```
DWORD    DivaRegisterAudioProvider (    DivaAppHandle       *pHandle,
                                        char                *Providername,
                                        DivaAPNotifyCall    pfnNotifyCall );
```

Parameters

pHandle

[out] The parameter *pHandle* points to a location that receives the handle for the audio provider registration.

Providername

[in] The parameter *Providername* identifies the audio provider. This is done by a symbolic name. The application uses the same name when attaching audio providers to data channels.

pfnNotifyCall

[out] The parameter *pfnNotifyCall* is the function entry of type *APNotifyCall* provided by the audio provider. The function is called to create a link between a call and the audio channel.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). If an audio provider with the same name is already registered, the function returns *DivaErrorAlreadyAssigned*.

Remarks

With a call to *DivaRegisterAudioProvider*, the application creates an instance at the Dialogic® Diva® API for the audio provider. The symbolic provider name and the notify function are stored for further requests. For more information on the function, refer to *APNotifyCall*.

See also

DivaReleaseAudioProvider, APNotifyCall, DivaConnectAudioProvider, DivaDisconnectAudioProvider

DivaReleaseAudioProvider

DivaReleaseAudioProvider releases a previously done registration of an audio provider.

```
DWORD    DivaReleaseAudioProvider (     DivaAppHandle       hApp );
```

Parameters

hApp

[in] The parameter *hApp* identifies the audio provider instance. The handle has been returned by a previous call to *DivaRegisterAudioProvider*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaReleaseAudioProvider* the instance at Dialogic® Diva® SDK level is removed. All streaming operations assigned to the audio provider are implicitly removed.

See also

DivaRegisterAudioProvider

DivaConnectAudioProvider

DivaConnectAudioProvider attaches an audio provider to the data channels of an existing call.

```
DWORD    DivaConnectAudioProvider (    DivaCallHandle    hdCall,
                                       char              *Providername,
                                       DivaIdDescriptor  *pDeviceId,
                                       DivaAPMode        WhatToConnect );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the call at the Dialogic® Diva® SDK. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Providername

[in] The parameter *Providername* identifies the audio provider. This is the same name that the audio provider used during registration.

pDeviceId

[out] The parameter *pDeviceId* defines the logical channel at the audio provider. The format of this identifier depends on the used ASR / TTS engine. The Diva SDK routes this parameter to the audio provider without any further processing.

WhatToConnect

[in] The parameter *WhatToConnect* defines in which direction the streaming should be done.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). If the handles cannot be assigned or the provider name is not valid, the function returns *DivaErrorInvalidHandle*. If the audio provider rejects the notification, the function returns *DivaErrorNoChannel*.

Remarks

With the call *DivaConnectAudioProvider*, the application, which controls the call, attaches the audio channel to an audio provider. The Diva SDK identifies the audio provider by the symbolic name.

The Diva SDK notifies the audio provider by calling *APNotifyCall*. The following communication between the audio provider and the Diva SDK is done via function entries exchanged during notification.

See also

DivaRegisterAudioProvider, APNotifyCall, DivaDisconnectAudioProvider

DivaDisconnectAudioProvider

DivaDisonnectAudioProvider removes an audio provider from the audio channel of the given call.

```
DWORD    DivaDisconnectAudioProvider (    DivaCallHandle    hdCall,
                                          char              *Providername,
                                          DivaAPMode        WhatToDisconnect );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the call at the Dialogic® Diva® SDK. The handle is either returned by *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Providername

[in] The parameter *Providername* identifies the audio provider. This is the same name that the audio provider used during registration.

WhatToDisconnect

[in] The parameter *WhatToDisonnect* defines in which direction the streaming should be removed.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). If the handles cannot be assigned or the provider name is not valid, the function returns *DivaErrorInvalidHandle*.

Remarks

With a call *DivaDisconnectAudioProvider,* the application that controls the call removes the data channel from the audio provider. The Diva SDK identifies the audio provider by the symbolic name.

The application controls the streaming direction to connect or disconnect. The assignment can be changed at any time. An application may switch the send direction to stream TTS or plain audio while the receive direction remains unchanged.

See also

DivaRegisterAudioProvider, APNotifyCall, DivaConnectAudioProvider

DivaAPSendAudio

DivaAPSendAudio is a function entry point provided by the Dialogic® Diva® SDK for streaming audio from an audio provider.

```
DWORD    (* DivaAPSendAudio (        DivaAppHandle       hApp,
                                     DivaCallHandle      hdCall,
                                     unsigned char       *pData,
                                     DWORD               Length,
                                     DivaAudioFormat     Format );
```

Parameters

hApp

[in] The parameter *hApp* identifies the audio provider instance. The handle has been assigned by *DivaRegisterAudioProvider.*

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with the call to *APNotifyCall*.

pData

[in] The parameter *pData* points to a location that contains the audio data to be streamed.

Length

[in] The parameter *Length* specifies the amount of data to be sent.

Format

[in] The parameter *Format* specifies the audio coding format of the given data.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The entry point of this function is provided by the Diva SDK and is exchanged when a call is notified to the audio provider. The audio provider calls this function to stream audio data from the TTS engine. The data buffer provided by the audio provider is owned by the Diva SDK until the confirmation function of the audio provider is called.

See also

DivaRegisterAudioProvider, APNotifyCall, APConfirmAudioSend, DivaConnectAudioProvider

DivaAPStopSendAudio

DivaAPStopSendAudio is a function entry point provided by the Dialogic® Diva® SDK for interrupting the streaming audio from an audio provider.

```
DWORD     (* DivaAPStopSendAudio (       DivaAppHandle        hApp,
                                         DivaCallHandle       hdCall );
```

Parameters

hApp

[in] The parameter *hApp* identifies the audio provider instance. The handle has been assigned by *DivaRegisterAudioProvider*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with the call to *APNotifyCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible other return value is *DivaErrorInvalidHandle*.

Remarks

The entry point of this function is provided by the Diva SDK and is exchanged when a call is notified to the audio provider. The audio provider calls this function to stop the streaming of audio data previously initiated by *DivaAPSendAudio*.

See also

DivaRegisterAudioProvider, APNotifyCall, DivaAPSendAudio, DivaConnectAudioProvider

DivaAPSetRecordFormat

DivaAPSetRecordFormat is a function entry point provided by the Dialogic® Diva® SDK for setting the audio format for the ASR engine.

```
DWORD     (* DivaAPSetRecordFormat (       DivaAppHandle        hApp,
                                           DivaCallHandle       hdCall,
                                           DivaAudioFormat      Format );
```

Parameters

hApp

[in] The parameter *hApp* identifies the audio provider instance. The handle has been assigned by DivaRegisterAudioProvider.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with the call to *APNotifyCall*.

Format

[in] The parameter *Format* specifies the audio format. Valid formats are the raw formats defined on *DivaAudioFormat*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidHandle* or *DivaErrorInvalidParameter.*

Remarks

The entry point of this function is provided by the Diva SDK and is exchanged when a call is notified to the audio provider. The audio provider may set the audio format at any time. The format is valid with the next audio data signaled to the audio provider via *APNotifyReceiveAudio*.´

See also

DivaRegisterAudioProvider, APNotifyReceiveAudio, DivaAudioFormat, DivaConnectAudioProvider

DivaAPSetVolume

DivaAPSetVolume is a function entry point provided by the Dialogic® Diva® SDK for setting the volume for received and sent audio.

```
DWORD     (* DivaAPSetVolume (          DivaAppHandle       hApp,
                                        DivaCallHandle      hdCall,
                                        DivaVolume          Volume,
                                        DivaDirection       Direction );
```

Parameters

hApp

[in] The parameter *hApp* identifies the audio provider instance. The handle has been assigned by *DivaRegisterAudioProvider*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with the call to *APNotifyCall*.

Volume

[in] The parameter *Volume* specifies the volume to be set. Valid formats are in the range defined by *DivaVolume*.

Direction

[in] The parameter *Direction* specifies if the volume should be set for receive and/or sent. For valid formats please refer to DivaDirection.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Other possible return values are *DivaErrorInvalidHandle* or *DivaErrorInvalidParameter.*

Remarks

The entry point of this function is provided by the Diva SDK and is exchanged when a call is notified to the audio provider. The audio provider may set the audio format at any time. The format is valid with the next audio data signaled to the audio provider via *APNotifyReceiveAudio*.

See also

DivaRegisterAudioProvider, APNotifyReceiveAudio, DivaDirection, DivaConnectAudioProvider

DivaAPCloseAudio

DivaAPCloseAudio is a function entry point provided by the Dialogic® Diva® SDK for closing a logical link between audio provider and Diva SDK.

```
DWORD     (* DivaAPCloseAudio (          DivaAppHandle        hApp,
                                         DivaCallHandle       hdCall );
```

Parameters

hApp

[in] The parameter *hApp* identifies the audio provider instance. The handle has been assigned by *DivaRegisterAudioProvider*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with the call to *APNotifyCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The entry point of this function is provided by the Diva SDK and is exchanged when a call is notified to the audio provider. The audio provider calls this function if the logical instance at the audio provider is no longer valid. The Diva SDK will not call any more function entries related to this call.

See also

DivaRegisterAudioProvider, APNotifyCall, DivaConnectAudioProvider

APNotifyCall

APNotifyCall is provided by the audio provider. The name of the function can be different, and the function entry point is exchanged during registration.

```
BOOL       APNotifyCall (    DivaAppHandle                hApp,
                             DivaAPNotifyCallInParams     *pInParams,
                             DivaAPNotifyCallOutParams    **pOutParams );
```

Parameters

hApp

[in] The parameter *hApp* identifies the audio provider instance. This is the same handle returned by a previous call to *DivaRegisterAudioProvider*.

pInParams

[in] The parameter *pInParams* is of the type *DivaAPNotifyCallInParams* and contains the input parameter.

pOutParams

[out] The parameter *pOutParams* is of the type *DivaAPNotifyCallOutParams* and contains the output parameter.

Return values

The function returns TRUE if the audio provider has assigned the call based on the input parameter. It returns FALSE if the identifier is unknown.

Remarks

The function is provided by the audio provider. Based on the identifier in the input parameters, the call is assigned. The Dialogic® Diva® SDK and the audio provider exchange handles for identification of the streaming channel and several function pointers to exchange the audio and control information. The Diva SDK provides the following functions in the input parameter:

- DivaAPSendAudio

- DivaAPStopSendAudio
- DivaAPSetRecordFormat
- DivaAPCloseAudio

The audio provider places the following functions in the output parameter:

- APNotifyCallClose
- APNotifyReceiveAudio (only if ASR supported)
- APConfirmAudioSend (only if TTS supported)

See also

DivaRegisterAudioProvider, DivaReleaseAudioProvider, DivaConnectAudioProvider, DivaDisconnectAudioProvider

APNotifyCallClose

APNotifyCallClose is a function entry point provided by the audio provider. The name of the function can be different, the function entry point is exchanged during notification via *APNotifyCall*.

```
DWORD    (* DivaAPNotifyCallClose (      AppCallHandle        hAPCall );
```

Parameters

hAPCall

[in] The parameter *hAPCall* identifies the instance or channel at the audio provider. The handle has been given to the Dialogic® Diva® SDK in the output parameters during *APNotifyCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The entry point of this function is provided by the audio provider. The Diva SDK calls this function when a call is disconnected and a link previously initiated by *DivaConnectAudioProvider* exists. The audio provider must stop to call any entry points at the Diva SDK exchanged during *APNotifyCall*.

See also

DivaRegisterAudioProvider, APNotifyCall, DivaConnectAudioProvider

APNotifyReceiveAudio

APNotifyReceiveAudio is a function entry point provided by the audio provider. The name of the function can be different, and the function entry point is exchanged during notification via *APNotifyCall*.

```
DWORD    (* DivaAPNotifyReceiveAudio (   AppCallHandle        hAPCall,
                                         unsigned char        *pData,
                                         DWORD                Length );
```

Parameters

hAPCall

[in] The parameter *hAPCall* identifies the instance or channel at the audio provider. The handle has been given to the Dialogic® Diva® SDK in the output parameters during *APNotifyCall*.

pData

[in] The parameter *pData* identifies the location where the Diva SDK provides the received audio stream.

Length

[in] The parameter *Length* specifies the amount of bytes available at pData.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The entry point of this function is provided by the audio provider. The Diva SDK calls this function for each data block if a link for received audio has been created by *DivaConnectAudioProvider*.

The format of the audio stream has been selected by the audio provider. The audio provider has to process the data to this function during the call. When the function returns, the data is no longer valid.

See also

DivaRegisterAudioProvider, APNotifyCall, DivaConnectAudioProvider, DivaAPSetRecordFormat

APConfirmAudioSend

APConfirmAudioSend is a function entry point provided by the audio provider. The name of the function can be different, and the function entry point is exchanged during notification via *APNotifyCall*.

```
DWORD    (* APConfirmAudioSend (       AppCallHandle        hAPCall );
```

Parameters

hAPCall

[in] The parameter *hAPCall* identifies the instance or channel at the audio provider. The handle has been given to the Dialogic® Diva® SDK in the output parameters during APNotifyCall.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

The entry point of this function is provided by the audio provider. The Diva SDK calls this function when a data buffer, sent by *DivaAPSendAudio,* has been processed, and the buffer is free.

There is no handle for buffers to be confirmed. The Diva SDK ensures that buffers are processed and confirmed in the order they were passed by the audio provider.

See also

DivaRegisterAudioProvider, APNotifyCall, DivaConnectAudioProvider, DivaAPSendAudio

Timer Handling

This chapter contains various timer handling functions.

DivaStartCallTimer

DivaStartCallTimer starts a single shot timer based on a call object.

```
DWORD     DivaStartCallTimer (          DivaCallHandle        hdCall,
                                        DWORD                 WaitTime );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

WaitTime

[in] The parameter *WaitTime* specifies the time, in milliseconds, when the timer will be fired. The minimum value is 100 milliseconds. The timer resolution is 100 milliseconds.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaStartCallTimer* the application registers for a timer event based on the call object. When the timer expires, the event *DivaEventCallTimer* is signaled to the applications event registration, e.g., the callback function. The timer is a single shot timer, and the application may restart the timer directly from the event handler.

If a call object is closed, a pending timer is silently discarded.

See also

DivaStopCallTimer, DivaStartApplicationTimer, DivaEventCallTimer

DivaStopCallTimer

DivaStopCallTimer stops a time initiated via *DivaStartCallTimer*.

```
DWORD     DivaStopCallTimer (           DivaCallHandle        hdCall );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the call at the Dialogic® Diva® API. The handle is either returned by *DivaCreateCall* or *DivaConnect* or signaled with the event *DivaEventIncomingCall*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaStopCallTimer* a timer started with *DivaStartCallTimer* is cleared.

See also

DivaStartCallTimer, DivaStartApplicationTimer, DivaStopApplicationTimer, DivaEventCallTimer

DivaStartApplicationTimer

DivaStartApplicationTimer starts a single shot timer based on an a registered application.

```
DWORD   DivaStartApplicationTimer (   DivaAppCallHandle   Handle,
                                      DWORD               WaitTime );
```

Parameters

Handle

[in] The application *Handle* that was returned by a call to *DivaRegister*.

WaitTime

[in] The parameter *WaitTime* specifies the time, in milliseconds, when the timer will be fired. The minimum value is 100 milliseconds. The timer resolution is 100 milliseconds.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaStartApplicationTimer* the application registers for a timer event. When the timer expires, the event *DivaEventApplicationTimer* is signaled to the applications event registration, e.g., the callback function. The timer is a single shot timer, the application may restart the timer directly from the event handler.

An application can only run one timer of the type "DivaApplicationTimer" at a time. Any pending timer is cleared by another call to *DivaStartApplicationTimer*.

See also

DivaStopApplicationTimer, DivaStartCallTimer, DivaEventApplicationTimer

DivaStopApplicationTimer

DivaStopApplicationTimer stops a timer started by a call to *DivaStartApplicationTimer*.

```
DWORD   DivaStopApplicationTimer (    DivaAppHandle       Handle );
```

Parameters

Handle

[in] The application *Handle* that was returned by a call to *DivaRegister*.

Return values

If the function succeeds, the return value is *DivaSuccess* (0). Another possible return value is *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaStopApplicationTimer* a timer started with a call to *DivaStartApplicationTimer* is stopped.

See also

DivaStartApplicationTimer, DivaStartCallTimer, DivaEventApplicationTimer

Tracing

The Dialogic® Diva® SDK can be enabled to write trace information to a file. The general trace settings are done by the CONFIG.EXE. The application may change these settings and select a different file or change the trace level. The following functions are now available.

DivaEnableTrace

DivaEnableTrace controls the trace level at runtime.

```
BOOL     DivaEnableTrace (     DWORD        nLevel );
```

Parameter

nLevel

[in] This parameter sets the level of tracing.

Returns

The function returns TRUE if the level has been changed. If an invalid level is passed, the function returns FALSE.

See also

DivaTraceLevel

DivaSetTraceFile

DivaSetTraceFile allows the application to specify the trace file name and location as well as the maximum size.

```
BOOL     DivaSetTraceFile (    char *       pFilename,
                               DWORD        dwMaxSize );
```

Parameter

pFilename

[in] This parameter specifies the path and file name of the new trace file.

dwMaxSize

[in] This parameter specifies the maximum size of the trace file.

Returns

The function returns TRUE if the new trace file has been accepted. If the file could not be opened, the function returns FALSE and the old trace file is used.

Remarks

The function requires a trace level higher than DivaTraceLevelNothing to be active. If the application does not know which level is active, set the trace level using *DivaEnableTrace* before setting the trace file.

See also

DivaEnableTrace

DivaLogPrintf

DivaLogPrintf writes trace messages into the Dialogic® Diva® SDK trace system.

```
void        DivaLogPrintf (     const char *        strApplication,
                                const char *        strModule,
                                const char *        strType,
                                const char *        strFormat );
```

Parameter

strApplication

The parameter specifies a short name to identify the application. The name is limited to three characters.

strModule

The parameter specifies the module that issues the message.

strType

The parameter specifies a type of the message, e.g., error or warning. The type is limited to three characters.

strFormat

Contains the message and format information. Syntax for the format information is the same as for printf.

Returns

None

Remarks

The function allows an application to trace into the standard Diva SDK trace environment.

See also

No references.

CHAPTER 5

Dialogic® Diva® API Events

The following events are reported by the Dialogic® Diva® API. The parameters delivered with the event are event specific. Most events pass the application call handle with the event. If the application has not set the call handle, the value INVALID_APP_CALL_HANDLE is used.

Please note that each event may be silently discarded by the application. No resources are bound to an event in the Diva API. An exception is the event *DivaEventCallDisconnected*, which must be processed in order to free resources bound to a call. Resources are set free by calling *DivaCloseCall*.

DivaEventIncomingCall

The *DivaEventIncomingCall* event signals a new incoming call. The *EventSpecific1* parameter contains the Dialogic® Diva® API call handle of this call. The application can retrieve more information on this call by calling DivaGetCallInfo with the given handle. The application can answer the call by calling DivaAnswer, DivaAnswerFax, or DivaAnswerVoice. If more information is needed, the application must attach a call handle by calling DivaAttachToCall. If the application is not able to service a call, DivaReject must be called.

DivaEventCallInfo

The *DivaEventCallInfo* event is signaled when new information for the call is available since the last call to *DivaGetCallInfo*. The *EventSpecific1* parameter contains the call handle of the application. The application may obtain more information on this call by calling DivaGetCallInfo.

This event is signaled when the content of one of the DivaCallInfo members has changed. An exception are the members that own a separate event, for example *CallState* which is reported by *DivaEventCallProgress*.

DivaEventCallProgress

The *DivaEventCallProgress* event signals the progress of an incoming or outgoing call. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the new call state. The application may obtain more information on this call by calling DivaGetCallInfo.

This event is available in addition to the *DivaEventCallConnected* and *DivaEventCallDisconnected* events to provide information on different steps of the call setup. The call state is one of the states defined in DivaCallState.

The event is signaled whenever the state of a call is changed. Other call-related information, e.g., disconnect reasons may not be updated at the time the event is received. The update of this kind of information is signaled with other events, e.g., *DivaEventCallInfo* or *DivaEventCallDisconnected*.

DivaEventCallConnected

The *DivaEventCallConnected* event signals that an incoming or outgoing call is physically connected and data transfer can be done on this call. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventCallDisconnected

The *DivaEventCallDisconnected* event reports the final disconnect of a call. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the call handle of the Dialogic® Diva® API. This parameter should be used to close the call via DivaCloseCall.

Please note that there is no application-specific call handle and no disconnect event is sent if the application has rejected a call.

DivaEventEarlyDataChannelConnected

The *DivaEventEarlyDataChannelConnected* event reports that voice information can be streamed. The physical connection may not have reached the connected state at this moment. The event is only signaled for outgoing calls that are established with *DivaVoiceOptionEarlyDataChannel* set in DivaVoiceOptions or for incoming calls that are not answered but the call type is set to voice using the *DivaVoiceOptionEarlyDataChannel* option*.* The *EventSpecific1* parameter contains the call handle of the application.

DivaEventSendVoiceDone

The *DivaEventSendVoiceDone* event signals the streaming of the audio data is completed. The *EventSpecific1* parameter contains the call handle of the application.

Please note that this event is also sent at the end of the file when continuous playing of the file has been selected. This event is obsolete. It is recommended to use the events DivaEventSendVoiceEnded, DivaEventSendVoiceRestarted, and DivaEventSendVoiceCanceled.

DivaEventSendVoiceEnded

The *DivaEventSendVoiceEnded* event signals that the streaming of the audio data is completed. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the reason for the termination. For valid reasons, see DivaSendVoiceEndReasons.

DivaEventSendVoiceRestarted

The *DivaEventSendVoiceRestarted* event signals that the continuous streaming of the audio data is restarted. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventSendVoiceCanceled

The *DivaEventSendVoiceCanceled* event signals that the streaming of the audio data is terminated by the application. Any resources used by the Dialogic® Diva® SDK for the streaming are no longer used. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventRecordVoiceEnded

The *DivaEventRecordVoiceEnded* event signals that the recording of the audio data is terminated. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the reason for the termination. For valid reasons, see DivaRecordEndReasons.

DivaEventFaxPageSent

The *DivaEventFaxPageSent* event is signaled when a page has been sent. The page number is provided with the event. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventFaxSent

The *DivaEventFaxSent* event is signaled when all pages have been sent. At this moment the remote side may automatically disconnect. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventFaxPageReceived

The *DivaEventFaxPageReceived* event is signaled when a page has been received. The event contains the information about the page index. It does not indicate if more pages follow. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventFaxReceived

The *DivaEventFaxReceived* event is signaled when all pages of a fax have been received. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventFaxPageSent

The *DivaEventFaxPageSent* event is signaled when a page has been sent. The *EventSpecific1* parameter contains the call handle of the application. The application must call DivaGetCallInfo to retrieve the number of pages currently sent. Additional fax-related parameters such as speed may have changed.

DivaEventFaxPageReceived

The *DivaEventFaxPageReceived* event is signaled when a page has been received. It does not indicate if more pages follow. The *EventSpecific1* parameter contains the call handle of the application. The application must call DivaGetCallInfo to retrieve the number of pages currently received. Additional fax-related parameters such as speed may have changed.

DivaEventDataAvailable

The *DivaEventDataAvailable* event is signaled when new data is available. The application has to retrieve the data using DivaReceiveData.

The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the amount of available data.

DivaEventDataSent

The *DivaEventDataSent* event is signaled when the data passed for sending has been sent and the buffer is free to be used by the application.

The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the data handle from the corresponding call to DivaSendData.

Note: The event might be signaled before the call to *DivaSendData* returns.

DivaEventDTMFReceived

The *DivaEventDTMFReceived* event is signaled when a DTMF tone is detected. The detection must be enabled by calling DivaReportDTMF.

The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the signaled DTMF tone.

DivaEventHoldCompleted

The *DivaEventHoldCompleted* event is signaled as a result of a hold request initiated by DivaHold. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the result of the request. If the call has reached the hold state, the result is *DivaSuccess*. In case of an error, the result is *DivaErrorNotSupported*.

DivaEventRetrieveCompleted

The *DivaEventRetrieveCompleted* event is signaled as a result of a retrieve request initiated by DivaRetrieve. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the result of the request. If the call has been successfully retrieved and reached the connected state, the result is *DivaSuccess*. In case of an error, the result is *DivaErrorNotSupported*.

DivaEventSetupTransferCompleted

The *DivaEventSetupTransferCompleted* event is signaled as a result of the request to transfer a call with DivaSetupCallTransfer. The *EventSpecific1* parameter contains the call handle of the primary call. The *EventSpecific2* parameter contains the result of the request. In case of success, the result code is *DivaSuccess*. Other possible values are *DivaErrorNotSupported* and *DivaErrorIvalidState*.

DivaEventTransferCompleted

The *DivaEventTransferCompleted* event is signaled as a result of a transfer request initiated by DivaCompleteCallTransfer or DivaBlindCallTransfer. The *EventSpecific1* parameter contains the call handle of the primary call. The *EventSpecific2* parameter contains the result of the request. In case of success, the result code is *DivaSuccess*. Other possible values are:

- *DivaErrorDestBusy*: This error can only occur during a blind transfer. The consultation call failed due to a busy condition.
- *DivaErrorUnallocatedNumber*: This error can only occur during a blind transfer. The consultation call failed, the switch reported that the dialed number is invalid.
- *DivaErrorNotSupported*: The requested function is not supported. This might be the hold request or the transfer itself.

DivaEventSendToneEnded

The *DivaEventSendToneEnded* event is signaled when the streaming of a tone or of the last tone of a sequence initiated by DivaSendTone or DivaSendContinuousTone has ended. *DivaSendContinuousTone* may end due to a timeout or a call to DivaStopContinuousTone. When this event occurs, all data-related to this tone have been streamed.

The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter is set to zero.

DivaEventSendDTMFToneEnded

The *DivaEventSendDTMFToneEnded* event is signaled when the streaming of a DTMF tone or of the last tone of a sequence initiated by DivaSendDTMF has ended. When this event occurs all data-related to this tone have been streamed.

The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter is set to zero.

DivaEventToneDetected

The *DivaEventToneDetected* event is signaled when the start or stop of a continuous tone or a multi-frequency tone is detected. The detection must be enabled by calling DivaReportDTMF.

The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the signaled tone. The continuous tones are defined in DivaContinuousTones, the multi-frequency tones in DivaMultiFrequencyTones.

DivaEventConferenceInfo

The *DivaEventConferenceInfo* event is signaled when new information concerning the conference is available. The application can retrieve the conference information by calling DivaGetConferenceInfo.

The *EventSpecific1* parameter contains the conference handle of the application passed to the Dialogic® Diva® API when the conference has been created.

DivaEventLIConnectCompleted

The *DivaEventLIConnectCompleted* event is signaled when Line Interconnect has been established. Line Interconnect must be initiated by DivaLIConnect.

The *EventSpecific1* parameter contains the call handle of the application for this call. On success, the *EventSpecific2* parameter contains the call handle of the interconnected call. If Line Interconnect fails, the *EventSpecific2* parameter is set to INVALID_APP_CALL_HANDLE.

DivaEventLIDisconnected

The *DivaEventLIDisconnected* event is signaled when Line Interconnect has been released. Line Interconnect must be initiated by DivaLIDisconnect. This event is also sent if the interconnected call is disconnected for some reason.

The *EventSpecific1* parameter contains the call handle of the application for this call. On success, the *EventSpecific2* parameter contains the call handle of the previously interconnected call.

DivaEventMonitorStatus

The event *DivaEventMonitorStarted* is signaled when the status of a monitor object created by DivaCreateMonitor has changed.

The *parameter EventSpecific1* contains the monitor handle of the application passed to the Dialogic® Diva® API when the monitor is created. The parameter *EventSpecific2* contains the new status. See the DivaMonitorStatus for possible status messages.

DivaEventMonitorCallInitiated

The event *DivaEventMonitorCallInitiated* is signaled when a setup message is seen on the monitored line.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the Dialogic® Diva® API when the monitor is created. The parameter *EventSpecific2* contains the handle of a call object. The call object can be used to retrieve information by calling DivaGetCallInfo or DivaGetCallProperties.

DivaEventMonitorCallConnected

The event *DivaEventMonitorCallConnected* is signaled when a call is established at D-channel level.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the Dialogic® Diva® API when the monitor is created. The parameter *EventSpecific2* contains the handle of a call object. The call object can be used to retrieve information by calling DivaGetCallInfo or DivaGetCallProperties.

DivaEventMonitorCallDisconnected

The event *DivaEventMonitorCallDisconnected* is signaled when a call is disconnected at D-channel level.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the Dialogic® Diva® API when the monitor is created. The parameter *EventSpecific2* contains the handle of a call object. The call object can be used to retrieve disconnect information by calling DivaGetCallInfo or DivaGetCallProperties.

DivaEventMonitorCallInfo

The event *DivaEventMonitorCallInfo* is signaled when call information has been changed, e.g., the state of a call changed or additional information is received as info messages.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the Dialogic® Diva® API when the monitor is created. The parameter *EventSpecific2* contains the handle of a call object. The call object can be used to retrieve information by calling DivaGetCallInfo or DivaGetCallProperties.

DivaEventMonitorRecordEnded

The event *DivaEventMonitorRecordEnded* is signaled when the recording stops.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the Dialogic® Diva® API when the monitor is created. The parameter *EventSpecific2* contains the reason for the stopping, typically user initiated.

DivaEventDeviceStatusChanged

The event *DivaEventDeviceStatusChanged* is reported when an enabled device status has changed.

The parameter *EventSpecific1* contains the line device ID of the device that has changed. The parameter *EventSpecifc2* contains the information about what has changed. This are one or more options of *DivaLineDeviceStatusEvents*.

DivaEventGenericToneEnded

The event *DivaEventGenericToneEnded* is reported when the generic tone generator enabled via *DivaGenerateSingleTone* or *DivaGenerateDualTone* has ended due to a timeout condition. The parameter *EventSpecific1* contains the call handle of the application.

DivaEventGenericToneDetected

The event *DivaEventGenericToneDetected* is reported when the generic tone detector enabled via *DivaDetectSingleTone* or *DivaDetectDualTone* has detected a tone that matches the detection parameter. The result must be retrieved via *DivaGetDetectToneResult*. The parameter *EventSpecific1* contains the call handle of the application.

DivaEventGenericToneInfo

The event *DivaEventGenericToneInfo* is reported when an answer to a low level generic tone request is available. The request has been previously issued by the application via the function *DivaSendGenericToneRequest*. The application must retrieve the information via *DivaGetGenericToneInfo*. The parameter *EventSpecific1* contains the call handle of the application.

DivaEventSms1MsgReceived

The event *DivaEventSms1MsgReceived* is signaled when a SMS message arrives. In this case, "message" refers to information origination in layer 2 or layer 3. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter is a pointer to the received message. The application must retrieve the information via *DivaSms1ReleaseMsgReceived*.

DivaEventSmsError

The event *DivaEventSmsError* is signaled by SMS when it detects an error. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter is a four-bytes integer.

DivaEventMWICompleted

The event *DivaEventMWICompleted* is reported when a message activation or deactivation request send via *DivaMWIActivate* or *DivaMWIDeactivate* has completed. The parameter *EventSpecific1* contains the application handle passed with the request. The parameter *EventSpecific2* contains the result. A value of zero indicates success.

DivaEventAnsweringMachineDetector

The event *DivaEventAnsweringMachineDetector* is signaled when the answering machine detector, started with DivaEnableAnsweringMachineDetector, has finished the analyses and reports the result. The event specific parameter 1 contains the handle of the call. The event specific parameter 2 contains the result. See DivaResultAnsweringMachineDetector for valid results.

DivaEventCallTimer

The event *DivaEventCallTimer* is signaled when a timer started with DivaStartCallTimer expires. The parameter *EventSpecific1* contains the call handle of the application.

DivaEventApplicationTimer

The event *DivaEventApplicationTimer* is signaled when a timer started with *DivaStartApplicationTimer* expires. There is no event specific parameter for this event.

DivaEventDTMFMaxDigits

The event *DivaEventDTMFMaxDigits* is signaled when DTMF processing for the group *DivaProcessingGroupEvent* is enabled and the amount of digits is reached. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* contains the amount of digits in the digit buffer.

DivaEventDTMFTerminationDigit

The event *DivaEventDTMFTerminationDigits* is signaled when DTMF processing for the group *DivaProcessingGroupEvent* is enabled and one of the enabled termination digits is received. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* contains the termination digit.

DivaEventDTMFInterDigitTimeout

The event *DivaEventDTMFInterDigitTimeout* is signaled when DTMF processing for the group *DivaProcessingGroupEvent* is enabled via DivaSetDTMFProcessingRules and the inter digit timeout is reached after receiving the last digit. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* contains the amount of digits in the digit buffer.

DivaEventDTMFInitialDigitTimeout

The event *DivaEventDTMFInitialDigitTimeout* is signaled when DTMF processing for the group *DivaProcessingGroupEvent* is enabled and no DTMF digit has been received within the initial digit timeout set via DivaSetDTMFProcessingRules. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* is always zero.

DivaEventCallHoldNotify

The event *DivaEventCallHoldNotify* is signaled when the remote end puts the call on hold. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventCallRetrievedNotify

The event *DivaEventCallRetrieveNotify* is signaled when the remote end puts retrieves a call that has been previously on hold. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventCallTransferredNotify

The *DivaEventCallTransferredNotify* event is signaled when the call is transferred by the remote party. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventFaxDocumentSent

The *DivaEventFaxDocumentSent* event is signaled when the multi fax document sending is ongoing and one document has been completed. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventFlashCompleted

The *DivaEventFlashCompleted* event is signaled when a hook flash initiated by *DivaSendFlash* has been completed. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventDTMFMaxTimeout

The event *DivaEventDTMFMaxDigitTimeout* is signaled when DTMF processing for the group *DivaProcessingGroupEvent* is enabled via DivaSetDTMFProcessingRules and no other part of the rules specified expired before the maximum timeout was detected. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* contains the amount of digits in the digit buffer.

DivaEventMonitorAudioData

The event *DivaEventMonitorAudioData* is signaled when audio data for the monitored call is available that can be retrieved by the application. The parameter *EventSpecific1* contains the monitor handle of the application passed to the Diva API when the monitor instance is created. The parameter *EventSpecific2* contains the handle of a call object. The call object can be used to retrieve the audio data using *DivaMonitorReceiveAudio*. It is recommended to call *DivaMonitorReceiveAudio* in a loop until the returned amount of bytes is zero.

CHAPTER 6

Dialogic® Diva® API Data Structures and Defines

This chapter contains the Dialogic® Diva® API data structures and defines.

DivaCallType

```
typedef enum
{
        DivaCallTypeVoice,
        DivaCallTypeFax,
        DivaCallTypeModem,
        DivaCallTypeDigitalData,
        DivaCallTypeX75,
        DivaCallTypeV120,
        DivaCallTypeGSM,
        DivaCallTypeVoIP,
        DivaCallTypeX25,
        DivaCallTypeSMS,
        DivaCallTypeAutoDetect
} DivaCallType;
```

DivaCallTypeVoice

The call is processed as a voice call. The data channel is set to plain audio streaming according to G.711. The Diva API handles a-law and µ-law coding for all voice streaming functions. Outgoing calls with this call type are signaled to the switch as speech.

DivaCallTypeFax

The call is processed as a fax G3 call. The data channel is set to support the fax G3 protocol including polling etc. Outgoing calls with this call type are signaled to the switch as 3.1 kHz audio.

DivaCallTypeModem

The call is processed as an analog modem call. The data channel is set to support a full analog modem including automatic speed negotiation. Outgoing calls with this call type are signaled to the switch as 3.1 kHz audio.

DivaCallTypeDigitalData

The call is processed as a digital data call. The data channel is set to handle digital data. Plain HDLC is done. This ensures that received data are valid but it does not guarantee packet delivery. Outgoing calls with this call type are signaled to the switch as unrestricted digital information.

DivaCallTypeX75

The call is processed as a reliable digital data call. The data channel is set to handle digital data using the X.75 protocol that guarantees packet delivery and flow control. Outgoing calls with this call type are signaled to the switch as unrestricted digital information.

DivaCallTypeV120

The call is processed as a reliable digital data call. The data channel is set to handle digital data using the V.120 protocol that guarantees packet delivery and flow control. Outgoing calls with this call type are signaled to the switch as unrestricted digital information.

DivaCallTypeGSM

The call is processed as a GSM data call. The data channel is set to handle the V.110 protocol. The default speed is 9600 bps, which is the typical speed for GSM connections. Outgoing calls with this call type are signaled as V.110, with specific information set in the ISDN protocol elements.

DivaCallTypeVoIP

The call is processed as a VoIP call. The data channel is set to handle the RTP protocol. Outgoing calls with this call type are signaled to the switch as speech.

DivaCallTypeX25

The call is processed as a reliable digital data call. The data channel is set to handle digital data using the X.25 protocol that guarantees packet delivery and flow control. Outgoing calls with this call type are signaled to the switch as unrestricted digital information.

DivaCallTypeSMS

The call is processed as SMS data call. The data channel is set to handle Short Messages using the SMS protocol 1.

DivaCallTypeAutoDetect

This call type is only valid for incoming digital calls. The Dialogic® Diva® SDK detects automatically if it is a digital data, X.75, X.25, or V.120 call type. The result is signaled in *DivaCallInfo*.

DivaListenType

```
typedef enum
{
        DivaListenNone = 0x00000000,
        DivaListenAll = 0xffffffff,
        DivaListenAllVoice = 0x00010032,
        DivaListenAllSMS = 0x00010012
        DivaListenSpeech = 0x00000002,
        DivaListenAudio3_1KHz = 0x00000010,
        DivaListenAudio7KHz = 0x00000020,
        DivaListenTelephony = 0x00010000,
        DivaListenFaxG3 = 0x00020000
} DivaListenType;
```

In the ISDN environment, calls may be signaled with different parameters depending on the switch capacities and the network. *DivaListenType* defines the different types for voice and fax G3.

Note: In certain environments, fax calls may be signaled as 3.1 kHz audio or even as speech. If the application cannot be sure which service is signaled, it should use *DivaListenAll*.

DivaLineDeviceInfo

```
typedef struct
{
     DWORD            Size                       // Size of the structure, may depend on the API version
     DWORD            Channels;                  // The number of data channels
     BOOL             bModemSupported;           // If true, analog modem is supported
     BOOL             bFaxSupported;             // If true, fax is supported
     BOOL             bVoIPSupport;              // If VoIP support is available
     DWORD            CapiControllerId;          // Controller ID assigned by CAPI
     DivaLineCodec    LineCodec                  // Audio codec on the line
     BOOL             bLISupported               // If true, line interconnect is supported
     DWORD            SerialNumber               // serial number of the hardware
     DWORD            HardwareLineIndex          // used for hardware that supports multiple lines. Index starting with
                                                 1
     BOOL             bExtVoiceSupported         // If true, extended voice capabilities supported
     BOOL             bHoldRetrievesupported;
     BOOL             bTransferSupported;
     BOOL             bForwardSupported;
     BOOL             bCallDeflectionSupported;
     BOOL             bManagementSupported;
} DivaLineDeviceInfo;
```

Members

Size

The *Size* defines the length of the structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the version of the Dialogic® Diva® API, the size may grow.

Channels

The *Channels* member defines the total number of parallel connections that can be established on the line device.

bModemSupported

If *bModemSupported* is set to TRUE, the line device supports analog modem connections.

bFaxSupported

If *bFaxSupported* is set to TRUE, the line device supports fax G3 connections.

bVoIPSupport

If *bVoIPSupport* is set to TRUE, the line device supports VoIP connections.

CapiControllerId

This parameter defines the CAPI controller number that is used by the line device.

LineCodec

This parameter indicates the audio format of the line. It depends on the ISDN protocol that is used on this line. Possible line codecs are given in DivaLineCodec.

bLISupported

If *bLISupported* is set to TRUE, the line device supports Line Interconnect. Line Interconnect is also the base for conferencing.

SerialNumber

This parameter defines the unique serial number of the hardware.

HardwareLineIndex

This parameter is used for hardware supporting multiple line devices. All line devices will have the same serial number and *HardwareLineIndex* identifies the line. The first line is 1, etc.

bExtVoiceSupported

If *bExtVoiceSupported* is set to TRUE, the line device supports extended voice capabilities like echo canceller and enhanced tone detection and generation.

bHoldRetrieveSupported

If *bHoldRetrieveSupported* is set to TRUE, the line device supports hold and retrieve. Please note that this does not guarantee that the switch supports it as well.

bTransferSupported

If *bTransferSupported* is set to TRUE, the line device supports call transfer. Please note that this does not guarantee that the switch supports it as well.

bForwardSupported

If *bForwardSupported* is set to TRUE, the line device supports call forwarding. Please note that this does not guarantee that the switch supports it as well.

bCallDeflectionSupported

If *bCallDeflectionSupported* is set to TRUE, the line device supports call deflection. Please note that this does not guarantee that the switch supports this as well.

bManagementSupported

If *bManagementSupported* is set to TRUE, the line device supports the management interface extensions. See DivaGetLineDeviceStatus, DivaGetLineDeviceConfiguration as well as DivaDeviceMgmtGetValue and DivaDeviceMgmtSetValue for more information.

DivaLineDeviceParamsFax

```
typedef struct
{
    DWORD               Size;
    // General parameters
    char                LocalNumber[MAX_ADDR_LEN];
    char                LocalSubAddress [MAX_SUBADDR_LEN];
    // Fax-related parameters
    char                LocalFaxId[MAX_ADDR_LEN];
    char                FaxHeadLine[MAX_ADDR_LEN];
    DWORD               DefaultMaxSpeed;
    DivaFaxOptions      FaxOptions
} DivaLineDeviceParamsFax;
```

Members

Size

The *Size* defines the length of the structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the Dialogic® Diva® API version, the size may grow.

LocalNumber

The *LocalNumber* is signaled to the remote side with outgoing calls. Please note that the underlying protocol must support transport of this information.

LocalSubAddress

The *LocalSubAddress* is signaled to the remote side with outgoing calls. Please note that the underlying protocol must support transport of this information.

LocalFaxId

The *LocalFaxId* is used during establishment of the fax connection. This parameter identifies the local fax station and is typically the phone number of the local fax machine.

FaxHeadLine

The *FaxHeadLine* is printed at the top of all sent fax pages. In addition to this line, the date and time as well as page information is given.

DefaultMaxSpeed

The *DefaultMaxSpeed* defines the maximum speed that can be negotiated for outbound and inbound fax G3 connections.

FaxOptions

The *FaxOptions* define how fax G3 connections are negotiated and which parameters are allowed. See *DivaFaxOptions* for possible options.

See also

DivaFaxOptions, DivaSetLineDeviceParamsFax

DivaLineDeviceParamsVoice

```
typedef struct
{
        DWORD                   Size;
        // General parameters
        char                    LocalNumber[MAX_ADDR_LEN];
        char                    LocalSubAddress[MAX_SUBADDR_LEN];
        // Voice-related parameter
        DWORD                   VoiceOptions
} DivaLineDeviceParamsVoice;
```

Members

Size

Size defines the length of the structure. The application sets this value before calling any function that gets this structure as a parameter. Depending on the version of the Dialogic® Diva® API, the size may grow.

LocalNumber

The *LocalNumber* is signaled to the remote side with outgoing calls. Please note that the underlying protocol must support transport of this information.

LocalSubaddress

The *LocalSubaddress* is signaled to the remote side with outgoing calls. Please note that the underlying protocol must support transport of this information.

VoiceOptions

The *VoiceOptions* define voice-specific parameters to be used for all voice calls. See DivaVoiceOptions for possible options.

See also

DivaVoiceOptions, DivaSetLineDeviceParamsVoice

DivaEventModes

```
typedef enum
{
            DivaEventModeCallback,
            DivaEventModeCallbackEx,
            DivaEventModeCallbackSignal
} DivaEventModes;
```

DivaEventModeCallback

The application receives the event reported by the Dialogic® Diva® API via a callback function. For more information on the callback function, refer to Callback function.

DivaEventModeCallbackEx

The application receives the event reported by the Diva API via a callback function. For more information on the callback function, refer to CallbackEx function.

DivaEventModeCallbackSignal

The application receives the notification that a new event is available at the Diva API via a callback function. For more information on the callback function, refer to CallbackSignal function.

DivaCallState

The *DivaCallState* constants define the current state of a call.

```
typedef enum
{
        DivaCallStateIdle = 0,
        DivaCallStateDialing = 1,
        DivaCallStateRinging,
        DivaCallStateOffering,
        DivaCallStateAnswered,
        DivaCallStateProceeding,
        DivaCallStateConnected,
        DivaCallStateOnHold,
        DivaCallStateDisconnecting,
        DivaCallStateDisconnected
} DivaCallState;
```

DivaCallStateIdle

This is the initial call state for a call handle created with *DivaCreateCall*.

DivaCallStateDialing

The call is initiated and the dialing information is being given to the switch. This call state is only available for outgoing calls.

DivaCallStateRinging

Dialing is finished and the confirmation has been received that ringing at the remote end has started. This call state is only available for outgoing calls.

DivaCallStateOffering

The call has been signaled to one or more applications and not yet been answered. This call state is only available for incoming calls.

DivaCallStateAnswered

The call has been answered by the application and call establishment is in progress. This call state is only available for incoming calls.

DivaCallStateProceeding

The call is connected at the signaling level and the data channel connection is initiated.

DivaCallStateConnected

The data channel is connected and data can be streamed in both directions.

DivaCallStateOnHold

The call is in hold state and no data channel is currently available.

DivaCallStateDisconnecting

Disconnect of the call is in progress.

DivaCallStateDisconnected

The call is disconnected. When it reaches this state, the application may read the disconnect reason via *DivaGetCallInfo* if required. It must then close the call by calling *DivaCloseCall*.

DivaCallInfo

```
typedef struct
{
        // Size of the structure, may depend on the API version
        DWORD                   Size
        DWORD                   LineDevice;
        DivaCallState           CallState;
        DivaListenType          Service;
        DivaCallType            CallType;
        BOOL                    bDataTransferPossible;
        char                    CallingNumber[MAX_ADDR_LEN];
        char                    CalledNumber[MAX_ADDR_LEN];
        DWORD                   RxSpeed;
        DWORD                   TxSpeed;
        DWORD                   Compression;
        // Parameters valid if call fails or after termination
        DivaDisconnectReasons   DisconnectReason;
        DWORD                   dwISDNCause;
        // Special parameters for CallType Fax
        char                    RemoteFaxId[MAX_ADDR_LEN];
        char                    LocalFaxId[MAX_ADDR_LEN];
        DWORD                   dwFaxPages;
        BOOL                    bPollingActive;
        DivaFaxResolution       PageResolution;
        BOOL                    bMRActive;
        BOOL                    bMMRActive;
        BOOL                    bECMActive;
        // Special parameters for CallType Voice
        BOOL                    bEchoCancellerActive;

        DWORD                   dwRedirectReason;
        char                    RedirectedNumber[MAX_ADDR_LEN];
        char                    RedirectingNumber[MAX_ADDR_LEN];

        DivaSignalledCallType   SignalledCallType;
        DWORD                   AssignedBChannel
} DivaCallInfo;
```

Members

Size

Size defines the length of the data structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the version of the Dialogic® Diva® API, the size may grow.

CallState

CallState defines the current state of a call. Valid call states are listed in *DivaCallState*. The event DivaEventCallInfo is signaled if the call state changes.

Service

The *Service* parameter defines the service of a call. It is automatically selected and corresponds to the call type for outgoing calls. For incoming calls, the service is read from the incoming call parameters, if possible. As long as an incoming call has not been accepted, the service may change online events, e.g., tone detection.

CallType

The *CallType* is selected by the application when connecting, either incoming or outgoing. It can also be changed during a call.

bDataTransferPossible

If this parameter is TRUE, data can be sent and received. Typically, data are transferred in connected state. In case of voice applications; however, data transfer may be possible earlier.

CallingNumber

The *CallingNumber* is a zero-terminated string containing the number of the caller.

CalledNumber

The *CalledNumber* is a zero-terminated string containing the number of the called party.

RxSpeed

RxSpeed specifies the connection speed in receive direction. This speed differs from the *TxSpeed* only in case of analog connections.

TxSpeed

TxSpeed specifies the connection speed in sending direction. This speed differs from the *RxSpeed* only in case of analog connections.

Compression

This parameter specifies if the used call type uses compression.

dwDisconnectReason

This parameter contains the converted disconnect reason. The parameter is valid when the call enters the state *DivaCallStateDisconnected*. The values for this parameter are defined by *DivaDisconnectReasons*.

dwISDNCause

This parameter contains the plain cause code for the disconnect reason, received from the underlying network. The parameter is valid when the call enters the state *DivaCallStateDisconnected*. The values for this parameter are defined by the ISDN specifications. Users of this parameter should be familiar with ISDN cause codes.

RemoteFaxId

This parameter is only valid if the call type is set to *DivaCallTypeFax*. It is a zero-terminated string containing the station identification of the remote fax. This identification does not rely on the underlying communication network and is not identical to the calling or called number.

dwLocalFaxId

This parameter is only valid if the call type is set to *DivaCallTypeFax*. The value is set by the application.

dwFaxPages

This parameter is only valid if the events *DivaEventFaxSent* or *DivaEventFaxReceived* have been signaled or the call type has been set to *DivaCallTypeFax*. It contains the total number of pages sent or received.

bPollingActive

This parameter is only valid if the call type is set to *DivaCallTypeFax*. It is set to true if the transfer direction of the fax has been changed or fax polling or fax on demand has been selected.

PageResolution

This parameter is only valid if the call type is set to *DivaCallTypeFax*. It defines the resolution of the fax and contains one of the values defined in *DivaFaxResolution*.

bMRActive, bMMRActive

These parameters are only valid if the call type is set to *DivaCallTypeFax*. They define the used extended fax coding method. If neither parameter is set, the default coding according to T.30 is used. Please note that these coding methods are only used on the line and do not correspond to any data format of the received or sent file.

bECMActive

This parameter is only valid if the call type is set to *DivaCallTypeFax*. The parameter indicates if the reliable Error Correction Mode has been negotiated or not.

bEchoCancellerActive

This parameter is only valid if the call type is set to *DivaCallTypeVoice*. If the parameter is set, the echo canceller is active.

dwRedirectReason

This parameter indicates if the call has been redirected from another party. Possible values are defined in *DivaRedirectReason*.

RedirectedNumber

This parameter contains a zero-terminated string with the redirected number.

RedirectingNumber

This parameter contains a zero-terminated string with the number of the redirecting party.

SignalledCallType

This parameter is only valid for incoming calls. The call type of an incoming call is evaluated, based on the information delivered from the network. For valid options, see DivaSignalledCallType.

AssignedBchannel

The call is available once it is proceeding in the assigned B-channel. This channel is the physical channel assigned by the network. Numbering starts with one. For Dialogic® Diva® PRI Media Boards, the numbering may not be continuous.

DivaDisconnectReasons

```
typedef enum
{
        DivaDRNormalCallClearing = 0,
        DivaDRActiveDisconnect,
        DivaDRBusy,
        DivaDRReject,
        DivaDRNoAnswer,
        DivaDRAlertingNoAnswer,
        DivaDRNumberUnknown,
        DivaDRInvalidNumber,
        DivaDRNumberChanged,
        DivaDRNoChannelAvailable,
        DivaDRCableError,
        DivaDRGeneralNetworkError,
        DivaDRUnspecifiedError,
        DivaDRUnallocatedNumber,
        DivaDRAnotherAppGotThatCall,
        DivaDRDataChannelFailed,
        DivaDRNoFaxDevice,
        DivaDRFaxTrainingFailed,
        DivaDRFaxRemoteAbort,
        DivaDRFaxLocalAbort,
        DivaDRModemNegotiationFailed,
        DivaDRModemNoAnswer,
        DivaDRModemCarrierLost,
        DivaDRIncompatibleDestination,
        DivaDRFileAccess,
        DivaDRLowMemory,
        DivaDRIllegalData,
        DivaDRConnectTimeout
} DivaDisconnectReasons
```

DivaDRNormalCallClearing

The call ended with the default cause.

DivaDRActiveDisconnect

The application initiated the disconnect of the call.

DivaDRBusy

The remote end is busy and could not take the call. This disconnect reason is only signaled for outgoing calls.

DivaDRReject

The call was rejected by the remote peer.

DivaDRNoAnswer

The remote end did not answer the call and the call timed out. This disconnect reason is only signaled for outgoing calls.

DivaDRAlertingNoAnswer

The remote end did not answer the call, even though it sent an alert to keep the call ringing. This disconnect reason is only signaled for outgoing calls.

DivaDRNumberUnknown

The switch responds that the dialed number is not known.

DivaDRInvalidNumber

The switch responds that the dialed number is not in a valid format or not complete.

DivaDRNumberChanged

The switch responds that the dialed number is not known because it has changed.

DivaDRNoChannelAvailable

All channels on the line device are already in use.

DivaDRCableError

There is no layer 1 connection between the line device and the switch. This is typically a cable error or an un-plugged line device.

DivaDRGeneralNetworkError

A general network error occurred during call establishment.

DivaDRUnspecifiedError

There is no specific information regarding why the call failed.

DivaDRUnallocatedNumber

The dialed number is no longer available.

DivaDRAnotherAppGotThatCall

Another application answered the call.

DivaDRDataChannelFailed

The negotiation of the data-related protocol failed, e.g., modem negotiation.

DivaDRNoFaxDevice

The remote side is not a fax device.

DivaDRFaxTrainingFailed

The line quality is too bad to establish a fax connection.

DivaDRFaxRemoteAbort

The remote side has aborted the fax protocol.

DivaDRFaxLocalAbort

The fax protocol has been terminated from the local side. Reasons may be too many retries, for example.

DivaDRModemNegotiationFailed

The modem negotiation failed. This may be a line quality problem, or the settings of both sides not matching.

DivaDRModemNoAnswer

The remote modem did not answer.

DivaDRModemCarrierLost

The modem connection lost the carrier signal.

DivaDRIncompatibleDestination

The network could not reach the remote side due to compatibility issues.

DivaDRFileAccess

The file that should be accessed to send or receive data could not be accessed. See Dialogic® Diva® SDK trace for more information.

DivaDRLowMemory

The system is running out of memory.

DivaDRIllegalData

A file that is expected to contain a specific data format, e.g., fax TIFF format, did not contain valid data.

DivaDRConnectTimeout

A connect timeout specified by the application is reached. The application controls the timeout by the properties *DivaCPT_NoAnswerTimeout* or *DivaCPT_ConnectTimeout*.

DivaRedirectReason

```
typedef enum
{
        DivaRedirectReasonUnknown,
        DivaRedirectReasonBusy,
        DivaRedirectReasonNoReply,
        DivaRedirectReasonCallDeflection,
        DivaRedirectReasonDTEOutOfOrder,
        DivaRedirectReasonByCalledDTE,
        DivaRedirectReasonUnconditional
} DivaRedirectReason;
```

DivaSignalledCallType

```
typedef enum
{
        DivaSignalledCallTypeUnknown = 0,
        DivaSignalledCallTypeAnalog,
        DivaSignalledCallTypeDigital,
} DivaSignalledCallType;
```

DivaSignalledCallTypeUnknown

The call type for the incoming call could not be determined.

DivaSignalledCallTypeAnalog

The call is signaled as an analog call. It might be a voice, fax, or modem call.

DivaSignalledCallTypeDigital

The call is signaled as a digital call.

DivaReturnCodes

Name	Value	Description
DivaSuccess	0	Success
DivaErrorLineDevice	1	The specified line device is not available.
DivaErrorInvalidFunction	2	The requested function could not be performed.
DivaErrorInvalidHandle	3	The handle passed to the Dialogic® Diva® API is not valid. Either the device does not support the function or the function could not be performed due to the selected parameter, e.g., call type.
DivaErrorInvalidParameter	4	One or more of the parameters passed to the function are not valid.
DivaErrorInvalidState	5	The requested function could not be performed in the current state.
DivaErrorOutOfMemory	6	The allocation of memory failed.
DivaErrorNoCapi	7	This result code is obsolete. Refer to the result code DivaErrorNoDevice
DivaErrorCapiError	8	The underlying Dialogic® communication platform is not accessible.
DivaErrorDestBusy	9	The remote peer is busy. This result code is only valid for a blind call transfer completion event.
DivaErrorNoAnswer	10	The remote peer did not answer. This result code is only valid for a blind call transfer completion event.
DivaErrorNoChannel	11	There is no data channel to initiate the call. This could happen if all data channels are in use or if a fixed channel is selected that is already in use.
DivaErrorOpenFile	12	The specified file could not be opened. Files are opened in read only mode.
DivaErrorUnsupportedFormat	13	The specified format is not supported. The format may be specified as a parameter or read from the file.
DivaErrorReadFile	14	The Diva API failed to read the expected data from the specified file.
DivaErrorAnotherAppGotThatCall	15	An incoming call was answered by another application.
DivaErrorTimeout	16	Future use
DivaErrorUnallocatedNumber	17	The dialed number is not known by the carrier. This result code is only valid for a blind call transfer completion event.
DivaErrorNotSupported	18	The requested function is not supported by the selected line device.
DivaErrorUnspecific	19	The outgoing call failed with an unspecified error. This result code is only valid for a blind call transfer completion event.
DivaErrorReadOnlyParameter	20	Future use
DivaErrorDataSize	21	The provided buffer space to return system information is not big enough.
DivaErrorWriteOnlyParameter	22	Future use
DivaErrorAlreadyAssigned	23	An audio provider with the same name is already assigned.
DivaErrorNoDataAvailable	24	There is no data available for a generic tone operation.
DivaErrorEndOfData	25	The end of the data has been reached for a received fax in TIFF format.
DivaErrorFormatNotEnabled	26	The fax document format is supported but not enabled by the application.

DivaFaxFormat

```
typedef enum
{
        DivaFaxFormatAutodetect,
        DivaFaxFormatDefault,
        DivaFaxFormatTIFF_ClassF = 1,
        DivaFaxFormatTIFF_ClassFSymmetric,
        DivaFaxFormatSFF,
        DivaFaxFormatTIFF_G3,
        DivaFaxFormatTIFF_G3Symmetric,
        DivaFaxFormatTIFF_G4,
        DivaFaxFormatTIFF_G4Symmetric,
        DivaFaxFormatColorJPEG,
        DivaFaxFormatASCII,
} DivaFaxFormat;
```

DivaFaxFormatAutodetect

The format of the fax document is detected by the file extension and the header information. This is only valid for function calls to transmit a fax.

DivaFaxFormatDefault

The default format is used. This option is only valid as parameter for *DivaReceiveFax*. The default format is TIFF class F.

DivaFaxFormatTIFF_ClassF

The data is coded according to the TIFF Class F specification.

DivaFaxFormatTIFF_ClassFSymmetric

The data is coded according to the TIFF Class F specification. Resolution of the pages is aligned to be symmetric.

DivaFaxFormatSFF

The data is coded according to the SFF format that is used as the internal format of the Dialogic® Diva® API and also the CAPI interface.

DivaFaxFormatTIFF_G3

The data is coded according to the TIFF Class F specification using the G3 coding.

DivaFaxFormatTIFF_G3Symmetric

The data is coded according to the TIFF Class F specification using the G3 coding. Resolution of the pages is aligned to be symmetric.

DivaFaxFormatTIFF_G4

The data is coded according to the TIFF Class F specification using the G4 coding. This format has a higher compression and requires less disk space.

DivaFaxFormatTIFF_G4Symmetric

The data is coded according to the TIFF Class F specification using the G4 coding format has a higher compression and requires less disk space. Resolution of the pages is aligned to be symmetric.

DivaFaxFormatColorJPEG

The data is coded as JPEG according to the color fax specification.

DivaFaxFormatASCII

The data provided should be interpreted as plain unformatted text. Note that this requires setting the call property DivaCPT_FaxUseTextForSending to true before initiating the connection of the switch to fax mode.

DivaExtensions

```
typedef enum
{
        DivaExtensionFaxFormat,
} DivaExtensions;
```

DivaExtensionFaxFormat

The extension *DivaExtensionFaxFormat* enables or disables the fax formats superfine and ultrafine. If enabled, the capabilities are signaled to the calling side on inbound faxes and negotiated on outbound faxes depending on the document format. By default, the Dialogic® Diva® API negotiates only formats up to fine.

DivaLineCodec

```
typedef enum
{
        // Audio codec on the line
        LineAudio_ALaw = 0,          // G.711 ALaw
        LineAudio_uLaw = 1,          // G.711 uLaw
} DivaLineCodec;
```

LineAudio_ALaw

The A-Law audio codec is generally used in Europe.

LineAudio_uLaw

The µ-Law audio codec is generally used in North America.

DivaAudioFormat

The Dialogic® Diva® SDK supports several audio formats. The formats contain the codec and the storage format. The storage format can be the well known wave format and the raw format.

The raw formats do not contain any header. The data is coded in the given format (codec) without any preceding information.

The format containing the wave header can be used for file-based streaming, but not for memory-based streaming.

```
typedef enum
{
            /*
             * Windows® waveform (WAV) formats:
             */
            DivaAudioFormat_aLaw8K8BitMono =  0,
            DivaAudioFormat_uLaw8K8BitMono =  1,
            DivaAudioFormat_PCM_8K8BitMono =  2,
            DivaAudioFormat_PCM_8K16BitMono =  3,
            DivaAudioFormat_GSM_610 = 10,
            /*
             * Raw audio formats:
             */
            DivaAudioFormat_Raw_aLaw8K8BitMono = 100,
            DivaAudioFormat_Raw_uLaw8K8BitMono = 101,
            DivaAudioFormat_Raw_PCM_8K8BitMono = 102,
            DivaAudioFormat_Raw_PCM_8K16BitMono = 103,
            DivaAudioFormat_Raw_ADPCM_8K4BitMono = 104,
            DivaAudioFormat_Raw_ADPCM_6K4BitMono = 105,
} DivaAudioFormat;
```

DivaAudioFormat_aLaw8K8BitMono

The data is coded as 8 Bit a-law with an 8 KHz sampling rate. The storage format contains a wave file header.

DivaAudioFormat_uLaw8K8BitMono

The data is coded as 8 Bit µ-law with an 8 KHz sampling rate. The storage format contains a wave file header.

DivaAudioFormat_PCM_8K8BitMono

The data is coded as 8 Bit PCM with an 8 KHz sampling rate. The storage format contains a wave file header. Please note that the 8 Bit PCM format may contain a higher noise than a-law or µ-law formats.

DivaAudioFormat_PCM_8K16BitMono

The data is coded as 16 Bit PCM with an 8 KHz sampling rate. The storage format contains a wave file header.

DivaAudioFormat_GSM_610

The format *DivaAudioFormat_GSM_610* specifies that the file format is a wave file and the coding is according to GSM 610. Please note that this format is only available for the function *DivaMonitorRecordAudio*. All other functions will return an error if this format is used.

DivaAudioFormat_Raw_aLaw8K8BitMono

The data is coded as 8 Bit a-law with an 8 KHz sampling rate. The storage format is raw and contains no header.

DivaAudioFormat_Raw_uLaw8K8BitMono

The data is coded as 8 Bit µ-law with an 8 KHz sampling rate. The storage format is raw and contains no header.

DivaAudioFormat_Raw_PCM_8K8BitMono

The data is coded as 8 Bit PCM with an 8 KHz sampling rate. The storage format is raw and contains no header. Please note that the 8 Bit PCM format may contain a higher noise than a-law or µ-law formats.

DivaAudioFormat_Raw_PCM_8K16BitMono

The data is coded as 16 Bit PCM with an 8 KHz sampling rate. The storage format is raw and contains no header.

DivaAudioFormat_Raw_ADPCM_8K4BitMono

The data is coded as 4 Bit Adaptive PCM. The sampling rate is 8 KHz. The storage format is raw and contains no header. This format is an adaptive format and can only be processed-based on an audio file.

DivaAudioFormat_Raw_ADPCM_6K4BitMono

The data is coded as 4 Bit Adaptive PCM. The sampling rate is 6 KHz. The storage format is raw and contains no header. The sampling rate of 6 KHz requires an underlying Dialogic® communication platform that supports "extended voice". This format is an adaptive format and can only be processed-based on an audio file.

DivaFaxOptions

```
typedef enum
{
        DivaFaxOptionsDefault = 0x00000000,
        DivaFaxOptionDisableHighResolution = 0x00000001,
        DivaFaxOptionDisableMR = 0x00000002,
        DivaFaxOptionDisableMMR = 0x00000004,
        DivaFaxOptionDisableECM = 0x00000008,
        DivaFaxOptionEnablePolling = 0x00000100,
        DivaFaxOptionRequestPolling = 0x00000200,
        DivaFaxOptionReverseSession = 0x00000400,
        DivaFaxOptionMultipleDocument = 0x00000800,
        DivaFaxOptionEnableColor = 0x00001000,
        DivaFaxOptionEnableInterrupt = 0x00010000,
        DivaFaxOptionRequestInterrupt = 0x00020000,
} DivaFaxOptions;
```

DivaFaxOptionDefault

By default, the fax G3 protocol negotiates the best possible options. Certain options can be disabled by the settings described below. The polling mode is disabled by default.

DivaFaxOptionDisableHighResolution

This option reduces the vertical resolution to 100 DPI in order to reduce transmission time.

DivaFaxOptionDisableMR

This option disables additional compression. The data will be exchanged using the modified Huffman compression.

DivaFaxOptionDisableMMR

This option disables additional compression. The data will be exchanged using the modified Huffman compression or MR compression.

DivaFaxOptionDisableECM

This option disables the error correction mode.

DivaFaxOptionEnablePolling

This option must be set to be able to enter polling mode. The polling mode changes the direction of the fax transmission. If this option is enabled for an incoming call, a polling request from the calling side is accepted.

DivaFaxOptionRequestPolling

This option must be set to enter polling mode. The polling mode changes the direction of the fax transmission. If this option is set for an outgoing call, a polling request is sent to the called party.

DivaFaxOptionReverseSession

This option is used to change the call type to fax and to reverse the call direction at the same time. For example, an incoming voice call can be changed to fax and the fax is sent to the caller.

DivaFaxOptionMultipleDocument

This option must be set when connecting or answering a call if multiple documents should be sent on the connection.

DivaFaxOptionEnableColor

If *DivaFaxOptionEnableColor* is set, the negotiation of color fax in the fax protocol is enabled.

DivaFaxOptionEnableInterrupt

If this option is set, the connection may continue as voice connection after the fax has been processed. The remote side is responsible for requesting the interrupt procedure.

DivaFaxOptionRequestInterrupt

If this option is set, a request to change to voice mode after the fax has been sent is send to the remote party. The result is reported in the call property *DivaCPT_FaxProcedureInterrupt*.

Dialog®**DivaFaxResolution**

```
typedef enum
{
        DivaFaxResolutionStandard = 1,
        DivaFaxResolutionFine,
        DivaFaxResolutionSuperFine,
        DivaFaxResolutionUltraFine
} DivaFaxResolution;
```

DivaVoiceOptions

```
typedef enum
{
        DivaVoiceOptionDefault = 0x000,
        DivaVoiceOptionEarlyDataChannel = 0x001,
        DivaVoiceOptionEchoCanceller = 0x002
} DivaVoiceOptions;
```

DivaVoiceOptionEarlyDataChannel

If the option *DivaVoiceOptionEarlyDataChannel* is selected, the data channel is established before the B-channel connection is confirmed by the remote side. This is useful to hear announcements, ring tones, ring back tones, and busy tones. Please note that this feature is only available for outgoing calls.

DivaVoiceOptionEchoCanceller

If the *DivaVoiceOptionEchoCanceller* option is selected, the echo canceller is enabled for this connection.

DivaVoIPParams

```
typedef struct
{
        DWORD                   Size;
        DivaPayloadProtocol     PayloadProtocol;
        DivaPayloadOptions      PayloadOptions;
        DWORD                   MaxPacketLateRate;
        DWORD                   MaxDejitterDelay;
        DWORD                   SSRC;
        DWORD                   VoiceOptions
} DivaVoIPParams;
```

Size

Size defines the length of the structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the version of the Dialogic® Diva® API, the size may grow.

PayloadProtocol

This parameter defines the coding of the data used in send and receive direction of the data channel. It must be one of the values defined in *DivaPayloadProtocol*.

PayloadOptions

The *PayloadOptions* parameter defines the options to be used in the data channel. For available options and a detailed description refer to *DivaPayloadOptions*.

MaxPacketLateRate

The *MaxPacketLateRate* parameter defines the maximum period of time, in milliseconds, that a packet may be delayed before it is discarded.

MaxDejitterDelay

The *MaxDejitterDelay* parameter defines the maximum size of the anti-jitter buffer in milliseconds.

SSRC

The *SSRC* parameter is part of the RTP-packet header. It identifies the synchronization source. It is chosen randomly with the intent that only one synchronization source within one RTP session has the same SSRC identifiers.

VoiceOptions

The *VoiceOptions* parameter defines voice-specific options like enabling or disabling the echo canceller. For valid options, see DivaVoiceOptions.

DivaPayloadProtocol

```
typedef enum
{
        DivaPayload_PCMU,
        DivaPayload_PCMA,
        DivaPayload_G723
        DivaPayload_G726,
        DivaPayload_GSM,
} DivaPayloadProtocol;
```

DivaPayload_PCMU

The data format complies to G.711 PCM with µ-law coding.

DivaPayload_PCMA

The data format complies to G.711 PCM with a-law coding.

DivaPayload_G723

The data format complies to G.723.1 coding.

DivaPayload_G726,

The data format complies to G.726 32 kbps coding.

DivaPayload_GSM,

The data format complies to GSM 06.10 / ETS 300 961 full rate coding.

DivaPayloadOptions

```
typedef enum
{
        DivaDisableVoiceActivityDetection,
        DivaDisableComfortNoise,
        DivaDisableDTMFDetection,
        DivaDisableDTMFGeneration,
        DivaG723LowCodingRate
} DivaPayloadOptions;
```

DivaDisableVoiceActivityDetection

This parameter disables voice activity detection. By default, voice activity detection is enabled.

DivaDisableComfortNoise

This parameter disables the generation of comfort noise when no data is transmitted. By default, comfort noise generation is enabled.

DivaDisableDTMFDetection

This parameter disables the detection of in-band DTMF tones.

DivaDisableDTMFGeneration

This parameter disables the generation of in-band DTMF tones.

DivaG723LowCodingRate

If this option is selected, the low coding rate of the G.723 codec, i.e. 5.3 kbps, is used. The default coding rate is 6.3 kbps.

DivaModemOptions

```
typedef enum
{
        DivaModemOptionsDefault = 0,
        DivaModemOption7BitsPerChar = 0x1,
        DivaModemOption2StopBits = 0x2,
        DivaModemOptionParityOdd = 0x4,
        DivaModemOptionParityEven = 0x8,
        DivaModemOptionDisableRetrain = 0x10,
        DivaModemOptionDisableCompression = 0x20,
        DivaModemOptionDisableV42 = 0x40,
        DivaModemOptionDisableMNP = 0x80,
        DivaModemOptionForceReliable = 0x100,
        DivaModemOptionDisableAnswerTone = 0x00001000,
        DivaModemOptionEnableAutoMode = 0x00002000,
        DivaModemOptionSDCLEnable = 0x00010000,

        /*
        *Modem options for synchronous modems. The options following *DivaModemOptionSynchronous are
        only valid if this bit is set.
        */
        DivaModemOptionSynchronous = 0x80000000,
        DivaModemOptionSyncV22Normal = 0x80100000,
        DivaModemOptionSyncEnableV22bis = 0x80200000,
        DivaModemOptionSyncEnableV22FastSetup = 0x80400000,
        DivaModemOptionSyncEnableV22BisFastSetup = 0x82000000,
        DivaModemOptionSyncEnableV29FastSetup = 0x80800000,
        DivaModemOptionSyncSDLCExtModuloMode = 0x81000000
} DivaModemOptions;
```

DivaModemOptionsDefault

The default modem options will be used. The default options are 8 data bits, 1 stop bit, no parity, a full negotiation for reliable protocols like MNP or V.42, and compression.

DivaModemOption7BitsPerChar

If this option is selected, the framing for asynchronous modem connections is set to 7 data bits. The number of stop bits depends on the setting of *DivaModemOption2StopBits*.

DivaModemOption2StopBits

If this option is selected, the number of stop bits for asynchronous modem connections is set to 2. The number of data bits depends on the setting of *DivaModemOption7BitsPerChar*.

DivaModemOptionParityOdd

If this option is selected, the framing for asynchronous modem connections is set to use odd parity bits. By default, no parity bits are inserted.

This option cannot be combined with *DivaModemOptionParityEven*.

DivaModemOptionParityEven

If this option is selected, the framing for asynchronous modem connections is set to use even parity bits. By default, no parity bits are inserted.

This option cannot be combined with *DivaModemOptionParityOdd*.

DivaModemOptionDisableRetrain

If this option is selected, retrain for established connections is disabled.

DivaModemOptionDisableCompression

If this option is selected, no compression is negotiated for analog modem connections.

DivaModemOptionDisableV42

If this option is selected, the modem negotiation refuses V.42. The acceptance of MNP depends on the option *DivaModemOptionDisableMNP*.

DivaModemOptionDisableMNP

If this option is selected, the modem negotiation refuses MNP. The acceptance of V.42 depends on the option *DivaModemOptionDisableV42*.

DivaModemOptionForceReliable

If this option is selected, the connection will only be established if a reliable modem connection can be negotiated. If the remote side has disabled V.42 and MNP, the connection will fail.

DivaModemOptionDisableAnswerTone

If this option is selected, the modem will not generate any answer tone. This is used for synchronous modem connections.

DivaModemOptionEnableAutoMode

If this option is set, the modem will negotiate the connection speed and mode automatically. This is currently only available for synchronous modes.

DivaModemOptionSDLCEnable

If this option is set, the connection will use SDLC on top of the synchronous modem connection.

DivaModemOptionSynchronous

If this option is set, a synchronous modem connection is negotiated. The modulation depends on the speed and the other modulation settings for V.22 up to V.22 Dialogic® Diva® Fast Setup.

DivaModemOptionSyncV22Normal

In general, V.22 modem connections are negotiated. If this option is deselected and other synchronous modulations are selected, V.22 is disabled and a higher speed is negotiated.

DivaModemOptionSyncEnableV22bis

If this option is set, V.22bis is added to the enabled modulation modes. If no higher option is selected, this is the preferred modulation.

DivaModemOptionSyncV22FastSetup

If this option is set, V.22 Dialogic® Diva® Fast Setup is added to the enabled modulation modes. If no higher option is selected, this is the preferred modulation.

DivaModemOptionSyncEnableV22BisFastSetup

If this option is set, V.22bis Dialogic® Diva® Fast Setup is added to the enabled modulation modes. If no higher option is selected, this is the preferred modulation.

DivaModemOptionSyncEnableV29FastSetup

If this option is set, V.29 Dialogic® Diva® Fast Setup is added to the enabled modulation modes. If no higher option is selected, this is the preferred modulation.

DivaModemOptionSyncSDLCExtModuloMode

If this option is set, the modulo mode will be set to 128. If this option is not selected, the default modulo mode 8 is active.

DivaFaxMaxSpeed

```
typedef enum
{
        DivaFaxMaxAutomatic,
        DivaFaxMaxSpeed2400,
        DivaFaxMaxSpeed4800,
        DivaFaxMaxSpeed7200,
        DivaFaxMaxSpeed9600,
        DivaFaxMaxSpeed14400,
        DivaFaxMaxSpeed33600
}DivaFaxMaxSpeed;
```

DivaTransferOptions

```
typedef enum
{
        DivaTransferOptionDefault = 0x0000,
        DivaTransferOptionNoHoldRequired,
        DivaTransferOptionOnAlerting,
        DivaTransferOptionOnProceeding,
        DivaTransferOptionCallDeflection,
        DivaTransferOptionUseCallingNumber,
        DivaTransferOptionUseCallingName
}DivaTransferOptions;
```

DivaTransferOptionDefault

DivaTransferOptionDefault uses the transfer capabilities of the switch or PBX to which the corresponding line device is connected. If the default transfer options are selected, the primary call is placed on hold before the transfer is initiated. On blind transfer no specific channel handling is done for the consultation call. The transfer is completed when the consultation call is connected.

DivaTransferOptionNoHoldRequired

If this option is set, the Dialogic® Diva® SDK did not transfer the primary call to the hold state before initiating the transfer.

DivaTransferOptionOnAlerting

If this option is set, the Diva SDK completes a blind call transfer if the secondary call reaches the alerting state. By default, the transfer is completed in the connected state. This option can only be used for *DivaBlindCallTransfer*.

DivaTransferOptionOnProceeding

If this option is set, the Diva SDK completes a blind call transfer if the secondary call reaches the proceeding state. By default, the transfer is completed in the connected state. This option can only be used for *DivaBlindCallTransfer*.

DivaTransferOptionCallDeflection

If this option is set, the Diva SDK handles a call transfer as call deflection. This is only possible if the primary call is in the offering state. This option can only be used for *DivaBlindCallTransfer*.

DivaTransferOptionUseCallingNumber

This option is only valid in combination with *DivaTransferOptionCallDeflection*. If this option is set, the Diva SDK signals the calling party number set by the call property *DivaCPT_CallingNumber* when deflecting the call.

DivaTransferOptionUseCallingName

This option is only valid in combination with *DivaTransferOptionCallDeflection*. If this option is set, the Diva SDK signals the calling party number set by the call property *DivaCPT_CallingName* when deflecting the call.

DivaContinuousTones

```
typedef enum
{
        DivaEndOfTone = 0x80,
        DivaUnknownTone,
        DivaDialTone,
        DivaPBXInternalDialTone,
        DivaSpecialDialTone,
        DivaSecondDialTone,
        DivaRingingTone,
        DivaSpecialRingingTone,
        DivaBusyTone,
        DivaCongestionTone,
        DivaSpecialInformationTone,
        DivaComfortNoise,
        DivaHoldTone,
        DivaRecordTone,
        DivaCallerWaitingTone,
        DivaCallWaitingTone,
        DivaPayTone,
        DivaPositiveIndicationTone,
        DivaNegativeIndicationTone,
        DivaWarningTone,
        DivaIntrusionTone,
        DivaCallingCardServiceTone,
        DivaPayphoneRecognitionTone,
        DivaCPEAlertingSignal,
        DivaOffHookWarningTone,
        DivaInterceptTone,
        DivaModemCallingTone,
        DivaFaxCallingTone,
        // These tones can be detected but not generated.
        DivaAnswerTone,
        DivaAnswerTonePhaseReversal,
        DivaANSam,
        DivaANSamPhaseReversal,
        DivaBell103AnswerTone,
        DivaFaxFlags,
        DivaFaxG2GroupId,
        DivaHumanSpeech,
        DivaAnsweringMachineTone
} DivaContinuousTones;
```

DivaMultiFrequencyTones

```
typedef enum
{
        DivaToneMF1 = 0xF1,
        DivaToneMF2,
        DivaToneMF3,
        DivaToneMF4,
        DivaToneMF5,
        DivaToneMF6,
        DivaToneMF7,
        DivaToneMF8,
        DivaToneMF9,
        DivaToneMF0,
        DivaToneMFStart = 0xFD,
        DivaToneMFStop = 0xFF
} DivaMultiFrequencyTones;
```

DivaVoiceDataSource

Audio data may be stored in a file or at a specific memory location if it is to be streamed by functions that support enhanced voice streaming.

```
typedef enum
{
        DivaVoiceDataSourceFile = 1,
        DivaVoiceDataSourceMemory
} DivaVoiceDataSource;
```

DivaVoiceDataSourceFile

The audio data is file-based. This parameter does not specify the coding format of the data stored in the file.

DivaVoiceDataSourceMemory

The audio data is located in the memory. This parameter does not specify the coding format of the memory-based audio data.

DivaVoicePositionFormat

For enhanced audio streaming, the offset from the start and the duration might be specified. The *DivaVoicePositionFormat* defines how the offset and duration are calculated.

```
typedef enum
{
        DivaVoicePositionFormatBytes = 1,
        DivaVoicePositionFormatMSec
} DivaVoicePositionFormat;
```

DivaVoicePositionFormatBytes

The offset and duration are given in bytes of the audio stream.

DivaVoicePositionFormatMSec

The offset and duration are given in milliseconds.

DivaVoiceDescriptor

```
typedef struct
{
        DWORD                       Size;
        DivaVoiceDataSource         DataSource;
        DivaAudioFormat             DataFormat;
        DivaVoicePositionFormat     PositionFormat;
        DWORD                       StartOffset;
        DWORD                       Duration;
        union
        {
        struct
        {
        char                        *pFilename;
        }File;
        struct
        {
        DWORD                       DataLength;
        char                        *pBuffer;
        }Memory;
        }Source;
}DivaVoiceDescriptor;
```

Size

Size defines the length of the structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the Dialogic® Diva® API version the size may grow.

DataSource

DataSource defines if the data is memory or file-based. For defined values, refer to DivaVoiceDataSource.

DataFormat

DataFormat defines the codec used for the data. For defined values, refer to DivaAudioFormat. If streaming from memory is selected, only the so called "raw" formats are valid.

PositionFormat

PositionFormat defines how *StartOffset* and *Duration* should be interpreted. For defined values, refer to DivaVoicePositionFormat.

StartOffset

StartOffset defines where to start streaming of the audio data. In general, the streaming is started at the beginning and *StartOffset* is set to zero.

Duration

Duration defines the maximum amount of audio data to be streamed. By default, it should be set to zero and the Diva API streams the complete data either defined by the file size or the memory size.

pFilename

This parameter is only valid if *DataSource* is set to *DivaVoiceDataSourceFile*. It specifies the name of the file, typically a complete file name containing the path information.

DataLength

This parameter is only valid if *DataSource* is set to *DivaVoiceDataSourceMemory*. It specifies the size of the memory location where the audio information is stored.

pBuffer

This parameter is only valid if *DataSource* is set to *DivaVoiceDataSourceMemory*. In this case, it specifies the beginning of the data buffer where the audio information is available.

DivaConferencePropertyType

```
typedef enum
{
    DivaConferencePropertyMaxMembers = 1,
    DivaConferencePropertyOptions = 2,
    DivaConferencePropertyDefRights = 3,        // Set Get
    DivaConferencePropertyMemberRights = 4,     // Set Get
    DivaConferencePropertySupervisor = 5,       // Set
    DivaConferencePropertyNumMembers = 6,       // Get
    DivaConferencePropertyMembers = 7,          // Get
    DivaConferencePropertyNumTalkers = 8,       // Get
    DivaConferencePropertyTalkers = 9,          // Get
} DivaConferencePropertyType;
```

DivaConferencePropertyMaxMembers

The property *DivaConferencePropertiesMaxMembers* sets the maximum number of members that can participate in the conference. By default, there is no limitation.

DivaConferencePropertyOptions

The property *DivaConferencePropertiesOptions* specifies the options, e.g., if the conference is to be handled on the switch or on the board. For more information, see DivaConferenceOptions.

DivaConferencePropertyDefRights

The property *DivaConferencePropertyDefRights* specifies the default rights for a new member of the conference. For valid rights refer to DivaConferenceRights. The property can be "Get" and "Set".

DivaConferencePropertyMemberRights

The property *DivaConferencePropertyMemberRights* gets and sets the rights for the specified member. For valid rights refer to DivaConferenceRights. The parameter used to get and set the parameter is defined by *DivaConferenceMemberRights*.

DivaConferencePropertySupervisor

The property *DivaConferencePropertySupervisor* sets the supervisor options. This controls streaming of the supervisor to the conference and a specific member. The parameter used to set the parameter is defined by *DivaConferenceSupervisor*.

DivaConferencePropertyNumMembers

The property *DivaConferencePropertyNumMembers* returns the number of members that belong to this conference.

DivaConferencePropertyMembers

The property *DivaConferencePropertyMembers* returns the information about the members of the conference. The information is returned via *DivaConferenceMemberInfo*, one per member. The application has to provide a buffer that is large enough to place the information for all members. The first member in the list is the current conference master.

DivaConferencePropertyNumTalkers

The property *DivaConferencePropertyNumTalkers* returns the number of active talkers in this conference.

DivaConferencePropertyTalkers

The property *DivaConferencePropertyTalkers* returns the information about the active talker of the conference. The information is returned via *DivaConferenceMemberInfo*, one per talking member. The application has to provide a buffer that is large enough to place the information for all members.

DivaConferenceRights

```
typedef enum
{
        DivaConferenceRightSpeak = 1,
        DivaConferenceRightListen = 2,
        DivaConferenceRightSuvervisor = 7
} DivaConferenceRights;
```

DivaConferenceRightSpeak

The right *DivaConferenceRightSpeak* specifies that a member of the conference can speak to all members of the conference.

DivaConferenceRightListen

The right *DivaConferenceRightListen* specifies that a member of the conference can listen to a conference.

DivaConferenceRightSuvervisor

The right *DivaConferenceRightSuvervisor* specifies that a member of a conference can act as a supervisor. This grants the right to speak to the conference or to speak to specific members.

DivaConferenceMemberInfo

The type *DivaConferenceMemberInfo* is used to report member information of a conference.

```
typedef struct
{
        DivaCallHandle              hdCall;
        AppCallHandle               haCall;
        DivaConferenceRights        Rights;
        BOOL                        bTalking;
} DivaConferenceMemberInfo;
```

hdCall

The parameter *hdCall* specifies the Dialogic® Diva® SDK call handle of this member.

haCall

The parameter *haCall* specifies the application call handle of this member.

Rights

The parameter *Rights* hold the current right of the member. Refer to DivaConferenceRights for more information on conference rights.

bTalking

If *bTalking* is set to TRUE, the member is currently talking.

DivaConferenceMemberRights

The type *DivaConferenceMemberRights* is used to set and get member rights via the property *DivaConferencePropertyMembers*.

```
typedef struct
{
        DivaCallHandle              hdCall;
        DivaConferenceRights        Rights;
} DivaConferenceMemberRights;
```

hdCall

The parameter *hdCall* specifies the Dialogic® Diva® SDK call handle of this member.

Rights

The parameter *Rights* hold the current right of the member. Refer to DivaConferenceRights for more information conference rights.

DivaConferenceSupervisor

The type *DivaConferenceSupervisor* is used to set the supervisor options. Note that the member must have supervisor rights.

```
typedef struct
{
        DivaCallHandle              hdSupervisor;
        BOOL                        bEnable;
        DivaCallHandle              hdTalkTo;
} DivaConferenceSupervisor;
```

hdSupervisor

The parameter *hdSupervisor* specifies the Dialogic® Diva® SDK call handle of the supervisor.

bEnable

If *bEnable* is set to TRUE, the supervisor link between *hdSupervisor* and *hdTalkTo* is created, otherwise the link is disconnected. If the link is disconnected and the right *DivaConferenceRightSpeak* is assigned, the supervisor talks to all members.

hdTalkTo

The parameter *hdTalkTo* specifies the member that will be supervised.

DivaConferenceOptions

```
typedef enum
{
        DivaConferenceOptionLocal = 1
} DivaConferenceOptions;
```

DivaConferenceOptionLocal

The conference is handled by the Dialogic® Diva® board. This mode does not require any conference featues of the PBX or switch and is the default setting.

DivaConferenceState

```
typedef enum
{
        DivaConferenceStateIdle = 0,
        DivaConferenceStateOnHold,
        DivaConferenceStateAdding,
        DivaConferenceStateRemoving,
        DivaConferenceStateConnected
} DivaConferenceState;
```

DivaConferenceStateIdle

The conference is idle if only one two-way call is assigned to the conference. This is the case if the conference is created or if all other members have been removed.

DivaConferenceStateOnHold

The conference is on hold. This is typically done to create another call to be added to the conference.

DivaConferenceStateAdding

Another call is being added to the conference. While this happens, the other calls may be on hold.

DivaConferenceStateRemoving

The removal of a call is in process. While this happens, the other calls may be on hold.

DivaConferenceStateConnected

There are at least two calls part of the conference and the conference is active, voice streaming and mixing is in process.

DivaConferenceInfo

```
typedef struc
{
    // Size of the structure, may depend on the API version
    DWORD                   Size;
    DivaConferenceState     State;
    DWORD                   MaxMembers;
    DWORD                   CurrentMembers;
    DivaCallHandle          hdConfMembers[MAX_CONF_MEMBERS];
    AppCallHandle           haConfMembers[MAX_CONF_MEMBERS]
} DivaConferenceInfo;
```

Members

Size

The *Size* defines the length of the structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the version of the Dialogic® Diva® API the size may grow.

State

The *State* defines the current state of the conference. Valid conference states are listed in DivaConferenceState.

MaxMembers

The *MaxMembers* parameter defines the maximum number of calls including the initial call that can be part of this conference. This parameter is set when the conference is created. Note that using *DivaConferenceInfo* and *DivaGetConferenceInfo* creates a limitation to the maximum amount of members that can be handled. Using *DivaConferenceGetProperties* removes these limitation.

CurrentMembers

The *CurrentMembers* parameter provides the information on the current number of calls that belong to the conference. There will be at least one call.

hdConfMembers

The array *hdConfMembers* contains the Diva API handles of the calls that belong to the conference.

haConfMembers

The array *haConfMembers* contains the application handles of the calls that belong to the conference.

DivaCallPropertyType

The DivaCallPropertyTypes are used to specify the type of the call property that should be read by DivaGetCallProperties or written by DivaSetCallProperties.

```
/*
* Common call properties for all types of calls
*/
```

DivaCPT_CallType =1	// Get Set	This parameter specifies the basic type of a call and is available for reading and writing. The basic call type may be modified by additional properties, i.e. modem settings.
DivaCPT_LineDevice,	// Get Set	*DivaCPT_LineDevice* specifies the device on which a call is handled and is available for reading and writing. On setting the parameter, the value may be set to LINEDEV_ALL. In this case the Dialogic® Diva® API automatically selects a line device. On reading, the actual selected line device is reported.
DivaCPT_SignaledService,	// Get Set	*DivaCPT_SignaledService* specifies the service that is signaled from the network for an incoming call or that is signaled to the network for an outgoing call. The parameter is read and write.
DivaCPT_BearerCapabilities,	// Get Set	*DivaCPT_BearerCapabilities* provides the bearer capabilities signaled for an incoming call on reading and specifies the bearer capabilities to be used for an outgoing call. This parameter is read and write.
DivaCPT_CalledNumber,	// Get	*DivaCPT_CalledNumber* is a read only parameter and provides the called number signaled for an incoming call. The called number for an outgoing call is specified by the dial string of DivaDial or one of the DivaConnect functions.
DivaCPT_CallingNumber,	// Get Set	*DivaCPT_CallingNumber* is a read and write parameter. For an incoming call the calling number signaled from the network is provided on reading. For an outgoing call, the number is signaled to the remote peer which is specified by writing the property.
DivaCPT_CalledNumber Params,	// Get Set	*DivaCPT_CalledNumberParams* is a read and write parameter and sets / gets the parameter for a called number. For information on the parameter see DivaNumberInformation.
DivaCPT_CallingNumber Params,	// Get Set	*DivaCPT_CallingNumberParams* is a read and write parameter and sets / gets the parameter for a calling number. For information on the parameter see DivaNumberInformation.
DivaCPT_RedirectingNumber,	// Get	The parameter is read only and provides the redirected and redirecting number if available.
DivaCPT_RedirectedNumber,	// Get	The parameter is read only and provides the redirected and redirecting number if available.
DivaCPT_TxSpeed,	// Get	The parameter is read only and provides the transmit and receive speed for the call. Depending on the type of call the transmit and receive speed may be different.
DivaCPT_RxSpeed,	// Get	The parameter is read only and provides the transmit and receive speed for the call. Depending on the type of call the transmit and receive speed may be different.
DivaCPT_DiscReason,	// Get	The parameter is read only and returns the disconnect reason. Refer to DivaDisconnectReasons for possible options
DivaCPT_SignaledLineDisc Reason,	// Get	The parameter is read only and returns the disconnect reason in the format signaled by the line.
DivaCPT_RejectReason,	// Set	The parameter is write only and specifies the reject reason to be used when the call is rejected by *DivaReject*. Please note that the property must be set prior to call *DivaReject*.

```
/*
* Voice parameter
*/
```

DivaCPT_VoiceEchoCanceler,	// Set	The parameter is write only and enables the echo canceller for the next call initiated or answered on this call handle.
DivaCPT_VoiceEarlyData Channel,	// Set	The parameter is write only and enables the data channel before the connection in the signaling channel is established. The property is only valid for outgoing calls and must be set before the first call to *DivaDial*.

DivaCPT_VoiceRecordSilence Timeout,	// Set	The parameter is write only and specifies the period of silence before a recording to an audio file should be terminated. The property must be set prior to calling *DivaRecordVoiceFile.*

/*
* 2nd calling party number for SMS
*/

DivaCPT_SecondCalling Number,	// Get	The parameter is read only and provides the information about a second calling party number. A second calling party number may be signaled by SMS gateways.
DivaCPT_SecondCalling NumberParams,	// Get	The parameter is read only and provides the information about a second calling party number. A second calling party number may be signaled by SMS gateways.

/*
* Calling / connected name
*/

DivaCPT_CallingName,	// Get	The parameter is read only. On read it provides the calling name for an incoming call. On write it allows to set the name for an outgoing call. The availability of the name depends on the underlying network.
DivaCPT_ConnectedName,	// Get	The parameter is read only. When the call is connected the property provides the name of the connected party. The availability of the name depends on the underlying network.

/*
* Calling / sub address
*/

DivaCPT_CallingSubAddress,	// Get Set	The parameter provides the calling party address signaled on an incoming call or sets the calling party address for an outgoing call. This parameter is read and write.
DivaCPT_CalledSubAddress,	// Get Set	The parameter provides the called party address signaled on an incoming call or sets the called party address for an outgoing call. This parameter is read and write.
DivaCPT_OriginalCalled Number,	// Get	*DivaCPT_OriginalCalledNumber* is a read only property and specifies the number that the originator of the call has dialed. This number can be different from the calling party number and the redirecting number if the call has been redirected.
DivaCPT_ConnectedNumber,	// Get	*DivaCPT_ConnectedNumber* is a read only property and specifies the number of the endpoint that answered the call. This can be different from the called number if the call is redirected.
DivaCPT_CalledName,	// Get	*DivaCPT_CalledName* is a read only parameter and specifies the name of the endpoint that answered the call.
DivaCPT_DataChannel,	// Get Set	*DivaCPT_DataChannel* is a read and write property. The write operation is only valid for outgoing calls and specifies the data channel to be used. The property must be set prior to the first call to DivaDial. On read the property provides the data channel (B-channel) used for the call.

/*
* Disconnect reason and disconnect cause
*/

DivaCPT_DisconnectReason,	// Set	*DivaCPT_DisconnectReason* is a write only property to set the disconnect reason. For valid disconnect reasons, see DivaActiveDiscReasons. Note that the disconnect reason is only used for calls that have already been answered. Calls that are in the offering state can be disconnected using the reject reasons.
DivaCPT_DisconnectCause,	// Set	*DivaCPT_DisconnectCause* is a write only property to set the disconnect cause. This is the Q.931 cause value. Note that the disconnect cause is only used for calls that have already been answered. Calls that are in the offering state can be disconnected using the reject reasons.

/*
* Redirection number, to which number has the call been diverted.
*/

DivaCPT_RedirectionNumber,	// Get	This is a read only parameter that provides the redirection number if a call is transferred by the remote party.

/*
* Streaming properties
*/

DivaCPT_VoiceRecordStart Tones= 100,	// Set	The property is write only and defines a list of tones to trigger the recording. By default, recording initiated by *DivaRecordVoiceFile* starts right away. Setting a start tone delays the start until one of the tones is detected. The tones are coded as string containing the codes for the tones as 8 bit values. The string may contain any DTMF, continuous tone or MF tone. The application must enable DTMF and tone detection. The property is valid for the next call to *DivaRecordVoiceFile*.
DivaCPT_VoiceDTMF_Send Duration,	// Set	The property is write only and specifies the duration and pause of generated DTMF tones.
DivaCPT_VoiceDTMF_Send Pause,	// Set	The property is write only and specifies the duration and pause of generated DTMF tones.
DivaCPT_VoiceDTMF_Detect Duration,	// Set	The property is write only and specifies the duration and pause for DTMF tone detection. The properties must be set prior to the call to *DivaReportDTMF*.
DivaCPT_VoiceDTMF_Detect Pause,	// Set	The property is write only and specifies the duration and pause for DTMF tone detection. The properties must be set prior to the call to *DivaReportDTMF*.
DivaCPT_VoiceRemoveDTMF FromStream,	// Set	The property is write only. If enabled, DTMF tones are removed for the audio stream. The DTMF tones are still reported via events.
DivaCPT_VoiceEarlyDataDisc OnInfo,	// Set	*DivaCPT_VoiceEarlyDataDiscOnInfo* is a write only property and specifies that a connection established with the early data channel option is disconnected when the network signals the disconnect via info message. By default, the connection is kept open to allow the application to record and process any announcement or tones.
DivaCPT_EchoCanceller EnableNLP,	// Set	The property is write only and enables the non-linear processing for the echo canceller.
DivaCPT_EchoCancellerAuto Disable1,	// Set	The property is write only and bypasses the echo canceller upon detection of phase reversed 2100 Hz (operation according to G.165).
DivaCPT_EchoCancellerAuto Disable2,	// Set	The property is write only. It bypasses the echo canceller upon detection of phase reversed or phase continuous 2100 Hz (operation according to G.164 and G.165).
DivaCPT_EchoCancellerTail Length,	// Set	The property is write only. Echo canceller time span in milliseconds, default is implementation-specific.
DivaCPT_EchoCancellerPre Delay,	// Set	The property is write only. Echo canceller pre-delay before starting.
DivaCPT_EnableDTMFTrailing Edge,	// Set	The property is a write only and enables the reporting of the training edge of a DTMF tone. The default is disabled.

/*
* Fax parameter
*/

DivaCPT_FaxLocalId = 200,	// Set	The property is write only and specifies the local identifier to be used in the fax communication. The property is only valid for fax communication.
DivaCPT_FaxHeadline,	// Set	The parameter is write only and specifies the headline to be printed on top of every fax page to be send. The property is only valid for fax communication.
DivaCPT_FaxRemoteId,	// Get	The property is read only and returns the identifier of the remote fax machine. The property is only valid for fax communication.
DivaCPT_FaxPages,	// Get	The property is read only and provides the fax page currently processed. The property is only valid for fax communication.
DivaCPT_FaxMaxSpeed,	// Set	The parameter is write only and defines the maximum fax speed to be negotiated. The property is only valid for fax communication.
DivaCPT_FaxHighResolution,	// Set	The parameter is write only and enables the negotiation of the high resolution. The used format depends on the remote capabilities. The property is only valid for fax communication.
DivaCPT_FaxEnablePolling,	// Set	The parameter is write only and enables the polling mode for fax communication. The property is only valid for fax communication.

DivaCPT_FaxReverseSession,	// Set	The parameter is write only and enables the reverse session used for fax on demand. The property is only valid for fax communication.
DivaCPT_FaxMultiDocument,	// Set	The property is write only and sets the processing of multiple fax files or multiple documents. Refer to *DivaSendMultipleFaxFiles* for comments on multi document support. The property is only valid for fax communication.
DivaCPT_FaxDisableECM,	// Set	The property is write only and disables ECM mode. The property is only valid for fax communication.
DivaCPT_FaxDisableMR,	// Set	The property is write only and disables MR mode. The property is only valid for fax communication.
DivaCPT_FaxDisableMMR,	// Set	The property is write only and disables MMR mode. The property is only valid for fax communication.
DivaCPT_FaxPageQuality,	// Get	*DivaCPT_FaxPageQuality* is a read parameter and only valid in fax mode. The parameter is updated every time a fax page is received or sent. For information on page quality, refer to DivaFaxPageQuality.
DivaCPT_FaxPageEndInfo,	// Get	*DivaCPT_FaxPageQuality* is a read parameter and only valid in fax receive mode. The parameter is updated every time a fax page is received. The parameter provides information on coming pages or documents. For information on valid page ends, refer to DivaFaxPageEnd.
DivaCPT_FaxRemote Features,	// Get	*DivaCPT_FaxRemoteFeatures* is a read only property and provides the binary coded capabilities of the receiving fax station. The information is coded in accordance with T.30 DIS and DTC frame.
DivaCPT_FaxRemoteMaxHorz Res,	// Get	*DivaCPT_FaxRemoteMaxHorzRes* is a read only property and provides the maximum horizontal resolution the receiving fax station can support. The value is given as pixel per line.
DivaCPT_FaxRemoteMaxVert Res,	// Get	*DivaCPT_FaxRemoteMaxVertRes* is a read only property and provides the maximum horizontal resolution the receiving fax station can support. The value is given as pixel per line.
DivaCPT_FaxRemoteMax Speed,	// Get	*DivaCPT_FaxRemoteMaxSpeed* is a read only property and provides the maximum speed the receiving fax station can support. Please note that this is not the finally negotiated speed because this depends on the line quality.
DivaCPT_FaxRemoteNSF,	// Get	*DivaCPT_FaxRemoteNSF* is a read only property and provides the non standard facilities received from the remote fax station. The data is provided as binary data, first byte length field.
DivaCPT_FaxLocalNSF,	// Set	*DivaCPT_FaxLocalNSF* is a write only property and specifies the non standard facilities to be send to the remote fax station. The data is expected as binary data, first byte length field.
DivaCPT_FaxEnableColor,	// Set	*DivaCPT_FaxEnableColor* is a write only property. If set, the color fax capabilities are signaled for incoming fax calls.
DivaCPT_FaxColorSelected,	// Get	*DivaCPT_FaxColorSelected* is a read only property and specifies that the fax negation results in sending a color fax document. The application must pass a document in the color fax JPEG format using the option *DivaFaxFormatColorJPEG*.
DivaCPT_EnableInterrupt,	// Set	The property is write only and enables the fax procedure interrupt. The usage is depending on the remote peer. The property *DivaCPT_FaxProcedureInterrupt* returns the result.
DivaCPT_RequestInterrupt,	// Set	The property is write only and requests the fax procedure interrupt. The usage is depending on the remote peer. The property *DivaCPT_FaxProcedureInterrupt* returns the result.
DivaCPT_FaxProcedure Interrupt,	// Get	The property is read only and returns the state of the procedure interrupt negotiation. The property can only be negotiated if the property *DivaCPT_RequestInterrupt* or *DivaCPT_FaxProcedureInterrupt* are enabled.
DivaCPT_FaxEnableSecurity,	// Set	The call property is write only and enables the negotiation of the secure fax options. The usage of the option depends on the remote peer.
DivaCPT_FaxRemote SupportsSubaddr,	// Get	The property is read only and provides the information if the remote party is able to handle secure fax protocols.
DivaCPT_FaxRemote SupportsPassword,	// Get	The property is read only and provides the information if the remote party is able to handle secure fax protocols.
DivaCPT_FaxSignalSub Address,	// Set	The property is write only and specifies the sub address and password to be send to the remote end within the Fax T.30 negotiation.
DivaCPT_FaxSignalPassword,	// Set	The property is write only and specifies the sub address and password to be send to the remote end within the Fax T.30 negotiation.

DivaCPT_FaxRemoteSub Address,	// Get	The property is read only and provides the sub address and password of the remote party negotiated during fax T.30 negotiation.
DivaCPT_FaxRemote Password,	// Get	The property is read only and provides the sub address and password of the remote party negotiated during fax T.30 negotiation.
DivaCPT_FaxDisableFile Buffering	// Set	The property disables the internal buffering of fax data to a temporary file. By default, the Dialogic® Diva® SDK buffers data to memory and also to file if the application does not call DivaReceiveFax fast enough to avoid loss of data. If this option is set, the file buffering will be disabled. Note that the application must ensure that DivaReceiveFax or DivaReceiveFaxToMemory is called shortly after the event DivaEventCallConnected is reported. The property is write only.
DivaCPT_FaxUseTextFor Sending	// Set	If this property is used before initiating a fax connection or changing the mode to fax transmission, the expected document format is plain ASCII text. The property is write only.
DivaCPT_FaxAllowDocument Stretching	// Set	If this option is selected before calling *DivaSendFax*, *DivaAppendFax*, or *DivaSendMultipleFaxFiles*, a TIFF document provided in a resolution that is half of the next matching fax format will be stretched. e.g., a document with a resolution of 800 pixels per line will be stretched to 1600 pixels per line and centered on the next matching resolution of 1728 pixels per line. The property is write only.
DivaCPT_FaxRemoteScan LineLength	// Get	The property is read only and provides the maximum scan line length the receiving fax station can support. The value is given as *DivaFaxScanLineMax*.
DivaCPT_FaxStoreMode	// Get	*DivaCPT_FaxStoreMode* is a write only property and specifies how single pages of a received fax are stored. For possible values, refer to *DivaFaxStoreModes*.

/*
* Calls that allow certain different speeds and framing
* Analog calls and asynchronous V.110
*/

DivaCPT_Maximum Speed =400,	// Set	*DivaCPT_MaximumSpeed* is a write only property and defines the maximum speed that is allowed for the connection. The parameter is only valid for analog modem and V.110 types. The real negotiated speed can be retrieved by the *DivaCPT_TxSpeed* and *DivaCPT_RxSpeed* properties.
DivaCPT_DataBits,	// Get Set	The property is read and write and sets / gets the framing for an asynchronous connection.
DivaCPT_StopBits,	// Get Set	The property is read and write and sets / gets the framing for an asynchronous connection.
DivaCPT_Parity,	// Get Set	The property is read and write and sets / gets the framing for an asynchronous connection.

/*
* Modem parameters
*/

DivaCPT_Disable Compression =800,	// Set	The property is write only and disables any compression for an analog modem connection.
DivaCPT_DisableV42,	// Set	The property is write only and disables any V.42 or MNP negotiation for an analog modem connection.
DivaCPT_DisableMNP,	// Set	The property is write only and disables any V.42 or MNP negotiation for an analog modem connection.
DivaCPT_ForceReliable,	// Set	The property is write only and valid for call type modem. If set, a reliable connection using V.42 or MNP is negotiated. If the remote peer is not able to handle one of these protocols, the connection will fail.
DivaCPT_DisableRetrain,	// Set	The property is write only and disables the retrain for an analog modem connection.
DivaCPT_ModulationClass,	// Set	The property is write only and valid for call type modem. It sets the modulation class between V.8 and V.110. The options are defined in *DivaModulationClass*.
DivaCPT_NegotiatedV42V42 bis	// Get	The property is read only and valid only for analog modem connections. If the property is set, the negotiation succeeds in the specified reliable protocol. If *DivaCPT_NegotiatedCompression* is also set, the corresponding compression, V.42bis or MNP5, is also negotiated.

DivaCPT_Negotiated MNP4MNP5	// Get	The property is read only and valid only for analog modem connections. If the property is set, the negotiation succeeds in the specified reliable protocol. If *DivaCPT_NegotiatedCompression* is also set, the corresponding compression, V.42bis or MNP5, is also negotiated.
DivaCPT_Negotiated Transparent	// Get	The property is read only and valid only for the call type modem. If the property is set, the modem connection is negotiated without using any reliable protocol.
DivaCPT_Negotiated Compression	// Get	The property is read only and valid only for the call type modem. If the property is set, the modem connection is negotiated using a compression protocol.
DivaCPT_DCD,	// Get	The property is read only and valid only for the call type modem. *DivaCPT_DCD* reflects the state of the DCD modem signal.
DivaCPT_CTS,	// Get	The property is read only and valid only for the call type modem. *DivaCPT_CTS* reflects the state of the CTS signal. **Note:** The CTS signal is only provided if the call type is modem and any of the extended modem settings have been enabled.
DivaCPT_ConnectedNorm,	// Get	The property is read only and valid only for the call type modem selected via the extended modem settings. The property holds the modulation result. For valid options, see DivaConnectedNorm.
DivaCPT_RoundTripDelay,	// Get	The property is read only and available for modem connections using V.34 modulation. The property is set when the DCD information is available and contains the time for receiving the echo of a signal set to the remote peer.
/* * Extended modem parameters */		
DivaCPT_GuardTone,	// Set	The property is write only. Specifies the modem guard tone. A value of zero selects no guard tone, one is for a 1800 Hz guard tone and two for a 550 Hz guard tone.
DivaCPT_ModemLeased Line,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_Modem4Wire Option,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableDiscOn BusyTone,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableCalling Tone,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableAnswer Tone,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableDialTone Detect,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableStepUp,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableStepDown,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableSpiltSpeed,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableTrellis Coding,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableFlush Timer,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableEmpty Frames,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_Enable Multimoding,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_BypassControl,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation V21,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation V22,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

DivaCPT_DisableModulation V22bis,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation V23,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation V32,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation V32bis,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation V34,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation V90DPCM,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation Bell103,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation Bell212A,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation AllFS,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation K56Flex,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulation X2,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV42 Detection,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulation V29FDX,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulation V33,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulation V90APCM,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulation V22FS,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulation V29FS,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulation V23_1,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulation V23_2,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_MinimumTxSpeed,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_MaximumTxSpeed,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_MinimumRxSpeed,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_MaximumRxSpeed,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_TxLevelAdjust,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34Tx LevelReduction,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34Pre Coding,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34Pre Emphasis,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34 Shaping,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34Non LinearEncoding,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34 ManualReduction,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

DivaCPT_DisableV34 Training16Point,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34 SymbolRate2400,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34 SymbolRate2743,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34 SymbolRate2800,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34 SymbolRate3000,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34 SymbolRate3200,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34 SymbolRate3429,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_ForceReliableIfV34,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableSDLC,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableReliable If1200,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_BufferDuringV42 Detection,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV42 SelectivReject,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableMNP3,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableMNP4,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableMNP10,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_TransparentMode IfV22bis,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_TransparentMode IfV32bis,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_BreakMode,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_ModemEarly Connect,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_ModemPass Indication,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCLinkAddress,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCModuloMode,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCWindowSize,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCXID,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCReverse Establishment,	// Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

/*
* V.18 parameters
*/

DivaCPT_V18Selected,	// Set	The property is write only and enables the V.18 mode.
DivaCPT_V18Probing Sequence,	// Set	Text of the message string used for probing. The property is write only.
DivaCPT_V18CountryProbing Sequence,	// Set	Pre-defined country probing sequences. For available countries refer to DivaV18DefProbings. The property is write only.

DivaCPT_V18Probing Message,	// Set	Array of bytes containing the sequence of modulation norm identifiers that specifies the order used in answer mode probing. The property is write only.
DivaCPT_V18ReinitializeOn Silence,	// Set	The property is write only and enables or disables the reinitialization on silence.
DivaCPT_V18RevertTo AnswerMode,	// Set	The property is write only and enables or disables the revert to answer mode on timeout.
DivaCPT_V18DisconnectOn Busy,	// Set	The property is write only and enables or disables disconnect on busy detection.
DivaCPT_V18Automoding MonitorMode,	// Set	The property is write only and enables or disables automoding monitor.
DivaCPT_V18TextProbing ForCarrierMode,	// Set	The property is write only and enables or disables continuous carrier probing with the message.
DivaCPT_V18TXPSpace ParityInOrigMode,	// Set	The property is write only and enables or disables the sending of TXP with space parity in origination mode.
DivaCPT_V18EnableV18 OriginationMode,	// Set	The property is write only and enables the V.18 originate mode (CI/TXP procedure, V.21 data state, TX: 980/1180 Hz 300 bit/s, RX: 1650/1850 Hz 300 bit/s).
DivaCPT_V18EnableV18 AnswerMode,	// Set	The property is write only and enables V.18 answer mode (CI/TXP procedure, V.21 data state, TX: 1650/1850 Hz 300 bit/s, RX: 980/1180 Hz 300 bit/s).
DivaCPT_V18EnableV21 OriginationMode,	// Set	The property is write only and enables V.21 originate mode (TX: 980/1180 Hz 300 bit/s, RX: 1650/1850 Hz 300 bit/s).
DivaCPT_V18EnableV21 AnswerMode,	// Set	The property is write only and enables V.21 answer mode (TX: 1650/1850 Hz 300 bit/s, RX: 980/1180 Hz 300 bit/s).
DivaCPT_V18EnableBell103 OrigMode,	// Set	The property is write only and enables Bell 103 originate mode (TX: 1270/1070 Hz 300 bit/s, RX: 2225/2025 Hz 300 bit/s).
DivaCPT_V18EnableBell103 AnswerMode,	// Set	The property is write only and enables Bell 103 answer mode (TX: 2225/2025 Hz 300 bit/s, RX: 1270/1070 Hz 300 bit/s).
DivaCPT_V18EnableV23 OriginationMode,	// Set	The property is write only and enables V.23 originate mode (TX: 390/450 Hz 75 bit/s, RX: 1300/1700 Hz 1200 bit/s).
DivaCPT_V18EnableV23 AnswerMode,	// Set	The property is write only and enables V.23 answer mode (TX: 1300/1700 Hz 1200 bit/s, RX: 390/450 Hz 75 bit/s).
DivaCPT_V18EnableEDT Mode,	// Set	The property is write only and enables EDT mode (980/1180 Hz 110 bit/s).
DivaCPT_V18EnableBAUDOT 45Mode,	// Set	The property is write only and enables BAUDOT 45 mode (1800/1400 Hz 22 ms/bit).
DivaCPT_V18EnableBAUDOT 47Mode,	// Set	The property is write only and enables BAUDOT 47 mode (1800/1400 Hz 21 ms/bit).
DivaCPT_V18EnableBAUDOT 50Mode,	// Set	The property is write only and enables BAUDOT 50 mode (1800/1400 Hz 20 ms/bit).
DivaCPT_V18EnableDTMF Mode,	// Set	The property is write only and enables DTMF mode (DTMF 50ms on / 50ms off).
DivaCPT_V18TransmitLevel,	// Set	The property is write only. Transmits level in dBm, coded as 2-s complement signed integer. Valid range: -12..-31 dBm Value 0 will set the default.
DivaCPT_V18AsyncFormat V21,	// Set	Asynchronous data format used in V.18 and V.21 mode. The coding is described in DivaV18Framing. Value 0 will set the default. The property is write only.
DivaCPT_V18AsyncFormat V23,	// Set	Asynchronous data format used in V.23 mode. The coding is described in DivaV18Framing. Value 0 will set the default. The property is write only.
DivaCPT_V18AsyncFormat Bell103,	// Set	Asynchronous data format used in Bell 103 mode. The coding is described in DivaV18Framing. Value 0 will set the default. The property is write only.
DivaCPT_V18AsyncFormat EDT,	// Set	Asynchronous data format used in EDT mode. The coding is described in DivaV18Framing. Value 0 will set the default. The property is write only.
DivaCPT_V18AsyncFormat BAUDOT,	// Set	Asynchronous data format used in BAUDOT modes. The coding is described in DivaV18Framing. Value 0 will set the default. The property is write only.
DivaCPT_V18TimerTc Timeout,	// Set	Timeout time for ITU-T V.18 timer Tc in milliseconds. (Tc specifies the maximum time waiting for response when sending a probing carrier in answer mode). Value 0 will set the default. The property is write only.
DivaCPT_V18TimerTm Timeout,	// Set	Timeout time for ITU-T V.18 timer Tm in milliseconds. (Tm specifies the maximum time waiting for response after a probing message has been sent in answer mode). Value 0 will set the default. The property is write only.

DivaCPT_V18CleanCarrier Time,	// Set	Time span in milliseconds for which the carrier is maintained in half duplex modes after the last pending character has been sent to the line. Value 0 will set the default. The property is write only.
DivaCPT_V18EchoSupress Time,	// Set	Time span in milliseconds for which the receiver is disabled in half duplex mode after the last send period in order to avoid interpretation of the echo signal. Value 0 will set the default. The property is write only.

/*
* Plain setting of all protocol parameters
*/

DivaCPT_B1Protocol = 1200,	// Set	The property is write only. The B1 protocol according to the CAPI 2.0 specification. For more information, see Plain Protocol parameter setting.
DivaCPT_B2Protocol,	// Set	The property is write only. The B2 protocol according to the CAPI 2.0 specification. For more information, see Plain Protocol parameter setting.
DivaCPT_B3Protocol,	// Set	The property is write only. The B3 protocol according to the CAPI 2.0 specification. For more information, see Plain Protocol parameter setting.
DivaCPT_B1Configuration,	// Set	The property is write only. The B1 configuration options according to the CAPI 2.0 specification. For more information, see Plain Protocol parameter setting.
DivaCPT_B2Configuration,	// Set	The property is write only. The B2 configuration options according to the CAPI 2.0 specification. For more information, see Plain Protocol parameter setting.
DivaCPT_B3Configuration,	// Set	The property is write only. The B3 configuration options according to the CAPI 2.0 specification. For more information, see Plain Protocol parameter setting.
DivaCPT_LLC,	// Get Set	Sets the Low Layer Compatibility Information for an outgoing call or retrieves them from an incoming call. The element is coded according to Q.931. The property is read and write.
DivaCPT_HLC,	// Get Set	Sets the High Layer Compatibility Information for an outgoing call or retrieves them from an incoming call. The element is coded according to Q.931. The property is read and write.
DivaCPT_B_ChannelInfo,	// Set	The property is write only and provides a flexible setting of the B-channel information. This can be used to select a specific channel for an outgoing call or to connect a special channel in leased line mode. The coding is done according to the CAPI Specification.
DivaCPT_KeypadFacility,	// Get Set	The property is read and write and gets or sets the keypad facility information for a setup message. The element is coded according to Q.931.
DivaCPT_UserUserInfo,	// Get Set	The property is available for read and write and sets the User User Information for an outgoing call or retrieves them from an incoming call. The element is coded according to Q.931. If the user-user information is set before calling *DivaAlert,* the information are passed in the alert message.
DivaCPT_FacilityDataArray,	// Get Set	The property is available for read and write an sets the Facility Data Information for an outgoing call or retrieves them from an incoming call. The element is coded according to Q.931.
DivaCPT_DisplayInfo,	// Get	The property is read only and reads the display information received from an incoming call.
DivaCPT_TotalChargeUnits,	// Get	The property is read only and provides the amount of charge units reported by the network.
DivaCPT_SpecialInfo Element,	// Get Set	The property is available for read and write and is used for setting-specific elements.
DivaCPT_ChannelInfo Element,	// Get	The property is read only and provides the channel information element as received from the line.
DivaCPT_ProgressInd Element,	// Get	The property is read only and provides the progress information element as received from the line.
DivaCPT_SetupMessage,	// Get	The property is read only and provides the recreated setup messages. Note that this is not the original setup message.
DivaCPT_Global Configuration,	// Set	The parameter is write only and allows to modify the "global configuration" option of the underlying CAPI interface. Currently the B-channel operation mode can be switched to DCT or DTE by this parameter.
DivaCPT_ReverseData ChannelConnect,	// Set	The parameter is write only and specifies that the data channel connection is not initiated by the side that has initiated the physical connection.
DivaCPT_CauseInfo Element	// Get	*DivaCPT_CauseInfoElement* is a read only property and provides the cause information element as signaled on the underlying network.

/*		
* B channel negotiation		
*/		
DivaCPT_X25_NCPI = 2000,	// Get Set	The property is available for read and write. It provides the ability to set and get the plain X.25 information exchanged between the endpoints.
DivaCPT_X25_Called Address,	// Get Set	The property is available for read and write and sets or retrieves the X.25 addresses for a call using call type DivaCallTypeX25.
DivaCPT_X25_Calling Address,	// Get Set	The property is available for read and write and sets or retrieves the X.25 addresses for a call using call type DivaCallTypeX25.
DivaCPT_X25_Reverse Restart	// Set	DivaCPT_X25_ReverseRestart is a write only property and enables the X.25 restart sequence. The property must be set before the call is initiated or answered.
/*		
* Autodetection		
*/		
DivaCPT_AutoDetectMode =2400,	// Set	The property is reserved for future use.
DivaCPT_AutoDetectX75 ForceX25,	// Set	The property is write only. If specified, the autodetect mode for digital protocols interprets X.25 frames in layer 2 as X.25 connections.
DivaCPT_AutoDetectMax Frames,	// Set	The property is write only and specifies the maximum amount of frames that should be used for autodetection of a digital protocol. The property is only valid if the call type is set to *DivaCallTypeAutoDetect*.
DivaCPT_AutoDetectMax Seconds,	// Set	The property is write only and specifies the maximum amount of seconds for the autodetect process. The property is only valid if the call type is set to *DivaCallTypeAutoDetect*.
/*		
* Special settings		
*/		
DivaCPT_UseSameChannel ForTransfer=4000,	// Set	The property is write only. If set to true for an existing call, a consultation call initiated via *DivaSetupCallTransfer* or *DivaBlindCallTransfer* uses the same B-channel. By default, the channel is assigned by the switch.
DivaCPT_NoAnswerTimeout,	// Set	The property is write only and specifies the amount of time in seconds to wait until the remote side picks up the call.
DivaCPT_ConnectTimeout,	// Set	The property is write only and specifies the amount of time in seconds to wait until an answered call reaches the connect state. This is typically the time to negotiate a modem or fax connection.
DivaCPT_NoHoldBefore Transfer,	// Set	*DivaCPT_NoHoldBeforeTransfer* is a write only property and specifies that the consultation call created via *DivaSetupCallTransfer* is established without setting the primary call on hold. This implies that the consultation call is done on a different channel
/*		
* Special hardware capability-dependent settings		
*/		
DivaCPT_UseExternal Equipment= 5000,	// Set	The property is write only and enables or disables the usage of the external equipment for a call. The default is disabled.
/*		
* Statistics		
*/		
DivaCPT_CallTimeStats = 5200,	// Get	The property is read only and only valid in monitoring mode. It provides the timing information about the detection of the different signaling messages for the monitored call.

DivaFaxPageQuality

```
typedef enum
{
        DivaFaxPageQualityPerfect,
        DivaFaxPageQualityGood,
        DivaFaxPageQualityAcceptable,
        DivaFaxPageQualityReject
} DivaFaxPageQuality;
```

Members

DivaFaxPageQualityPerfect

The page has been received without any errors.

DivaFaxPageQualityGood

The page may contain a few error lines, the overall quality is good.

DivaFaxPageQualityAcceptable

The page contains error line but the quality is acceptable.

DivaFaxPageQualityReject

The page contains too many errors and has been rejected. Behavior depends on the remote peer.

DivaFaxPageEnd

```
typedef enum
{
        DivaFaxPageEndUndefined,
        DivaFaxPageEndMPS,
        DivaFaxPageEndEOM,
        DivaFaxPageEndEOP
} DivaFaxPageEnd;
```

Members

DivaFaxPageEndUndefined

The page end information is not available.

DivaFaxPageEndMPS

Another page that belongs to the same document will follow.

DivaFaxPageEndEOM

Another page will follow. The page belongs to a new document.

DivaFaxPageEndEOP

This is the last page. The disconnect will follow.

DivaModulationClass

```
typedef enum
{
        DivaModulationClassNone,
        DivaModulationClassInClass,
        DivaModulationClassV100,
        DivaModulationClassV8
} DivaModulationClass;
```

Members

DivaModulationClassNone

The speed negotiation is done without any modulation class. This allows speeds to 14.400 bps.

DivaModulationClassInClass

The speed negotiation is done within the current active modulation class. The speed negotiation is done within the modulation class defined by V.100. This covers speeds up to V.32

DivaModulationClassV8

The speed negotiation is done within the modulation class defined by V.8. This covers speeds of V.34 and above.

See also

Extended modem parameters, V18 Properties, Plain Protocol parameter setting

Extended modem parameters

The properties for the extended modem parameters allow the setting of certain modem parameters. Working with these parameters requires knowledge of modulation and modem protocols. The names of the parameters are self-explaining. For details on modulation and protocols, refer to standard modem documentation.

The disabled modulation properties are used to remove the specified modulation(s) from the automoding procedure. To enable specific modulations that are not included in the automoding procedure, the enable modulation properties are used.

V18 Properties

The properties for V.18 are used for text phone modes. V.18 modulation is part of the extended modem functionality and based on the call type *DivaCallTypeModem*. To enable V.18 the property *DivaCPT_V18Selected* must be set. The different modulations may be selected by the application and the negotiated mode is available via the property *DivaCPT_ConnectedNorm.*

The probing sequence can be selected directly or by a country default. The country defaults are selected via the property *DivaCPT_V18CountryProbingSequence*. According to ITU-T V.18 they have the following sequence:

Value for Property DivaCPT_V18CountryProbingSequence	**Probing Sequence**
V18DefProbingAustria	10, 3, 7, 8, 12, 5
V18DefProbingIreland	10, 3, 7, 8, 12, 5
V18DefProbingGermany	8, 3, 7, 10, 12, 5
V18DefProbingSwiss	8, 3, 7, 10, 12, 5
V18DefProbingItaly	8, 3, 7, 10, 12, 5
V18DefProbingNetherlands	12, 3, 7, 10, 8, 5
V18DefProbingScandinavian	3, 12, 10, 8, 7, 5
V18DefProbingUK	3, 10, 7, 8, 12, 5
V18DefProbingUS	10, 5, 3, 7, 8, 12
V18DefProbingFrance	7, 8, 12, 10, 3, 5
V18DefProbingBelgium	7, 8, 12, 10, 3, 5

Plain Protocol parameter setting

If an application wants to select specific protocol settings for layer 1, layer 2, and layer 3 that are not covered by the call types and additional properties, it may use the plain protocol parameter setting. The protocol settings and configuration has to be coded according to the CAPI specification or Dialogic-specific extensions documented in the CAPI extensions.

DivaBinaryData

The data type *DivaBinaryData* is used for setting and reading call properties, i.e., bearer capabilities.

```
typedef struct
{
        unsigned char           nDataLength;
        unsigned char           Data[255]
} DivaBinaryData;
```

Members

nDataLength

The *nDataLength* specifies the amount of bytes stored in the member *Data*.

Data

The *Data* member contains the binary information.

DivaPlainNumber

```
typedef struct
{
        unsigned char           Number[MAX_ADDR_LEN]
} DivaPlainNumber;
```

Members

Number

The *Number* specifies called, calling or redirecting numbers. This represents only the digits, not the information of the number. The number information is available in the *DivaNumberInformation* property.

DivaNumberInformation

```
typedef struct
{
        DWORD TypeOfNumber;
        DWORD NumberIdentification;
        DWORD Presentation;
        DWORD Screening
} DivaNumberInformation;
```

Members

TypeOfNumber

The *TypeOfNumber* specifies how the number should be interpreted, i.e. if the number is handled as international or national number. Valid values are defined in *DivaNumberType*. Detailed information is available in the ISDN specifications.

NumberIdentification

The *NumberIdentification* specifies how the number should be interpreted. Valid values are defined in *DivaNumberId*. Detailed information is available in the ISDN specifications.

Presentation

The *Presentation* specifies if the number should be presented or not. The parameter is only available for calling numbers. Please note that the presentation information must be supported by the switch, it is not guaranteed that setting this information at Dialogic® Diva® SDK level suppresses presentation. Valid values are defined in *DivaNumberPresentation*.

Screening

The *Screening* values are defined in *DivaNumberScreening*. Detailed information about screening is available in the ISDN specifications.

DivaCallPropertyValue

The *DivaCallPropertyValue* is used to read and write various call properties. The union covers all types and is defined in the Dialogic® Diva® SDK header files. For a description of the call properties see DivaCallPropertyType.

Applications may declare a variable of type *DivaCallPropertyValue* and set the appropriate member corresponding to the used *DivaCallPropertyType* or use directly a variable of the type corresponding to the property type, i.e. BOOLEAN. The API will internally check only for the needed size corresponding to the property type. If a direct type is used, this needs to be casted to *DivaCallPropertyValue* before calling the get / set function.

DivaV18DefProbings

The *V18DefProbings* specify the available options for the property *DivaCPT_CountryProbingSequence*. The names of the options contain the country and are self explaining. For information on the V.18 probing settings behind the country default see the remarks of DivaCallPropertyType.

```
typedef enum
{
        V18DefProbingNone = 0,
        V18DefProbingAustria,
        V18DefProbingIreland,
        V18DefProbingGermany,
        V18DefProbingSwiss,
        V18DefProbingItaly,
        V18DefProbingNetherlands,
        V18DefProbingScandinavian,
        V18DefProbingUK,
        V18DefProbingUS,
        V18DefProbingFrance,
        V18DefProbingBelgium
} DivaV18DefProbings;
```

DivaV18Framing

The *V18Framing* specifies the framing options for the properties *DivaCPT_V18AsyncFormatxxx*, where xxx is the modulation. The framing is selected by combining the options for data bits, parity and stop pits, i.e. V18DataBits8 | V18ParityNo | V18StopBits1.

```
typedef enum
{
        V18DataBits4                    =0x0004,
        V18DataBits5                    =0x0005,
        V18DataBits6                    =0x0006,
        V18DataBits7                    =0x0007,
        V18DataBits8                    =0x0008,
        V18ParityNo                     =0x0000,
        V18ParityOdd                    =0x0010,
        V18ParityEven                   =0x0020,
        V18ParityMark                   =0x0030,
        V18ParitySpace                  =0x0040,
        V18EnableRxParityCheck          =0x0080,
        V18StopBits1                    =0x0000,
        V18StopBits1_5                  =0x0010,
        V18StopBits2                    =0x0020
} Diva V18Framing;
```

DivaConnectedNorm

DivaConnectedNorm specifies the modulation result of the modem or V.18 connection. The connected norm can be read by the property *DivaCPT_ConnectedNorm*.

```
typedef enum
{
        ConnNormUnspecified             =0,
        ConnNormV21                     =1,
        ConnNormV23                     =2,
        ConnNormV22                     =3,
        ConnNormV22bis                  =4
        ConnNormV32bis                  =5,
        ConnNormV34                     =6,
        ConnNormBell212A                =8,
        ConnNormBell103                 =9,
        ConnNormV29LeasedLine           =10,
        ConnNormV33LeasedLine           =11,
        ConnNormV90                     =12,
        ConnNormV32                     =18,
        ConnNormK56Flex                 =19,
        ConnNormX2                      =20,
        ConnNormTxtPhoneUnspecified     =21,
        ConnNormTxtPhoneV18Org          =22,
        ConnNormTxtPhoneV18Ans          =23,
        ConnNormTxtPhoneV21Org          =24,
        ConnNormTxtPhoneV21Ans          =25,
        ConnNormTxtPhoneBell103Org      =26,
        ConnNormTxtPhoneBell103Ans      =27,
        ConnNormTxtPhoneV23Org          =28,
        ConnNormTxtPhoneV23Ans          =29,
        ConnNormTxtPhoneEDT             =30,
        ConnNormTxtPhoneBAUDOT45        =31,
        ConnNormTxtPhoneBAUDOT47        =32,
        ConnNormTxtPhoneBAUDOT50        =33,
        ConnNormTxtPhoneDTMF            =34,
        ConnNormV23ETS300659_2          =35,
        ConnNormV23ETS300659_1          =36,
        ConnNormV22FastSetup            =37,
        ConnNormV22bisFastSetup         =38,
        ConnNormV29FastSetup            =39
} DivaConnectedNorm;
```

DivaMonitorSource

```
typedef enum
{
        DivaMonitorSourceOriginator,
        DivaMonitorSourceAnswerer,
        DivaMonitorDeviceAToDeviceB,
        DivaMonitorDeviceBToDeviceA,
        DivaMonitorSourceBoth
} DivaMonitorSource;
```

DivaMonitorSourceOriginator

The audio streamed from the originator of a call is recorded.

DivaMonitorSourceAnswerer

The audio streamed from the answerer of a call is recorded.

DivaMonitorSourceBoth

The audio streamed from both parties is recorded to an audio file with two channels.

DivaMonitorDeviceAToDeviceB

The audio streamed from the line device A. The device is specified during creation of the monitor object by DivaCreateMonitor.

DivaMonitorDeviceBToDeviceA

The audio streamed from the line device B. The device is specified during creation of the monitor object by DivaCreateMonitor.

DivaMonitorStatus

```
typedef enum
{
        DivaMonitorStarted = 1,
        DivaMonitorStopped,
        DivaMonitorErrorStarting,
        DivaMonitorLayer1Up,
        DivaMonitorLayer1Down
} DivaMonitorStatus;
```

DivaTime

The Dialogic® Diva® API reports time stamps via the *DivaTime* format. The members are self-explanatory.

```
typedef struct
{
        WORD        wYear;
        WORD        wMonth;
        WORD        wDayOfWeek;
        WORD        wDay;
        WORD        wHour;
        WORD        wMinute;
        WORD        wSecond;
        WORD        wMilliseconds;
} DivaTime;
```

DivaCallTimeStatistics

The Dialogic® Diva® SDK reports time stamps for signaling messages received in monitoring mode via the *DivaCallTimeStatistics* structure. The members define by name to which signaling message the time stamp belongs. If a member is set to zero, the corresponding message has not been received. The information is reported by the call property *DivaCPT_CallTimeStats*.

```
typedef struct
{
        DivaTime      SetupTime;
        DivaTime      CallProcTime;
        DivaTime      AlertTime;
        DivaTime      ConnectTime;
        DivaTime      ConnectAckTime;
        DivaTime      DisconnectTime;
        DivaTime      ReleaseTime;
        DivaTime      ReleaseCompTime;
} DivaCallTimeStatistics;
```

DivaRecordEndReasons

The *DivaRecordEndReasons* specifies the reason for the termination of the audio streaming.

```
typedef enum
{
        DivaRecordEndReasonUndefined = 0,
        DivaRecordEndReasonDisconnected = 1,
        DivaRecordEndReasonTimeout = 2,
        DivaRecordEndReasonSilence = 3,
        DivaRecordEndReasonMaxDTMF = 4,
        DivaRecordEndReasonTerminationDigit = 5,
        DivaRecordEndReasonInterDigitTimeout = 6,
        DivaRecordEndReasonIntialDigitTimeout = 7,
} DivaRecordEndReasons;
```

DivaRecordEndReasonUndefined

There is no specific reason for the termination of the recording. The application has terminated the recording.

DivaRecordEndReasonDisconnected

The recording terminates because the call has been disconnected.

DivaRecordEndReasonTimeout

The recording is terminated because the maximum time for recording to a file is reached. The time can be set in the call to DivaRecordVoiceFile or via DivaSetDTMFProcessingRules.

DivaRecordEndReasonSilence

The recording is terminated because the maximum silence time during recording to a file is reached. The option to terminate on silence is set by the call property *DivaCPT_VoiceRecordSilenceTimeout*.

DivaRecordEndReasonMaxDTMF

The recording is terminated because the maximum DTMF digits specified with a call to DivaSetDTMFProcessingRules have been received.

DivaRecordEndReasonTerminationDigit

The recording is terminated because one of the termination digits specified with a call to DivaSetDTMFProcessingRules has been received.

DivaRecordEndReasonInterDigitTimeout

The recording is terminated because the inter digit timeout specified with a call to DivaSetDTMFProcessingRules has been exceeded.

DivaRecordEndReasonIntialDigitTimeout

The recording is terminated because the initial digit timeout specified with a call to DivaSetDTMFProcessingRules has been exceeded.

DivaIdFormat

```
typedef enum
{
        DivaIdFormatDWORD,
        DivaIdFormatString,
        DivaIdFormatBinary
} DivaIdFormat;
```

DivaIdFormatDWORD

The identifier is given as a 32 bit binary value.

DivaIdFormatString

The identifier is given as a zero terminated string.

DivaIdFormatBinary

The identifier is given as a plain binary value. The first byte contains the length of the following binary data.

DivaIdDescriptor

```
typedef struct
{
        DWORD               IdFormat;
        unsigned char       Id[100];
} DivaIdDescriptor;
```

IdFormat

Specifies the format of the data in "Id". Valid options are defined by DivaIdFormat.

Id

Contains the identifier in the format defined by IdFormat.

DivaAPNotifyCallInParams

```
typedef struct
{
        DivaCallHandle              hdCall;
        DivaIdDescriptor            Identifier;
        DivaAPSendAudio             pfnSendAudio;
        DivaAPStopSendAudio         pfnStopSendAudio;
        DivaAPSetRecordFormat       pfnSetRecordAudio;
        DivaAPCloseAudio            pfnCloseAudio;
        DivaAPPassEvent             pfnPassEvent;
        DivaAPSetVolume             pfnSetVolume;
} DivaAPNotifyCallInParams;
```

hdCall

Identifies the channel / call at Dialogic® Diva® SDK level. The audio provider passes this handle with each call to the function entry points provided in *DivaAPNotifyCallInParams*.

Identifier

The *Identifier* is passed transparent through the Dialogic® Diva® SDK and contains information for the audio provider on the channel.

pfnSendAudio

Provides an entry point at the Diva SDK to be called by the audio provider to stream audio. If the assignment is only for inbound streaming, *pfnSendAudio* will be zero.

pfnStopSendAudio

Provides an entry point at the Diva SDK to be called by the audio provider to terminate the streaming of audio. If the assignment is only for inbound streaming, *pfnStopSendAudio* will be zero.

pfnSetRecordFormat

Provides an entry point at the Diva SDK to be called by the audio provider to set the format for received audio. The audio format is initially specified during notification of the call, but can be changed at any time.

pfnCloseAudio

Provides a function entry point at the Diva SDK. This function must be called by the audio provider when the instance is closed.

pfnPassEvent

This function is for future use and is set to zero.

pfnSetVolume

Provides an entry point at the Diva SDK to be called by the audio provider to set the volume for received and sent audio. For valid values, refer to DivaVolume. The volume can be changed at any time.

DivaAPNotifyCallOutParams

```
typedef struct
{
        AppCallHandle                   hAPCall;
        DivaAPNotifyCallClose           pfnNotifyCallClose;
        DivaAPNotifyReceiveAudio        pfnNotifyReceiveAudio;
        DivaAPConfirmAudioSend          pfnConfirmSend;
        DivaAPGetEventDetails           pfnGetEventDetails;
        DivaAudioFormat                 Format;
} DivaAPNotifyCallOutParams;
```

hAPCall

Identifies the channel / call at audio provider level. The Dialogic® Diva® SDK passes this handle with each call to the function entry points provided in *DivaAPNotifyCallOutParams*.

pfnNotifyCallClose

Provides an entry point at the audio provider. This function is called by the Diva SDK when the link between the audio provider and the Diva SDK for this call is disconnected. For more information, see APNotifyCallClose.

pfnNotifyReceiveAudio

Provides an entry point at the audio provider. This function is called by the Diva SDK when audio data is received. For more information, see APNotifyReceiveAudio.

pfnConfirmSend

Provides an entry point at the audio provider. This function is called by the Diva SDK to confirm that audio data passed by *DivaAPSendAudio* is sent and the buffer is free. For more information, see APConfirmAudioSend.

pfnGetEventDetails

This function entry is reserved for future use.

Format

The audio format to be used when *DivaAPNotifyReceiveAudio* is called. The format can be switched at any time using *DivaAPSetRecordFormat*.

DivaVolume

DivaVolume defines the range for setting the input and output volume. The volume is defined in the range from -18 to +18 db. The value for the unchanged volume is 0.

```
typedef enum
{
        DivaVolumeMin = -18,
        DivaVolumeNormal = 0,
        DivaVolumeMax = +18
} DivaVolume;
```

DivaVoicePosition

```
typedef struct
{
        DWORD                           Size;
        DivaVoicePositionFormat         Format;
        int                             Offset;
} DivaVoicePosition;
```

Size

Size defines the length of the data structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the version of the Dialogic® Diva® API, the size may grow.

Format

The parameter *Format* defines how the offset is interpreted. The offset can be given as byte offsets (*DivaVoicePositionFormatBytes*) or as time, in milliseconds (*DivaVoicePositionFormatMSec*).

Offset

Offset for positioning. Depending on the parameter format this is interpreted as bytes or as milliseconds. Note that the byte offset is depending on the audio formats (coding bits).

DivaDirection

```
typedef enum
{
        DivaDirectionInbound = 1,
        DivaDirectionOutbound = 2,
        DivaDirectionBoth = 3
} DivaDirection;
```

DivaDirectionInbound

The settings are only valid for inbound audio streaming.

DivaDirectionOutbound

The settings are only valid for outbound audio streaming.

DivaDirectionBoth

The settings are valid for inbound and outbound audio streaming.

DivaSignalService

```
typedef enum
{
        DivaSignalServiceSpeech = 1,
        DivaSignalServiceDigital = 2,
        DivaSignalServiceV110 = 8,
        DivaSignalServiceAudio3_1KHz = 4,
        DivaSignalServiceAudio7KHz = 5,
        DivaSignalServiceTelefony = 16,
        DivaSignalServiceFaxG3 = 17,
        DivaSignalServiceFaxG4 = 18,
        DivaSignalServiceVideo = 6
} DivaSignalServices;
```

The enum *DivaSignalService* defines the service to be signaled to the network for an outgoing call.

DivaSignalServiceSpeech

The call is signaled with the capabilities set to speech.

DivaSignalServiceDigital

The call is signaled as a pure digital call.

DivaSignalServiceV110

The call is signaled as a digital call using GSM services.

DivaSignalServiceAudio3_1KHz

The call is signaled as an audio call with 3.14 KHz.

DivaSignalServiceAudio7KHz

The call is signaled as an audio call with 7 KHz.

DivaSignalServiceTelefony

The call is signaled as ISDN telephone call.

DivaSignalServiceFaxG3

The call is signaled as analog call carrying fax G3 data.

DivaSignalServiceFaxG4

The call is signaled as digital call carrying fax G4 data.

DivaSignalServiceVideo

The call is signaled as digital call carrying video data.

DivaDeviceConfigType

DivaDeviceConfigType defines the available configuration parameter that can be read for a line device. The data types are defined in the union *DivaDeviceConfigValue*.

```
typedef enum
{
        DivaDCT_SwitchType,
        DivaDCT_PBXName,
        DivaDCT_DDIEnabled,
        DivaDCT_Layer2Mode,
        DivaDCT_NumberCollectLength,
        DivaDCT_AutoSpid,
        DivaDCT_SPID1,
        DivaDCT_SPID2,
        DivaDCT_DirectoryNumber1,
        DivaDCT_DirectoryNumber2
} DivaDeviceConfigType;
```

DivaDCT_SwitchType

Provides the switch type configured for the device. See DivaSwitchType for a list of values.

DivaDCT_PBXName

The symbolic name of the configured switch. Only available if the *DivaSwitchType* is set to *DivaSwitchTypeQ-Sig*. For all other switch types an empty string is returned. The data is given as zero terminated string.

DivaDCT_DDIEnabled

Specifies whether the direct inward dialing or direct dial in is enabled for the device. The data type is Boolean.

DivaDCT_Layer2Mode

Defines the information on the layer 2 mode. This information is needed to interpret the layer 2 and layer 1 status changes. The values are specified in *DivaLayer2Mode*.

DivaDCT_NTMode

Specifies if the line device is working as NT or TE. The data type is Boolean.

DivaDCT_NumberCollectLength

Provides the amount of digits that must be available as called number for an incoming call before the call is signaled to the application. The data type is DWORD.

DivaDCT_AutoSpid

Specifies if automatic SPID assignment is enabled. Only valid for US protocols. The data type is Boolean.

DivaDCT_SPID1

Specifies the SPID configured and assigned to the B-channel. Only valid for US protocols. The data type is a zero terminated string.

DivaDCT_SPID2

Specifies the SPID configured and assigned to the B-channel. Only valid for US protocols. The data type is a zero terminated string.

DivaDCT_DirectoryNumber1

Specifies the directory number configured and assigned to the B-channel. Only valid for US protocols. The data type is a zero terminated string.

DivaDCT_DirectoryNumber2

Specifies the directory number configured and assigned to the B-channel. Only valid for US protocols. The data type is a zero terminated string.

DivaDeviceConfigValue

The *DivaDeviceConfigValue* is used to read various device configuration options. The union covers all possible types and is defined in the Dialogic® Diva® SDK header files. For a description of the call properties, see DivaDeviceConfigType.

Applications may declare a variable of type *DivaDeviceConfigValue* and set the appropriate member corresponding to the used *DivaDeviceConfigType* or directly use a variable of the type corresponding to the property type, i.e. BOOLEAN. The Dialogic® Diva® API will internally check only for the needed size corresponding to the property type.

DivaDeviceStatusType

DivaDeviceStatusType defines the available status parameter that can be read for a line device. The table below provides information on the status parameter and the data type. The data types are defined in the union *DivaDeviceStatusValue*.

```
typedef enum
{
        DivaDST_Layer1Status,
        DivaDST_PotsLineStatus,
        DivaDST_Layer2Status,
        DivaDST_RedAlarm,
        DivaDST_BlueAlarm,
        DivaDST_YellowAlarm,
        DivaDST_ActiveConnections,
        DivaDST_TotalDSPs,
        DivaDST_UsedDSPs
        DivaDST_OutOfServiceDSPs,
        DivaDST_DSPStates
} DivaDeviceStatusType;
```

DivaDST_Layer1Status

Provides the information on the layer 1 status. Values are defined in *DivaLayer1Status*.

DivaDST_PotsLineStatus

The Dialogic® Diva® Analog Media Board provides the status information per line of the device. The application must set the *Channel* field of *DivaPotsLineState* before calling *DivaGetLineDeviceStatus*. The values are defined in *DivaPotsLineStatus*.

DivaDST_Layer2Status

Provides the information on the layer 2 status. Values are defined in *DivaLayer2Status*.

DivaDST_RedAlarm

If true the red alarm is active. Only valid for a Dialogic® Diva® PRI Media Board. The data type is Boolean.

DivaDST_BlueAlarm

If true the blue alarm is active. Only valid for a Diva PRI Media Board. The data type is Boolean.

DivaDST_YellowAlarm

If true the yellow alarm is active. Only valid for a Diva PRI Media Board. The data type is Boolean.

DivaDST_ActiveConnections

Number of currently active connections. The data type is DWORD.

DivaDST_TotalDSPs

Number of physical installed DSPs. The data type is DOWRD.

DivaDST_UsedDSPs

Number of DSPs that are currently attached to a B-channel. The data type is DWORD.

DivaDST_OutOfServiceDSPs

Number of DSPs that are currently out of service. The data type is DWORD.

DivaDST_DSPStates

Provides the state of each physical available DSP. The data type is DivaDSPStateArray.

DivaDeviceStatusValue

The *DivaDeviceStatusValue* is used to read various device status information. The union covers all types and is defined in the Dialogic® Diva® SDK header files. For a description of the call properties see DivaDeviceStatusType.

Applications may declare a variable of type *DivaDeviceStatusValue* and set the appropriate member corresponding to the used *DivaDeviceStatusType* or use directly a variable of the type corresponding to the property type, i.e. BOOLEAN. The Dialogic® Diva® API will internally check only for the needed size corresponding to the property type.

DivaSwitchType

```
typedef enum
{
        DivaSwitchType1TR6,
        DivaSwitchTypeETSI,
        DivaSwitchTypeFranceVN4,
        DivaSwitchTypeBelgium,
        DivaSwitchTypeSweden,
        DivaSwitchTypeNI_DMS,
        DivaSwitchType5ESSCustom,
        DivaSwitchTypeJapan,
        DivaSwitchTypeItaly,
        DivaSwitchTypeTaiwan,
        DivaSwitchTypeAustralia,
        DivaSwitchType4ESS_SDN,
        DivaSwitchType4ESS_SDS,
        DivaSwitchType4ESS_LDS,
        DivaSwitchType4ESS_MGC,
        DivaSwitchType4ESS_MGI,
        DivaSwitchTypeHongkong,
        DivaSwitchTypeRBSCAS,
        DivaSwitchTypeQSIG,
        DivaSwitchTypeNI_EWSD,
        DivaSwitchTypeNI_5ESS,
        DivaSwitchTypeQSIG_T1,
        DivaSwitchTypeTrunkE1,
        DivaSwitchTypeTrunkT1,
        DivaSwitchTypeR2CAS,
        DivaSwitchTypeFranceVN6,
        DivaSwitchTypePOTS
} DivaSwitchType;
```

DivaSwitchType specifies the options for the switch type read by the configuration property *DivaDCT_SwitchType*. For the description of configuration options, refer to the Dialogic® Diva® Configuration Manager Online Help file (DSMain.chm).

DivaLayer2Mode

```
typedef enum
{
        DivaLayer2ModePermanent,
        DivaLayer2ModeOnDemand,
        DivaLayer2ModeNoDisconnect
} DivaLayer2Mode;
```

DivaLayer2Mode specifies the options returned for the configuration property *DivaDCT_Layer2Mode*. For the description on configuration options refer to the Dialogic® Diva® Configuration Manager Online Help file (DSMain.chm).

DivaLayer1Status

```
typedef enum
{
        DivaLayer1Down = 0,
        DivaLayer1Up,
        DivaLayer1SyncLost,
        DivaLayer1Synchronized
} DivaLayer1Status;
```

DivaLayer1Status defines the values that can be returned by the status property *DivaDST_Layer1Status*.

DivaLayer1Down

Specifies that there is no layer 1 activity.

DivaLayer1Up

Specifies that the layer 1 is fully synchronized and operational.

DivaLayer1SyncLost

The synchronization at the layer 1 has been lost.

DivaLayer1Synchronized

The layer 1 has reached synchronization state.

DivaPotsLineStatus

```
typedef enum
{
        DivaPotsLineDown= 0,
        DivaPotsLineHookOff,
        DivaPotsLineIdle,
        DivaPotsLineRing,
        DivaPotsLinePolarityReverse
} DivaPotsLineStatus;
```

The *DivaPotsLineStatus* defines the values that can be returned by the status property *DivaDST_PotsLineStatus*.

DivaPotsLineDown

Specifies that there is no layer 1 activity.

DivaPotsLineHookOff

The line is in the hook off state.

DivaPotsLineIdle

TBD

DivaPotsLineRing

A ring is detected at the line.

DivaPotsLinePolarityReverse

Reverse polarity has been detected on the line.

DivaLayer2Status

```
typedef enum
{
        DivaLayer2Down = 0,
        DivaLayer2Up,
        DivaLayer2Closing,
        DivaLayer2Activating,
        DivaLayer2Initializing
} DivaLayer2Status;
```

DivaLayer2Status defines the values that can be returned by the status property *DivaDST_Layer2Status*.

DivaLayer2Down

The layer 2 is inactive.

DivaLayer2Up

The layer 2 is fully negotiated and operational. Layer 3 data packets can be exchanged.

DivaLayer2Closing

The layer 2 connection in disconnecting.

DivaLayer2Activating

The layer 2 is activated.

DivaLayer2Initializing

The layer 2 is initialized in order to establish the layer 2 connection.

DivaDSPState

```
typedef enum
{
        DivaDSPStateIdle,
        DivaDSPStateUsed,
        DivaDSPStateOutofService,
        DivaDSPStateUnavailable
} DivaDSPState;
```

DivaDSPState defines the status of a single DSP. Possible options are:

DivaDSPStateIdle

The DSP is operational and currently not attached to a data channel.

DivaDSPStateUsed

The DSP is attached to a data channel

DivaDSPStateOutofService

The DSP has reported problems and has been taken out of service.

DivaDSPStateUnavailable

The DSP is not populated or is not working.

DivaDSPStateArray

```
typedef struct
{
        DWORD                  Entires;
        DivaDSPState           State[100];
} DivaDSPStateArray;
```

DivaDSPStateArray defines the format of the DSP state data retrieved by the status property *DivaDST_DSPStates*.

Entries

The amount of DSP state entries in "State".

State

An array of type *DivaDSPState* containing the state for each DSP.

DivaDeviceStatusEvents

```
typedef enum
{
        DivaDSE_Layer1,
        DivaDSE_Layer2,
        DivaDSE_Alarms
} DivaDeviceStatusEvents;
```

DivaDeviceStatusEvents specify which events should be signaled to the applications event mechanism when a change occurs. These values are also used to specify the event class when the event *DivaEventDeviceStatusChanged* is signaled to the application.

DivaDSE_Layer1

The layer 1 state or the pots line state has changed.

DivaDSE_Layer2

The layer 2 state has changed.

DivaDSE_Alarms

The state of one of the alarms (red / blue / yellow) has changed.

DivaGenericToneFunction

```
typedef enum
{
        DivaGenericToneGetSupportedServices,
        DivaGenericToneEnableOperation,
        DivaGenericToneDisableOperation
} DivaGenericToneFunction;
```

DivaGenericToneGetSupportedServices

The function *DivaGenericToneGetSupportedServices* retrieves information on the capabilities of the selected line device.

DivaGenericToneEnableOperation

The function *DivaGenericToneEnableOperation* switches on a tone generation or detection or updates the parameter of a running tone operation.

DivaGenericToneDisableOperation

The function *DivaGenericToneDisableOperation* switches off a running tone generation or detection.

DivaSingleToneReport

DivaSingleToneReport defines which information for a detected single tone should be reported to the application in *DivaDetectorResults*.

```
typedef enum
{
        DivaSingleToneReportSignalNoiseRatio        = 0x00000002,
        DivaSingleToneReportEnergy                  = 0x00000004,
        DivaSingleToneReportFrequency               = 0x00000008,
        DivaSingleToneReportEnergyVariation         = 0x00000010,
        DivaSingleToneReportFrequencyVariation      = 0x00000020
} DivaSingleToneReport;
```

DivaDualToneReport

DivaDualToneReport defines which information for a detected dual tone should be reported to the application in *DivaDetectorResults*.

```
typedef enum
{
        DivaDualToneReportSignalNoiseRatio          = 0x00000002,
        DivaDualToneReportEnergyLowTone             = 0x00000004,
        DivaDualToneReportEnergyHighTone            = 0x00000008,
        DivaDualToneReportFrequencyLowTone          = 0x00000010,
        DivaDualToneReportFrequencyHighTone         = 0x00000020
} DivaDualToneReport;
```

DivaGenericToneResultType

DivaGenericToneResultType specifies if the information relates to a single or dual tone.

```
typedef enum
{
        DivaSingleToneStart = 1,
        DivaDualToneStart,
        DivaSingleToneStop,
        DivaDualToneStop
} DivaGenericToneResultType;
```

DivaGenericToneResult

DivaGenericToneResult specifies the result of a generic tone detection.

```
typedef enum
{
        DivaGenericToneDoneSuccess = 0,
        DivaGenericToneDoneParamsShrinked,
        DivaGenericToneDoneParamsIgnored,
        DivaGenericToneErrorInvalidParams = 0x80,
        DivaGenericToneErrorOutOfResource,
        DivaGenericToneErrorMissingParams
}DivaGenericToneResult;
```

Result codes named DivaGenericToneError… (bit 7 is set) indicate that the operation has not been started. For all others, the operation has started but parameters may be ignored or shrunk.

DivaToneDetectorResults

```
typedef struct
{
        DivaGenericToneResultType     Type;
        DWORD                         TimeStamp;
        DivaGenericToneResult         Result;
        union
        {
        struct
        {
        short                         SignalNoiseRatio;
        short                         Energy;
        WORD                          Frequency;
        WORD                          AmplitudeVariation;
        WORD                          FrequencyVariation;
        }Single
        struct
        {
        short                         SignalNoiseRatio;
        short                         EnergyToneLow;
        short                         EnergyToneHigh;
        WORD                          FrequencyToneLow;
        WORD                          FrequencyToneHigh;
        }Dual;
        }Tone;
} DivaToneDetectorResults;
```

DivaGenericToneInfo

DivaGenericToneInfo provides the confirmation or indication for a low level generic tone request.

```
typedef struct
{
        BOOL                          Confirmation;
        Void                          *Handle;
        DivaGenericToneFunction       Function;
        DWORD                         DataLength;
        BYTE                          Data[1];
} DivaGenericToneInfo;
```

Confirmation

If the parameter *Confirmation* is set, the following data are a confirmation to a previous request. If the parameter is not set, the following data must be interpreted as an indication.

Handle

The parameter *Handle* is only valid if the parameter *Confirmation* is set. The value of the handle is the same as the value passed to *DivaSendGenericToneRequest*.

Function

The parameter *Function* specifies the function that has been requested. The parameter is only valid if the parameter *Confirmation* is set.

DataLength

The parameter *DataLength* specifies the amount of bytes written to the Data array.

Data

The parameter *Data* contains the confirmation or indication information. The size depends on the function and the request. For more information, see CxTone.pdf.

DivaActiveDiscReasons

DivaActiveDiscReasons specifies the reasons for an active disconnection.

```
typedef enum
{
        DivaActiveDiscReasonBusy = 1,
        DivaActiveDiscReasonReject,
        DivaActiveDiscReasonNoAnswer,
        DivaActiveDiscReasonNumberUnknown,
        DivaActiveDiscReasonInvalidNumber,
        DivaActiveDiscReasonNumberChanged,
        DivaActiveDiscReasonUnallocatedNumber,
        DivaActiveDiscReasonOutOfOrder,
} DivaActiveDiscReasons;
```

DivaSMSProtocol

DivaSMSProtocol specifies the SMS protocol to be used for sending and receiving Short Messages.

```
typedef enum
{
        DivaSMSProtocolOne,
        DivaSMSProtocolTwo,
} DivaSMSProtocol;
```

DivaSMSProtocolOne

Protocol 1 is more widely used for SM transmission. This protocol is supported by the Dialogic® Diva® API.

DivaSMSProtocolTwo

Protocol Two is not presently supported by the Diva API.

DivaMessageStatus

```
typedef enum
{
        DivaMessageAdded,
        DivaMessageRemoved,
        DivaMessageUnknown,
} DivaMessageStatus;
```

DivaMessageNumberInfo

```
typedef enum
{
        DivaMessageNumberUnknown,
        DivaMessageNumberNotAvailable,
} DivaMessageNumberInfo;
```

DivaMessageInvokeMode

```
typedef enum
{
        DivaMessageInvokeDeferred,
        DivaMessageInvokeImmediate,
        DivaMessageInvokeCombined,
        DivaMessageInvokeSupress,
} DivaMessageInvokeMode;
```

DivaMessageInvokeMode specifies how the message activation should be invoked. Options are deferred or combined. If set to *DivaMessageInvokeSupress*, the invocation mode is suppressed.

DivaMWIActivateParams

```
typedef struct
{
    DWORD                   Size;
    DWORD                   Handle;
    DWORD                   Service;
    DWORD                   NumMessages;
    DivaMessageStatus       Status;
    DWORD                   Reference;
    DivaMessageInvokeMode   InvokeMode;
    char                    ReceivingUserNumber[MAX_ADDR_LEN];
    char                    ControllingUserNumber[MAX_ADDR_LEN];
    char                    ControllingUserProvidedNumber[MAX_ADDR_LEN];
    DivaTime                Time;
} DivaMWIActivateParams;
```

Size

The *Size* defines the length of the structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the Dialogic® Diva® API version, the size may grow.

Handle

This parameter is not interpreted by the Dialogic® Diva® SDK. It is passed with the event *DivaEventMWICompleted* when the operation is finished. The application may use this handle to assign the result delivered with the event to a request.

Service

Specifies the service that should be signaled to the switch. This identifies the media type of the message, e.g., voice or fax. For IVR systems that signal voice messages, this value must be set to 1.

NumMessages

Specifies the amount of messages that should be signaled.

Status

Specifies the status; for options see DivaMessageStatus.

Reference

The parameter is only valid if *Status* is not set to *DivaMessageUnknown*.

InvokeMode

Specifies the invocation mode. For valid options, see DivaMessageInvokeMode.

ReceivingUserNumber

The extension of the user to whom the messages should be signaled.

ControllingUserNumber

This parameter depends on the used switch. Some switches use this number to authenticate the requester. This must be set in accordance with the switch configuration.

ControllingUserProvidedNumber

This parameter is switch-dependent and should be set to an empty string by default.

DivaMWIDeactivateParams

```
typedef struct
{
        DWORD                       Size;
        DWORD                       Handle;
        DWORD                       Service;
        DivaMessageInvokeMode       InvokeMode;
        char                        ReceivingUserNumber[MAX_ADDR_LEN];
        char                        ControllingUserNumber[MAX_ADDR_LEN];
        char                        ControllingUserProvidedNumber[MAX_ADDR_LEN];
        DivaTime                    Time;
```

DivaResultAnsweringMachineDetector

```
typedef enum
{
        DivaResultUserTerminated,
        DivaResultHumanTalker,
        DivaResultAnsweringMachine,
        DivaResultAnsweringMachineTone,
        DivaResultSilence,
        DivaResultFaxOrModem,
} DivaResultAnsweringMachineDetector;
```

DivaResultUserTerminated

The application has terminated the answering machine detector with a call to *DivaDisableAnsweringMachineDetectior*.

DivaResultHumanTalker

The remote side is a human talker.

DivaResultAnsweringMachine

The remote side is an answering machine.

DivaResultSilence

No signal received within a given timeout. The detector terminated due to the initial silence timeout.

DivaResultFaxMachine

The remote end is a fax machine.

DivaResultModem

The remote end is a modem.

DivaTerminationDigits

The *DiveTerminationDigits* are a mask to specify which digits will be used to terminate an ongoing operation or signal a special event.

```
#define    DivaTerminationDigit_None      0x000000000
#define    DivaTerminationDigit_0         0x000000001
#define    DivaTerminationDigit_1         0x000000002
#define    DivaTerminationDigit_2         0x000000004
#define    DivaTerminationDigit_3         0x000000008
#define    DivaTerminationDigit_4         0x000000010
#define    DivaTerminationDigit_5         0x000000020
#define    DivaTerminationDigit_6         0x000000040
#define    DivaTerminationDigit_7         0x000000080
#define    DivaTerminationDigit_8         0x000000100
#define    DivaTerminationDigit_9         0x000000200
#define    DivaTerminationDigit_A         0x000000400
#define    DivaTerminationDigit_B         0x000000800
#define    DivaTerminationDigit_C         0x000001000
#define    DivaTerminationDigit_D         0x000002000
#define    DivaTerminationDigit_S         0x000004000
#define    DivaTerminationDigit_H         0x000008000
#define    DivaTerminationDigit_FAXCNG    0x000010000
#define    DivaTerminationDigit_FAXCED    0x000020000
```

DivaProcessingGroup

```
typedef enum
{
        DivaProcessingGroupEvent = 1,
        DivaProcessingGroupSending,
        DivaProcessingGroupRecording,
} DivaProcessingGroup;
```

DivaProcessingGroupEvent

This option defines that the processing parameters are used to create the events *DivaEventDTMFMaxDigit*, *DivaEventDTMFTerminationDigit*, and *DivaEventDTMFInterDigitTimeout*.

DivaProcessingGroupSending

This option defines that the processing parameters have influence on the termination of the outbound streaming operation initiated by *DivaSendVoice…* functions.

DivaProcessingGroupRecording

This option defines that the processing parameters have influence on the termination of the inbound recording operation initiated by *DivaRecordVoiceFile*.

DivaSendVoiceEndReasons

```
typedef enum
{
        DivaSendVoiceEndReasonUndefined = 0,
        DivaSendVoiceEndReasonDisconnected,
        DivaSendVoiceEndReasonMaxDTMF,
        DivaSendVoiceEndReasonTerminationDigit,
        DivaSendVoiceEndReasonInterDigitTimeout,
        DivaSendVoiceEndReasonInitialDigitTimeout,
        DivaSendVoiceEndReasonMaxTimeout
} DivaSendVoiceEndReasons;
```

DivaSendVoiceEndReasonUndefined

There is no specific reason for the termination. Note that user requested termination is reported by a separate event and is not listed here.

DivaSendVoiceEndReasonDisconnected

The sending terminates because the call has been disconnected.

DivaSendVoiceEndReasonMaxDTMF

The sending terminates because the maximum amount of DTMF digits has been received. The maximum amount of digits has been specified by the application with a call to *DivaSetDTMFTerminationRules*.

DivaSendVoiceEndReasonTerminationDigit

The sending terminates because one of the termination digits has been received. The termination digits have been specified by the application with a call to *DivaSetDTMFTerminationRules*.

DivaSendVoiceEndReasonInterDigitTimeout

The sending terminates because the inter digit timeout has exceeded. The inter digit timeout has been specified by the application with a call to *DivaSetDTMFTerminationRules*.

DivaSendVoiceEndReasonInitialDigitTimeout

The sending terminates because the initial digit timeout has exceeded. The initial digit timeout has been specified by the application with a call to *DivaSetDTMFTerminationRules*.

DivaSendVoiceEndReasonMaxTimeout

The sending terminates because the maximum timeout has exceeded. The maximum timeout has been specified by the application with a call to *DivaSetDTMFTerminationRules*.

DivaSysConfCallDirection

```
typedef enum
{
        DivaSysConfCallDirectionNone,
        DivaSysConfCallDirectionOutbound,
        DivaSysConfCallDirectionBoth,
} DivaSysConfCallDirection;
```

DivaSysConfCallDirectionNone

Specifies that no calls can be made on this interface.

DivaSysConfCallDirectionOutbound

Specifies that only outgoing calls, but no incoming calls, can be made on this interface.

DivaSysConfCallDirectionBoth

Specifies that inbound and outbound calls can be done on this interface.

DivaSysConfType

```
typedef enum
{
        DivaSCT_RASInstalled,
        DivaSCT_RASTotalChannels,
        DivaSCT_RASCallDirection,
        DivaSCT_TAPIInstalled,
        DivaSCT_TAPITotalChannels,
        DivaSCT_TAPICallDirection,
        DivaSCT_CAPIInstalled,
        DivaSCT_CAPITotalChannels,
        DivaSCT_CAPICallDirection,
        DivaSCT_DataModemInstalled,
        DivaSCT_DataModemTotalChannels,
        DivaSCT_DataModemCallDirection,
        DivaSCT_FaxModemInstalled,
        DivaSCT_FaxModemTotalChannels,
        DivaSCT_FaxModemCallDirection,
} DivaSysConfType;
```

DivaSysConfValue

```
typedef union
{
    BOOL                        RASInstalled;
    DWORD                       RASTotalChannels;
    DivaSysConfCallDirection    RASCallDirection;
    BOOL                        TAPIInstalled;
    DWORD                       TAPITotalChannels;
    DivaSysConfCallDirection    TAPICallDirection;
    BOOL                        CAPIInstalled;
    DWORD                       CAPITotalChannels;
    DivaSysConfCallDirection    CAPICallDirection;
    BOOL                        DataModemInstalled;
    DWORD                       DataModemTotalChannels;
    DivaSysConfCallDirection    DataModemCallDirection;
    BOOL                        FaxModemInstalled;
    DWORD                       FaxModemChannels;
    DivaSysConfCallDirection    FaxModemDirection;
} DivaSysConfValue;
```

DivaDeviceCapabilities

```
typedef enum
{
        DivaDevCapsPSTNBased,
        DivaDevCapsIPBased,
        DivaDevCapsAnalogBased,
        DivaDevCapsExtEquipment,
} DivaDeviceCapabilities;
```

DivaDevCapsPSTNBased

If the function *DivaCheckDeviceCapabilities* returns true for DivaDevCapsPSTNBased, the line device is PSTN-based. This includes ISDN and analog-based devices.

DivaDevCapsIPBased

If the function *DivaCheckDeviceCapabilities* returns true for DivaDevCapsIPBased, the line device is IP-based. This includes H.323 and SIP devices.

DivaDevCapsAnalogBased

If the function *DivaCheckDeviceCapabilities* returns true for DivaDevCapsAnalogBased, the line device is PSTN and analog-based. This is only true for the Dialogic® Diva® Analog Media Board, not for RBS lines.

DivaDevCapsExtEquipment

If the function *DivaCheckDeviceCapabilities* returns true for DivaDevCapsExtEquipment, the line device has an external equipment, e.g., it can connect a headset and microphones.

DivaTraceLevel

```
typedef enum
{
        DivaTraceLevelNothing = 0,
        DivaTraceLevelError,
        DivaTraceLevelWarning,
        DivaTraceLevelInformation,
        DivaTraceLevelApiEntry,
        DivaTraceLevelDebug,
        DivaTraceLevelDebugHigh,
        DivaTraceLevelDataMsg,
} DivaTraceLevel;
```

DivaTraceLevelNothing

No trace information is written.

DivaTraceLevelError

Error information is written.

DivaTraceLevelWarning

Warning information is written.

DivaTraceLevelInformation

Additional information is written.

DivaTraceLevelApiEntry

The entry and return from interface functions are written.

DivaTraceLevelDebug

Developer trace information is written

DivaTraceLevelDebugHigh

High level debug information is written.

DivaTraceLevelDataMsg

Data messages are written. This creates a very large amount of trace output and may have an impact on the performance of the system.

DivaChannelStatus

```
typedef enum
{
        DivaChannelStatusUnblocked = 0,
        DivaChannelStatusActiveBlocked = 1,
        DivaChannelStatusPassiveBlocked = 2,
        DviaChannelStatusInUse = 3
} DivaChannelStatus;
```

DivaChannelStatusUnblocked

The channel is unblocked and can be used for communication.

DivaChannelStatusActiveBlocked

The channel is blocked by the application and cannot be used for communication.

DivaChannelStatusPassiveBlocked

The channel is blocked by the switch and cannot be used for communication.

DivaChannelStatusInUse

The channel is currently in use for a call or not yet available for making or answering another call.

DivaDataOptions

DivaDataOptions specifies attributes for a data frame.

```
typedef enum
{
        DivaDataOptionQualifier = 0x00000001,
        DivaDataOptionMoreData  = 0x00000002,
        DivaDataOptionDelivery  = 0x00000004

} DivaDataOptions;
```

DivaDataOptionQualifier

The option "qualifier" is set via this option. The availability of this option depends on the used protocol.

DivaDataOptionMoreData

The option "more data follows" is set via this option. This allows the application to do fragmentation of frames. The availability of this option depends on the used protocol.

DivaDataOptionDelivery

The option "Delivery confirmation" is set via this option. The availability of this option depends on the used protocol.

DivaFaxScanLineMax

The maximum scan line provides information about the scan line capabilities of a receiving fax.

```
typedef enum
{
        DivaFaxScanLineUnknown = 0,
        DivaFaxScanLineMax215,
        DivaFaxScanLineMax255,
        DivaFaxScanLineMax303
} DivaFaxScanLineMax;
```

DivaFaxScanLineUnknown

The remote side did not provide the scan line capabilities.

DivaFaxScanLineMax215

The remote fax is able to handle scan lines of 215 millimeter. This corresponds to the ISO A4 format.

DivaFaxScanLineMax255

The remote fax is able to handle scan lines of 255 millimeter. This corresponds to the ISO B4 format.

DivaFaxScanLineMax303

The remote fax is able to handle scan lines of 303 millimeter. This corresponds to the ISO A3 format.

DivaFaxStoreModes

With *DivaFaxStoreModes*, the rules for storing received fax pages in files is selected.

```
typedef enum
{
        DivaFaxStorePerSession = 0,
        DivaFaxStorePerDocument,
        DivaFaxStorePerPage,
} DivaFaxStoreModes;
```

DivaFaxStorePerSession

All pages of the complete fax session are stored in one file. This is the default setting.

DivaFaxStorePerDocument

This option is reserved for future use.

DivaFaxStorePerPage

Each page of a fax reception is stored in a separate file. This option is currently only available for color fax JPEG documents. The file names get the addition "_D1_Px," where "x" is the page index.