Dialogic

Dialogic® Diva® API

Developer's Reference Guide

Part of the Dialogic® Diva® Software Development Kit

March 2015 206-444-12

www.dialogic.com

Copyright and Legal Notice

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 SOFTWARE LICENSE AGREEMENT ("AGREEMENT") ON THIS SEALED PACKAGE BEFORE OPENING THE PACKAGE. BY OPENING THE PACKAGE, YOU ACCEPT THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE WITH OR ARE UNWILLING TO ACCEPT THESE TERMS AND CONDITIONS, YOU MAY RETURN THE PACKAGE IN UNOPENED "AS NEW" CONDITION (INCLUDING ALL DOCUMENTATION AND BINDERS OR OTHER CONTAINERS) FOR A FULL REFUND. BY DOWNLOADING, INSTALLING, COPYING OR OTHERWISE USING THE ENCLOSED SOFTWARE ("PROGRAM"), YOU FURTHER AGREE AND ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT AND UNDERSTAND IT, AND THAT BY TAKING ANY ONE OR MORE OF SUCH STEPS/ACTIONS YOU AGREE TO BE BOUND BY SUCH TERMS AND CONDITIONS. DIALOGIC IS UNWILLING TO LICENSE THE SOFTWARE TO YOU IF YOU DO NOT ACCEPT AND AGREE TO BE BOUND BY THE TERMS AND CONDITIONS OF THIS AGREEMENT.

Intellectual Property

The enclosed Software ("Program") and all accompanying documentation are individually and collectively owned by Dialogic Corporation ("Dialogic"), its subsidiaries and/or its suppliers and are protected by all applicable intellectual property laws and international treaty provisions. Therefore, You and Your Authorized Users must treat the Program and documentation like any other material so protected, except as expressly permitted in this Agreement. In particular, but without limitation, You acknowledge that the Program and its accompanying documentation constitute valuable intellectual property rights, including without limitation trade secrets and copyrights, and confidential information of Dialogic. The Program and all programs developed thereunder and all copies thereof (including without limitation translations, compilations, partial copies with modifications and updated works) are proprietary to Dialogic and title to all applicable copyrights, trade secrets, patents and other intellectual property rights therein remains in Dialogic, its subsidiaries, and/or its suppliers. Except as expressly permitted in this Agreement, You shall not sell, transfer, publish, disclose, display or otherwise make available the Program or copies thereof to others. You agree to secure and protect the Program, its accompanying documentation and copies thereof in a manner consistent with the maintenance of Dialogic's rights therein and to take appropriate action by instruction or agreement with Your employees and/or consultants who are permitted access to the Program to satisfy Your obligations hereunder. Violation of any provision of this paragraph shall be the basis for immediate termination of this Agreement. Because unauthorized use or transfer of the Software or documentation may diminish substantially the value of such materials and irrevocably harm Dialogic, if You breach the provisions of this Section of this Agreement, Dialogic shall be entitled to injunctive and/or other equitable relief, in addition to other remedies afforded by law, to prevent a breach of this Section of this Agreement.

Grant of License

Subject to the terms and conditions of this Agreement Dialogic grants to You a non-exclusive, personal, non-transferable license to use the Program in object code form only and solely in accordance with the following terms and conditions:

- You may make, install and use only one (1) 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 You may also make one copy solely for backup or archive purposes;
- 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 distribute the Program in object code only and only as part of, or integrated by You into, a computer system that (i) contains a Dialogic hardware product, (ii) includes a substantial amount of other software and/or hardware manufactured or marketed by You and (iii) is marketed and sublicensed to an end user for the end user's own internal use in the regular course of business (a "Licensed System");
- Each end user to whom a Licensed System is distributed must agree to license terms with respect to the Program that are at least as protective of Dialogic's rights in the Program as those set forth in this Agreement;
- You shall receive one (1) Program master disk, and shall be solely responsible for copying the Program into the Licensed Systems and for warranting the physical media on which it is copied
- You may make one (1) copy of the documentation accompanying the Program, 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 shall not remove, and each copy of the Program shall contain, the same copyright, proprietary, patent and/or other applicable intellectual property or other ownership notices, plus any restricted rights legends that appear in the Program and/or this Agreement and, if You copy the Program onto media to which a label may be attached, You shall attach a label to the media that includes all such notices and legends that appear on the Program master disk and envelope;
- You may not rent or lease the Program. You may not reverse engineer, decompile or disassemble the Program. Except as is strictly necessary for You to integrate the Program with other software and/or hardware to produce the Licensed Systems, You shall not copy, modify or reproduce the Program or documentation in any way. You shall use Your best efforts to ensure that any user of the Program does not reverse engineer, decompile or disassemble the Program to derive a source code equivalent of the Program;
- 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 Agreement, this license is immediately and automatically terminated;
- The Program may be used only in conjunction with Dialogic hardware;
- The Program shall not be exported or re-exported in violation of any export provisions of the United States or any other applicable jurisdiction.

Upgrades

If the Program is provided as an upgrade and the upgrade is an upgrade from another 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 Agreement does not grant You additional license(s). If You and Your Authorized Users choose to upgrade this Program or the product used together with the Program and such upgrade requires the license of additional software (whether a charge is associated with such software or not), the license agreement associated with such additional software shall govern the license of such additional software to the exclusion of this Agreement.

Term

The Agreement is effective until terminated. You may terminate it at any time by notifying Dialogic and/or by destroying the Program and all accompanying documentation together with all copies, modifications and merged portions in any form. The Agreement will also terminate automatically upon the occurrence or lack of occurrence of certain terms and/or conditions set forth in this Agreement, or if You fail to comply with any term or condition of this Agreement. You agree that upon any such termination You shall destroy or return to Dialogic the Program and all accompanying documentation supplied by Dialogic, together with any and all copies, modifications and merged portions in any form. All provisions of this Agreement relating to disclaimers of warranties, limitation of liability, remedies, or damages, and licensor's proprietary rights shall survive termination.

Limited Warranty

Dialogic solely warrants the media on which the Program is furnished to You to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of purchase by You as evidenced by a copy of Your receipt. If such a defect appears within the warranty period, You may return the defective media to Dialogic for replacement without charge provided Dialogic, in good faith, determines that it was defective in materials or workmanship. Replacement is Your sole remedy with respect to such a defect. 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.

Disclaimers, Limitations of Liability and Customer Remedies

Except as set forth in the "Limited Warranty" Section of this Agreement, the Program and accompanying documentation are provided to You "as is." Neither Dialogic, its subsidiaries, its suppliers, nor its licensor(s) (if any) warrants that the Program will meet Your requirements or that its use will be uninterrupted or error-free. Except as set forth in the "Limited Warranty" Section, EACH OF DIALOGIC, ITS SUBSIDIARIES, ITS SUPPLIERS AND ITS LICENSOR(S) (IF ANY) DISCLAIMS ANY AND ALL REPRESENTATIONS AND WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE PROGRAM AND ACCOMPANYING DOCUMENTATION, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR AGAINST LATENT DEFECTS. Except as set forth in the "Limited Warranty" Section, neither Dialogic, its subsidiaries, its suppliers, nor its licensor(s) (if any) shall have any liability to You or any third party for any claim, loss or damage of any kind, including but not limited to lost business profits, business interruption, loss of information, or other pecuniary loss and indirect, punitive, incidental, economic, consequential or special damages, arising out of or in connection with this Agreement and/or the use, inability to use the Program and/or the Program's performance or inability to perform nor from or in connection with the Program's accompanying documentation, or any data or equipment related thereto or used in connection therewith. In no event shall Dialogic's, its subsidiaries', its suppliers' or its licensor(s)'s liability for damages, whether arising out of contract, negligence, warranty, or patent or copyright infringement, exceed the fees You paid for the Program. No representation or warranty regarding the Program may be made without Dialogic's, its subsidiaries', its suppliers', or its licensor(s)'s (if any) prior written consent, and any warranty or representation made by You or Your customers regarding the Program shall not constitute an obligation of Dialogic, its subsidiaries, its suppliers, or other licensor(s) (if any). This limited warranty gives You specific legal rights. You may have other rights, which may vary from jurisdiction to jurisdiction. Also, as some jurisdictions do not allow the exclusion or limitation for certain damages, some of the above limitations may not apply to You.

Right to Audit

If this Program is licensed for use in a Company, Your Company and You individually and collectively agree to keep all usual and proper records and books of accounts and all usual proper entries relating to each installation 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 licensed channels or 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 Program and/or referred to in this Agreement is provided "as is" with no warranty of any kind.

Miscellaneous

You acknowledge that You have read this Agreement, that You understand it, and that You agree to be bound by its terms and conditions, and You further agree that this is the complete and exclusive statement of the Agreement between the Dialogic and You ("the Parties"), which supersedes and merges all prior proposals, understandings and all other agreements, oral and written, between the Parties relating to the Program. You agree to indemnify and hold harmless Dialogic and its subsidiaries, affiliates, suppliers, officers, directors and employees from and against any claim, injury, loss or expense, including reasonable attorneys' fees, arising out of (i) Your failure to comply with the provisions of this Agreement, or (ii) any other wrongful conduct by or on behalf of You. This Agreement applies to all updates, future releases, modifications and portions of the Program contained in or merged into other programs. This Agreement may not be modified or altered except by written instrument duly executed by Dialogic. No action, regardless of form, arising out of this Agreement or the use of the Program may be brought by You more than two (2) years after the cause of action has first arisen. Except as provided herein, neither this Agreement nor any rights granted are assignable or transferable, and any assignment or transfer will be null and void. If You authorize any other person to copy the Program, You shall obligate that person in writing to comply with all conditions of this Agreement. Dialogic shall have the right to collect from You its reasonable expenses incurred in enforcing this agreement, including attorney's fees. The waiver or failure of Dialogic to exercise in any respect any right provided for herein shall not be deemed a waiver of any further right hereunder. All rights and remedies, whether conferred hereunder or by any other instrument or law, will be cumulative and may be exercised singularly or concurrently. Failure by either Dialogic or You to enforce any term or condition of the Agreement will not be deemed a waiver of future enforcement of that or any other term or conditions. The terms and conditions stated herein are declared to be severable. Should any term(s) or condition(s) of this Agreement be held to be invalid or unenforceable the validity, construction and enforceability of the remaining terms and conditions of this Agreement shall not be affected. It is expressly agreed that Dialogic and You are acting as independent contractors under this Agreement. These terms and conditions will prevail notwithstanding any different, conflicting or additional terms and conditions that may appear on any other agreement between Dialogic and You. Deviations from these terms and conditions are not valid unless agreed to in writing in advance by an authorized representative of Dialogic. Any notices sent to Dialogic under this Agreement must be sent by registered mail or courier to the attention of Dialogic's legal department at the address below or such other address as may be listed on www.dialogic.com from time to time as being Dialogic's Montreal headquarters.

U.S. Government Restricted Rights

The Program and all accompanying 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)(iii) 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, both as applicable.

Governing Law

Any and all claims arising under this Agreement shall be construed and controlled by the laws in force in the Province of Quebec, Canada, excluding its principles of conflict of laws and the United Nations Convention on Contracts for the Sale of Goods. Dialogic is not obligated under any other agreements unless they are in writing and signed by an authorized representative of Dialogic.

Contents

About This Publication

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, the Dialogic® Diva® Component API, and the Diva API for .NET.
Dialogic® Diva® API Overview	Introduction to the components provided with the 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 Call Properties	Description of all call properties 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 1

Dialogic® Diva® SDK Overview

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

The Diva SDK releases are 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.

Notes:

- Previous Diva SDK versions supported the Dialogic® Host Media Processing (HMP) software ; however support for it has been discontinued under this SDK at the time of this publication.
- The Diva SDK allows the use Diva® SoftIP Software resources; however this SW has been discontinued at the time of this publication and is no longer supported in this version of the SDK. References to retired product or IP SDK functions are left in the document for the sake of completness only.

The Diva SDK includes the following components:

- Libraries providing functions to access the Dialogic® communication platforms
- Programming samples in source code
- Documentation explaining the 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, or 4.xx
- Installed threading library (pthread)

Installation for RPM-based systems

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

The Diva SDK is available for the GNU compiler versions 2.xx and 3.xx and 4.xx. 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 are 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

File(s)	Description
/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

Installation for DEB-based systems

For Debian-based distributions, the Dialogic® Diva® SDK is provided as a DEB package that contains the documentation with the binaries, header files, and the samples. Use a package tool or the command line version of dpkg to install the Diva SDK.

To install use: dpkg -i dssdk-<version>-1.i386.rpm

The Diva SDK is available for the GNU compiler versions 2.xx and 3.xx and 4.xx (only 64-bit). 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/dssdk/CxDtmf.pdf	Documentation about proprietary DTMF extensions to CAPI interface
/usr/share/doc/dssdk/CxEcho.pdf	Documentation about proprietary echo cancelling extensions to CAPI interface
/usr/share/doc/dssdk/CxFax.pdf	Documentation about proprietary FAX extensions to CAPI interface
/usr/share/doc/dssdk/CxModem.pdf	Documentation about proprietary Modem extensions to CAPI interface
/usr/share/doc/dssdk/CxTone.pdf	Documentation about proprietary tone generation and recognition extensions to CAPI interface
/usr/share/doc/dssdk/DivaSAPI.pdf	The main Diva SDK documentation
/usr/share/doc/dssdk/readme.html	Documentation about installation of Diva SDK
/usr/share/doc/dssdk/examples.tgz	Archive that contains the samples

Installing the samples

If you unpack the file "examples.tgz" into the current directory, one directory for each sample is created. Some of 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
audiomonitorcodecs	Monitoring calls and recording audio in uncompressed and compressed formats
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
faxoutcustomheadline	Derived from faxoutsimple, adding a fax headline in Utf-8 characer coding on the fly

Directory	Sample Description
faxserver	Simple faxserver
smsservicecenter	SMS service center
showdevicestatistics	Sample for printing the statistics of a device
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
POS_modem_scenarios	Modem scenarios for POS application
SpecificModemConnection	Specify the modulation type to use for the modem connection

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.

www.dialogic.com/products/tdm_boards/development_tools/default.htmThe Diva SDK is freely distributed with Dialogic® communication platforms. You do not have to purchase licences for developing applications based on the software development kit.

Dialogic® Diva® SDK application programming interfaces

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

- 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 (only for Diva Media Boards): It provides Dialogic-specific CAPI extensions that are fully CAPI 2.0 compliant.

Dialogic® communication platforms provide call control, media streaming, and management functionality that are available on the Diva API.

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 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.

Dialogic® communication platform-related information

The Diva SDK uses the Dialogic® Diva® System Release software to communicate to the TDM or IP-based communication resources if available. The Diva-based software is automatically started at system start and configured via the Dialogic® Diva® Configuration Manager.

Some software features are based on licenses, and there are various options that can be combined. Based on the available licenses, Diva API interface functions may return *DivaErrorNotSupported* if a requested function is not licensed or no more licenses are available.

CHAPTER 3

Dialogic® Diva® API Overview

The 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 Diva Media Boards software.

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.

Requirements for installation

- Installed and running Dialogic® communication platform
- Installed CAPI
- Installed GNU C/C++ compiler version 2.xx, 3.xx, or 4.xx
- Installed threading library (pthread)

Installation for RPM-based systems

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`

The Diva SDK is available for the GNU compiler versions 2.xx and 3.xx and 4.xx (only 64-bit). 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 are 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

File(s)	Description
/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

Installation for DEB-based systems

For Debian-based distributions, the Dialogic® Diva® SDK is provided as a DEB package that contains the documentation with the binaries, header files, and the samples. Use a package tool or the command line version of dpkg to install the Diva SDK.

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

The Diva SDK is available for the GNU compiler versions 2.xx and 3.xx and 4.xx (only 64-bit). 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/dssdk/CxDtmf.pdf	Documentation about proprietary DTMF extensions to CAPI interface
/usr/share/doc/dssdk/CxEcho.pdf	Documentation about proprietary echo cancelling extensions to CAPI interface
/usr/share/doc/dssdk/CxFax.pdf	Documentation about proprietary FAX extensions to CAPI interface
/usr/share/doc/dssdk/CxModem.pdf	Documentation about proprietary Modem extensions to CAPI interface
/usr/share/doc/dssdk/CxTone.pdf	Documentation about proprietary tone generation and recognition extensions to CAPI interface
/usr/share/doc/dssdk/DivaSAPI.pdf	The main Diva SDK documentation
/usr/share/doc/dssdk/readme.html	Documentation about installation of Diva SDK
/usr/share/doc/dssdk/examples.tgz	Archive that contains the samples

Installing the samples

If you unpack the file "examples.tgz" into the current directory, one directory for each sample is created. Several 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
audiomonitorcodecs	Monitoring calls and recording audio in uncompressed and compressed formats
audiomonitorex	Interactively monitoring calls and record audio streams
faxdial	Sample for processing outgoing fax calls
faxinsimple	Mainstream sample for fax reception

Directory	Sample Description
faxoutsimple	Mainstream sample for sending fax
faxserver	Simple faxserver
faxoutcustomheadline	Sends a fax with Utf-8 character coding fax headline
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.

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

How applications use ports or channels

The communication channels provided by the installed 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 Diva API does not reserve resources and does not block resources against access by other applications like serial interfaces do.

Access specific line devices and channels

By default, the Diva SDK selects line devices and channels for outgoing call on demand. Applications may want to place a call on a specific line device or even a specific channel or timeslot. To select a specific line device, the application specifies the line device in the call to *DivaConnect*. Applications using *DivaCreateCall* and *DivaDial* use the call property *DivaCPT_LineDevice* to specify the line device to be used. Applications that want to set the data channel or timeslot use the call property *DivaCPT_DataChannel*. Call properties for outgoing calls can only be used with *DivaCreateCall* and *DivaDial*. Note that the property *DivaCPT_DataChannel* uses logical channels and numbers channels from 1 to the amount of data channels. If the underlying protocol has a signaling channel between data channels, e.g., E1 protocol, the Diva SDK handles this. The property *DivaCPT_DataChannel* is also used to select a line of a Dialogic® Diva® Analog Media Board. To retrieve the information on which channel an incoming call is signaled, the application can also use the property *DivaCPT_DataChannel*. In addition, the member *AssignedBChannel* of *DivaCallInfo* provides the real timeslot information.

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:
  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. The sample is realized as a simple command line program.
- audiomonitorcodecs - Monitoring calls and recording audio in uncompressed and compressed formats:
  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 and to store the recorded audio in any format supported by the Dialogic® Diva® Media Boards. The sample 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 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.
- faxoutcustomheadline - Outgoing fax application with custom headline:
  This command-line sample application shows how to apply a TrueType or OpenType font file for the purpose of inserting a headline text in Utf-8 character coding. The fax data must be available as a file in TIFF Class F format, and a user-supplied (.TTF) font file must be supplied.
- 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.

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 Diva API header files are available for the application developer. The following groups of functions are available:

- Registration
- Set common parameters for all calls
- Connection management, Connect / Disconnect / Get and Set 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 (only Diva Media Boards)
- ASR / TTS Integration

Instances

Each application of the 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 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 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 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 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 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	Dialogic® Media Boards
ISO A4 at:	R8 x 3.85 R8 x 7.7 R8 x 15.4 200 x 200	1728	x
ISO B4 at:	R8 x 3.85 R8 x 7.7 R8 x 15.4 200 x 200	2048	x
ISO A3 at:	R8 x 3.85 R8 x 7.7 R8 x 15.4 200 x 200	2432	x
ISO A4 at:	300 x 300	2592	x
ISO B4 at:	300 x 300	3072	x
ISO A3 at:	300 x 300	3648	x
ISO A4 at:	R16 x 15.4 400 x 400	3456	x
ISO B4 at:	R16 x 15.4 400 x 400	4096	x
ISO A3 at:	R16 x 15.4 400 x 400	4864	x
ISO A4 at:	600 x 600	5184	x
ISO B4 at:	600 x 600	6144	x
ISO A3 at:	600 x 600	7296	x
ISO A4 at:	1200x 1200	10368	x
ISO B4 at:	1200 x 1200	12288	x
ISO A3 at:	1200 x 1200	14592	x

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.

Customizing the fax headline

New fax Call Properties allow applying a TrueType or OpenType font file for the purpose of customizing the fax headline text; it allows the text to be in either Utf-8 or Utf-16 little endian character coding; default is 8-bit ASCII extension. In addition it provides the ability of setting the size of the headline in points; omitting the date, time, and page information in the headline; and selecting the font face if multiple fonts are present in the font file.

The use of a user-supplied font file (.TTF) must be enabled in the Call Properties for the feature to take effect, otherwise the normal headline method is used instead.

Voice processing

The 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 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 Diva API supports the feature rich conference sessions provided by the Dialogic® communication platform.

Conference setup and management

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 Diva API implements conferences handled by Diva Media Boards.

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 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:

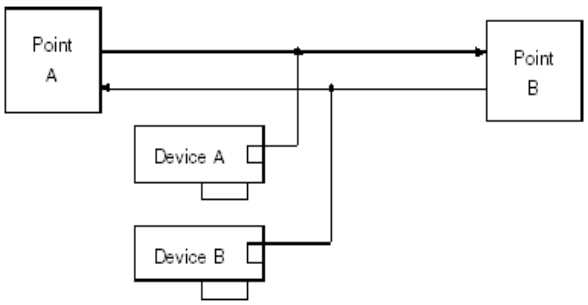
- Create Line Interconnect between two existing calls.
- Receive data on one or both RX-channels optionally.
- Get mixed data streams of both RX-channels on one call object optionally.
- Have data streaming on one or both lines optionally.

Monitoring and tapping

In general, 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.

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 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

The monitoring of signaling information is available for lines running E1 or T1 Q.931-based protocols and for E1 R2-based lines. For Q.931-based lines, called and calling party numbers are extracted from the setup and info messages. For E1 R2 based lines, line state related signaling information is exchanged from channel 16. Depending on the selected R2 variant when the monitoring is started, the called and calling party number is retrieved from the inband information before the call is indicated to the application.

Monitoring of the signaling channel can be done at different layers depending on the application requirements. The 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*.

For Q.931-based lines, 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 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.

Media Resource Control Protocol for Speech Recognizer

A common interface for communication to the speech recognizer and synthesizer is the Media Resource Control Protocol (MRCP). MRCP is used to control the commutation between an application and the recognizer / synthesizer, the audio signal is exchanged via RTP. The Diva SDK supports speech recognizer via MRCPv1 and implements an MRCP client that handles the MRCP protocol. The RTP stream negotiated via MRCP is controlled by the Divas SDK to stream the audio signal to the speech recognizer. All streaming and recording operation as well as DTMF and tone detection can be done in parallel to the speech recognition. For details refer to the chapter "Speech Recognizer Support".

The configuration of the parameter required for speech recognizer access can be done via Diva SDK interface functions or via the configuration file dssdl.xml.

Below is a simple sample of a configuration to communicate to a speech recognizer and the description of the parameter.

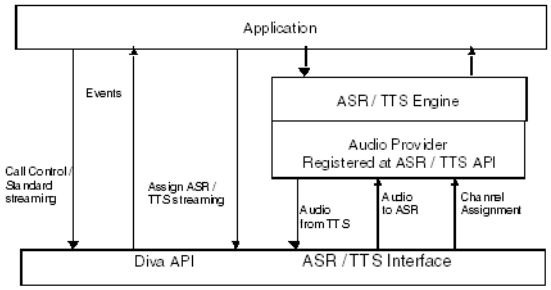
```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE DivaAPIConfiguration>
<DivaAPIConfiguration>

  <SpeechRecognizer Name="MyRecognizer">
    <MrcpVersion>V1</MrcpVersion>
    <ServerName>Speech</ServerName>
    <ServerIPAddress>192.168.0.100</ServerIPAddress>
    <ServerPort>4900</ServerPort>
    <LocalIPAddress>192.168.0.50</LocalIPAddress>

    <Parameter>
        <Name>Confidence-Threshold</Name>
        <Value>80</Value>
    </Parameter>

    <Grammar>
        <ContentType>text/uri-list</ContentType>
        <RequestId></RequestId>
       <Content> builtin:grammar/digits </Content>
    </Grammar>

  </SpeechRecognizer>

</DivaAPIConfiguration>
```

MrcpVersion	Specifies the version to be used, options are V1 and V2, the default is V1. Currently only V1 is supported.
ServerName	Specifies the name of the server that is running the recognizer. The parameter is optional if a *ServerIpAddress* is specified. If no name is provided, the *ServerIpAddress* will be used.
ServerIPAddress	Specifies the IP address of the server running the recognizer. The parameter is optional if a *ServerName* is provided.
ServerPort	Port number the recognizer is listening for requests. The parameter is optional the default for MRCPv1 is 4900.
LocalIPAddress	Specifies the local IP address to be used for communication with the server running the recognizer. The parameter is optional if not set the default network interface is used.

Parameter	Multiple parameters may be specified. Parameters consist of a name and a value. The parameter will be passed unchanged to the recognizer before recognition starts. The application may at runtime add or overwrite parameters on a per call base.
Grammar	The grammar definition contains the type, how the grammar is specified, and the grammar content itself. The type may be text/uri-list as in the sample above or application/grammar+xml. If the content is formatted in XML, the XML control characters must be escaped or the data must be included in a CDATA section. The grammar may be overwritten by the application on per call base at runtime.

Audio provider (ASR/TTS) interface

Generally, a voice application using the 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 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*.

The application can control the service state of a Dialogic® Diva® Media Board. This is used to take boards out of service for maintenance and avoid that the switch signals calls to these devices. The service state of a line device can be controlled via the functions DivaSetServiceState. To busy-out a line of a Dialogic® Diva® Analog Media Board, use *DivaSetAnalogHookState*.

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 Diva API offers common functions for all call types as well as functions for specific call types, e.g., for fax and voice. This chapter contains the following sections:

Startup and version

To ensure that the resources for the 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.

The Diva API provides the following startup and version functions:

- DivaInitialize
- DivaTerminate
- DivaGetVersion
- DivaGetVersionEx

DivaInitialize

The *DivaInitialize* function initializes the internal core of the 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 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 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 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 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.

The following functions are provided in this section:

- DivaGetNumLineDevices
- DivaGetLineDeviceInfo
- DivaCheckDeviceCapabilities
- DivaRegister
- DivaUnregister
- DivaSetLineDeviceParamsFax
- DivaSetLineDeviceParamsVoice
- DivaGetLineDeviceConfiguration
- DivaGetLineDeviceStatus
- DivaSetLineDeviceStatusEvents
- DivaGetLineDeviceStatistics
- DivaClearLineDeviceStatistics
- DivaEnableExtensions
- DivaDisableExtensions
- DivaGetDeviceName
- DivaDeviceMgmtGetValue
- DivaDeviceMgmtSetValue
- DivaDeviceMgmtExecute
- DivaGetChannelStatus
- DivaSetChannelStatus
- DivaGetSystemConfiguration
- DivaSystemConfigurationActive
- DivaSetAnalogHookState

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 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 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.
DivaEventModeOSEvent	Event object to be signaled	NULL
DivaEventModeMsgLoop	Windows® handle	Message type

See also

DivaUnregister

DivaUnregister

The *DivaUnregister* function releases all allocated resources at the 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 the 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 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 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 DivaDeviceStatisticsType 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

DivaDeviceStatisticsType, DivaEventDeviceStatusChanged, DivaRegister

DivaGetLineDeviceStatistics[1]

DivaGetLineDeviceStatistics queries the layer 1 statistics for the line device.

```
DWORD   DivaGetLineDeviceStatistics (   DWORD                    LineDeviceId,
                                        DivaDeviceStatisticsType Type,
                                        DivaDeviceStatisticsValue* pValue,
                                        DWORD                     ValueSize);
```

Parameters

LineDeviceId

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

Type

[in] TThis parameter specifies which configuration parameter should be read. Valid enumeration types are defined in *DivaDeviceStatisticsType.*

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

1. While this function is part of the common SDK, Layer 1 statistics functionality is not necessarily available in Windows

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

Remarks

The function queries the line device for layer 1 (if applicable) and internal CPU statistics. Refer to for more information. The function is a synchronous function and returns right away.

Also note that the LoadxxPercent values are only filled in for the first device Id of a physical board..

See also

DivaClearLineDeviceStatistics, DivaLayer1Statistics, DivaDeviceStatisticsType

DivaClearLineDeviceStatistics

DivaClearLineDeviceStatistics clears the line device statistics for the line device.

```
DWORD   DivaGetLineDeviceStatistics (   DWORD                   LineDeviceId,
                                        DivaDeviceStatisticsType Type);
```

Parameters

LineDeviceId

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

Type

[in] TThis parameter specifies which configuration parameter should be read. Valid enumeration types are defined in *DivaDeviceStatisticsType.*

Return values

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

Remarks

The function clears (resets) the values of the line device layer 1 (if applicable) and internal CPU statistics. Refer to for more information. The function is a synchronous function and returns right away.

See also

DivaGetLineDeviceStatistics, DivaDeviceStatisticsType

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". 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 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 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 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 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, 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

DivaSetServiceState

DivaSetServiceState sets the specified line device in service or out of service.

```
DWORD   DivaSetServiceState (   DWORD       LineDevice,
                                BOOL        bEnable );
```

Parameters

LineDevice

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

bEnable

[in] The *bEnable* parameter specifies if the device should be placed in service or out of service. If set to true, the device is placed in service.

Return values

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

Remarks

The function changes the service state of a line device. The function is only supported on line devices based on Diva Media Boards. If a line device is disabled, the layer 1 is signaled as down to the remote peer. This can be used for maintenance reasons to indicate to a switch or PBX that no more calls should be signaled to this device.

See also

No references.

Connection-oriented functions

This chapter contains the following connection-oriented functions:

- DivaCreateCall
- DivaDial
- DivaListen
- DivaProceeding
- DivaAlert
- DivaAttachToCall
- DivaAnswer
- DivaAnswerFax
- DivaAnswerVoice
- DivaAnswerVoIP (RTP)
- DivaAnswerModem
- DivaAnswerSMS
- DivaReject
- DivaConnect
- DivaConnectFax
- DivaConnectVoice
- DivaConnectVoIP
- DivaConnectModem
- DivaConnectSMS
- DivaSetCallType
- DivaSetCallTypeFax
- DivaSetCallTypeVoice
- DivaSetCallTypeVoIP
- DivaDisconnect
- DivaGetCallInfo
- DivaCloseCall

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 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 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 called number can be a single number or a list of numbers separated by semicolons. One range can be specified by the LOW: and HIGH: keywords, e.g., LOW:100;High:200. 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).

The Diva API supports up to 10 single numbers or one range.

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 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. A progress indicator can be added to the proceeding message via the call property *DivaCPT_FacilityDataArray*.

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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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

DivaEnableDataChannel

DivaEnableDataChannel changes the status of the data channel in manual data channel mode.

```
DWORD    DivaEnableDataChannel (    DivaCallHandle    hdCall,
                                    BOOL              bEnable );
```

Parameters

hdCall

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

bEnable

[in] The *bEnable* parameter specifies if the data channel should be enabled or disabled. If set to true, the data channel is enabled.

Return values

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

Remarks

The function initiates the establishment or disconnect of the data channel. This function is only valid if the application has enabled the manual data channel mode via the call property *DivaCPT_ManualDataChannel*. If the manual data channel mode is not enabled, the function returns *DivaErrorInvalidState*.

Once the new state of the data channel is reached, the event *DivaEventDataChannelStatus* is signaled. The event provides the new status in the second parameter.

See also

No references.

Data transfer functions

This chapter contains the following data transfer functions:

- DivaSendData
- DivaReceiveData
- DivaSendFrame
- DivaReceiveFrame

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 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 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 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 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 the following 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 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 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 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 *DivaErrorOpenFile*, *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

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 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 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 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 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

DivaValidateFaxFile

DivaValidateFaxFile validates the format of a given fax document.

```
DWORD   DivaValidateFaxFile ( const char*                pFilename,
                              BOOL*                      pRequiresExtensions,
                              DivaFaxDocumentProperties* pProperties );
```

Parameter

pFilename

[in] The *pFilename* parameter points to the filename of the file to be validated. The process context of the caller must have read access rights for this file.

pRequiresExtension

[out] The *pRequiresExtension* parameter points to a memory location of type BOOL. If the given file requires extended fax capabilities from the peer the value TRUE will be written to this location.

pProperties

[out] The *pProperties* parameter points to a memory location of type DivaFaxDocumentProperties that receives details about the fax document. The parameter is optional and may be set to NULL is no document properties are needed.

Return values

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

Remarks

The function validates the format of the given file and checks for valid SFF and TIFF formats. If the format is supported the function returns *DivaSuccess*, otherwise the function returns *DivaErrorUnsupportedFormat*.

Some fax formats may requires special capabilities that are no guaranteed by all fax endpoints. If this is the case the function sets the *RequiresExtension* flag.

If the application provides a reference to a *DivaFaxDocumentProperties* variable, the page count and details about the document resolution are returned, if the given file format allows this.

See Also

DivaSendFax, DivaAppendFax, DivaSendMultipleFaxFiles, DivaAppendFaxFiles

Voice transfer functions

This chapter contains the following 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 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.

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 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 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 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 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 *DivaEventSendVoiceCanceled*.

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 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 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 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 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 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 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 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 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.

DivaSetSamplingRate

DivaSetSampling sets the sampling rate for play or recording.

```
DWORD    DivaSetSamplingRate (  DivaCallHandle    hdCall,
                                DWORD             SamplingRate,
                                DivaDirection     Direction );
```

Parameters

hdCall

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

SamplingRate

[in] The *SampingRate* parameter specifies the new sampling rate to be used. The default sampling rate is 8000. The valid range for the sampling rate is given by the *DivaSamplingRate*. See Remarks below.

Direction

[in] The *Direction* parameter specifies if the sampling rate should be set for inbound, outbound, or both directions. For valid values, refer to DivaDirection.

Return values

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

Remarks

The function sets the sampling at the given direction. If the underlying Dialogic® communication platform does not support different sampling rates, *DivaErrorNotSupported* is returned.

If the function is called before the data channel is available, the sampling rate is stored and activated when the data channel is established.

The default sampling rate is 8000. The minimum supported sampling rate is 1250 and the maximum sampling rate is 51200 as defined in *DivaSamplingRate*.

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 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.

DivaEnableTransactionRecording

DivaEnableTransactionRecording enables or disables the transaction recording.

```
DWORD    DivaEnableTransactionRecording (    DivaCallHandle    hdCall,
                                             BOOL              bEnable );
```

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*.

bEnable

[in] The parameter *bEnable* specifies if transaction recording 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

With the function *DivaEnableTransactionRecording*, the application controls if the recorded audio signal should also contain the sent audio signal. If this function is enabled, the Diva Media Boards will mix send and received audio into one audio stream.

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 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 *DivaErrorInvalidFunction*, *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, *DivaErrorInvalidHandle*, *DivaErrorUnsupportedFormat*, and *DivaErrorOpenFile*.

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 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 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 (    char*                   pFilename,
                                     DivaAudioFormat         Format,
                                     DivaVoicePositionFormat PositionFormat,
                                     DWORD*                  pLength );
```

Parameters

pFilename

[in] The *pFilename* parameter points to the file name 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.

PositionFormat

[in] The *PositionFormat* parameter specifies if the length should be returned in bytes or as duration. For supported options, see DivaVoicePositionFormat.

pLength

[out] 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    DivaSetVoiceFileLength (    char*                   pFilename,
                                     DivaAudioFormat         Format,
                                     DivaVoicePositionFormat PositionFormat,
                                     DWORD                   Length );
```

Parameters

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.

PositionFormat

[in] The *PositionFormat* parameter specifies if the length is given in bytes or as duration. For supported options, see DivaVoicePositionFormat.

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 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 *DivaErrorInvalidFunction*, *DivaErrorInvalidParameter*, *DivaErrorInvalidState*, *DivaErrorInvalidHandle*, *DivaErrorUnsupportedFormat*, and *DivaErrorOpenFile*.

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

DivaEnableNoiseSuppression

DivaEnableNoiseSuppression enables or disables the noise suppression.

```
DWORD   DivaEnableNoiseSuppression ( DivaCallHandle      hCall,
                                     BOOL                bEnable );
```

Parameter

hdCall

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

bEnable

[in] The parameter *bEnable* specifies if the noise suppression is enabled or disabled.

Return values

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

Remarks

The function enables or disables the noise suppression for a connected call. An application may also enable noise suppression via the call property *DivaCPT_EnableNoiceSuppression* before answering or dialing a call. The state of the noise suppression can be retrieved by the call property *DivaCPT_NoiseSuppressionActive.* Noise suppression is only supported on Diva Media boards with DSPs.

See Also

DivaEnableEchoCanceller

DTMF, tone, and AMD 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:

- DivaReportDTMF
- DivaSendDTMF
- DivaReportTones
- DivaSendTone
- DivaSendContinuousTone
- DivaStopContinuousTone
- DivaGenerateSingleTone
- DivaGenerateDualTone
- DivaStopToneGeneration
- DivaDetectSingleTone
- DivaDetectDualTone
- DivaGetToneDetectorResult
- DivaSendGenericToneRequest
- DivaGetGenericToneInfo
- DivaSpecifyCustomTone
- DivaSetDTMFProcessingRules
- DivaGetDTMFBuffer
- DivaClearDTMFBuffer
- DivaEnableAnsweringMachineDetector
- DivaDisableAnsweringMachineDetector
- DivaDetectFSKData
- DivaStopDetectFSKData

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 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 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 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 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 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 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 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 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 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 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 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 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 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 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.

DivaSpecifyCustomTone

Via *DivaSpecifyCustomTone*, the application specifies a specific event to be signaled if a custom tone or sequence is detected.

```
DWORD    DivaSpecifyCustomTone (    DivaCallHandle          hdCall,
                                    DWORD                   Type,
                                    DWORD                   Recurrences,
                                    DWORD                   NumDefinitions,
                                    DivaToneDefinition*     pDefinitions );
```

Parameter

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*.

Type

[in] The parameter *Type* identifies how the tone should be interpreted. Valid options are DivaRingingTone and DivaBusyTone.

Recurrences

[in] The parameter *Recurrences* specifies how often the tone and pause sequence specified by *pDefinitions* must be repeated to trigger the event specified by *Type*.

NumDefinitions

[in] The parameter *NumDefinitions* specifies the number of members in the definition table pointed to by *pDefinitions*.

pDefinitions

[in] The parameter *pDefinitions* points to an array containing elements of type *DivaToneDefinition*. These elements describe the frequencies, duration, pause, and variations of the tone.

Return values

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

Remarks

Via *DivaSpecifyCustomTone*, the application may specify one or more custom tones and cadences to be indicated via a single event to the application. Each tone definition specified via a *DivaToneDefinition* contains the specification about the tone, namely, whether the tone is single or dual, what the duration of the tone is, and optionally what the tone pause is. Multiple definitions can be used to define the cadence to be detected. If the cadence should occur multiple times before this is indicated to the application, the parameter *Recurrences* can be used. The occurrence of the tone or cadence is indicated by the event *DivaEventCustomToneDetected* and the *Type* parameter is signaled with the event.

See also

DivaEventCustomToneDetected

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 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 separately for each group. 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 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 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

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 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 than the specified time, it will be interpreted as an answering machine.

MaxInterSpeakerTimeout

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

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 *DivaEnableAnsweringMachineDetector*, the application enables the analysis 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 answering machine has answered.

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

If the detector detects the remote peer as an answering machine, the termination event with the result *DivaResultAnsweringMachine* is signaled. At this time, the announcement of the answering machine may still be streamed.

Note: The answering machine detector requires detection capabilities not currently 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, DivaResultAnsweringMachineDetector

DivaDisableAnsweringMachineDetector

DivaDisableAnsweringMachineDetector stops the answering machine detector.

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

Parameters

hdCall

[in] The *hdCall* identifies the call at the 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

DivaDetectFSKData

DivaDetectFSKData switches the detection of FSK data on.

```
DWORD    DivaDetectFSKData (    DivaCallHandle        hdCall,
                                int                   Sensitivity,
                                DivaFSKModulation     Modulation,
                                DWORD                 Options );
```

Parameter

hdCall

The parameter *hdCall* is the Diva SDK call handle provided with the event *DivaEventIncomingCall* or returned by *DivaCreateCall*.

Sensitivity

This parameter specifies the sensitivity. 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.

Modulation

This parameter specifies the modulation to be used for FSK detection. For valid options, refer to DivaFSKModulation.

Options

The *Options* parameter specifies how FSK data is provided to the application. Valid options are defined by FSKDetectorOptions; see Remarks for more information.

Returns

The function returns *DivaSuccess* (0) if the underlying line device supports detection of FSK data. Other possible errors are *DivaErrorInvalidHandle*, *DivaErrorNotSupported*, and *DivaErrorInvalidParameter*.

Remarks

The function starts the detector for FSK data. The function is only valid if the call is established with the call type *DivaCallTypeVoice*. By default, the Diva SDK filters the FSK data on a frame base by validating length fields and the checksum. If the application requires the complete received FSK data, the option *DivaFSKOptionTransparent* can be set. In that case, the application gets all data including the preamble bytes.

A sample call to enable detection of V.23 modulated data would be:

```
Result = DivaDetectFSKData ( hCall,
                             -46,
                             DivaFSKModulationV23,
                             DivaFSKOptionDefault );
```

See also

DivaStopDetectFSKData

DivaStopDetectFSKData

DivaStopDetectFSKData switches a previously enabled FSK detector off.

```
DWORD    DivaStopDetectFSKData (   DivaCallHandle          hdCall );
```

Parameter

hdCall

The parameter *hdCall* is the Diva SDK call handle provided with the event *DivaEventIncomingCall* or returned by *DivaCreateCall*.

Returns

The function returns *DivaSuccess* (0) if the underlying line device supports detection of FSK data. Other possible errors are *DivaErrorInvalidHandle*, *DivaErrorNotSupported*, and *DivaErrorInvalidState*.

Remarks

The function stops a previously enabled FSK detector.

See also

DivaDetectFSKData

Speech Recognizer Support

The Dialogic® Diva® SDK provides build in access to speech recognizer via the Media Resource Control Protocol (MRCP) version 1. The communication with the Speech Recognizer is done by the Diva SDK, the application continues to use the Diva SDK interface. The Diva SDK streams the audio signal via RTP to the speech recognizer and controls the recognition process via MRCPv1. The application may use any streaming and recording function in parallel to the speech recognition.

The Diva SDK will provides high level functions to setup speech recognizer sessions, to start recognition and to retrieve recognition results. Setting up the parameter to access a speech recognizer and configure the session can be done by the application calling Diva SDK functions or via a configuration file. The Diva SDK implements a high abstraction level comparable to enabling and processing DTMF tones. As a minimum requirement the application would call *DivaOpenSpeechRecognizer* when a call is connected and *DivaStartSpeechRecognition* when recognition should start. The result would be reported via event and the application would retrieve the result via *DivaGetSpeechRecognizerResult*. Optionally the Diva SDK allows to set detailed configuration parameter e.g. for grammar via *DivaSetSpeechRecognizerParameter* or *DivaSetSpeechRecognizerGrammar*..

In the simplest scenario the parameter for accessing the speech engine are configured via the configuration file and the application just needs to perform the following steps:

1. Open a recognizer session via *DivaOpenSpeechRecognizer* when the call is initiated, answered or connected.
2. Start recognition via *DivaStartSpeechRecognizer*

3. Process the event *DivaEventSpeechRecognizerProgress*
4. Retrieve recognizer results via *DivaGetSpeechRecognizerResult*
5. Continue with 2) for next recognition on the same call if needed
6. Stop the recognizer session via DivaStopSpeechRecognizer (will be done implicit when the call is closed).

The following functions are available for speech recognizer support:

- DivaInitializeSpeechProcessing
- DivaOpenSpeechRecognizer
- DivaCloseSpeechRecognizer
- DivaStartSpeechRecognizer
- DivaStopSpeechRecognizer
- *DivaGetSpeechRecognizerResult*
- DivaGetSpeechRecognizerResultDetails
- *DivaSetSpeechRecognizerParameter*
- *DivaSetSpeechRecognizerGrammar*
- *DivaCreateSpeechRecognizer*
- DivaSetSpeechRecognizerDefaultParameter
- *DivaSetSpeechRecognizerDefaultGrammar*

DivaInitializeSpeechProcessing

DivaInitializeSpeechProcessing initializes the speech processing for recognizer and synthesizer.

```
DWORD   DivaInitializeSpeechProcessing ( const char*         LocalIpAddress );
```

Parameter

LocalIpAddress

[in] The *LocalIpAddress* parameter allows the application to specify the local IP address to be used. The parameter is optional and may be set to zero, see remarks.

Return values

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

Remarks

The function initializes the speech processing and must be called before any other speech recognizer or speech synthesizer related function is called. The application may specify a local IP address, if not specified the default network address is used. Different local IP addresses may also be specified per speech server.

See Also

DivaCreateSpeechRecognizer

DivaOpenSpeechRecognizer

DivaOpenSpeechRecognizer establishes a session with a speech recognizer.

```
DWORD   DivaOpenSpeechRecognizer ( DivaCallHandle          hdCall,
                                   const char*             RecognizerName );
```

Parameter

hdCall

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

RecognizerName

[in] The *RecognizerName* parameter specifies the recognizer to be used. The recognizer is identified by a symbolic name, which is specified in the configuration file or via *DivaCreateSpeechRecognizer*. The parameter is optional, if not specified the first configured speech recognizer is used.

Return values

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

Remarks

The function initiates a session with the specified speech recognizer. The status of the session is reported asynchronously via the event *DivaEventSpeechRecognizerStatus*. Note that only one recognizer can be open for a DivaCallHandle. If a recognizer session is already open the function will return *DivaErrorInvalidState.*

Multiple speech recognizers may be available in a system. The recognizers are identified by a symbolic name and the application may select which recognizer should be opened by the parameter *RecognizerName.* If only one speech recognizer is available the parameter can be set to NULL or an empty string.

See Also

DivaEventSpeechRecognizerStatus, *DivaStartSpeechRecognizer*, *DivaCloseSpeechRecognizer*

DivaCloseSpeechRecognizer

DivaCloseSpeechRecognizer terminates a session with a speech recognizer.

```
DWORD   DivaCloseSpeechRecognizer ( DivaCallHandle          hdCall );
```

Parameter

hdCall

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

Return values

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

Remarks

The function terminates a session with the specified speech recognizer. The status of the session is reported asynchronously via the event *DivaEventSpeechRecognizerStatus*.

See Also

DivaEventSpeechRecognizerStatus, *DivaStartSpeechRecognizer*, *DivaCloseSpeechRecognizer*

DivaStartSpeechRecognizer

DivaStartSpeechRecognizer starts the recognition process.

```
DWORD   DivaStartSpeechRecognizer ( DivaCallHandle          hdCall,
                                    BOOL                    bAutoRestart );
```

Parameter

hdCall

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

bAutoRestart

[in] The *bAutoRestart* parameter specifies that the recognition process is automatically restarted after a match or no-match event. The parameter is reserved for future use.

Return values

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

Remarks

The function starts the recognition process. The progress of the recognition is reported asynchronously via the event *DivaEventSpeechRecognizerProgress*. For progress options refer to *DivaSpeechRecognizerProgress*. The application may stop processing before the recognition completes by calling *DivaStopSpeechRecognizer*.

See Also

DivaEventSpeechRecognizerProgress, *DivaStopSpeechRecognizer*, *DivaOpenSpeechRecognizer*, *DivaCloseSpeechRecognizer*

DivaStopSpeechRecognizer

DivaStopSpeechRecognizer starts the recognition process.

```
DWORD   DivaStopSpeechRecognizer ( DivaCallHandle          hdCall );
```

Parameter

hdCall

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

Return values

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

Remarks

The function stops the recognition process. The stop is confirmed via the event *DivaEventSpeechRecognizerProgress*. Note that the speech recognizer resource remains open and attached to the call.

See Also

DivaEventSpeechRecognizerProgress, DivaStartSpeechRecognizer, DivaOpenSpeechRecognizer, DivaCloseSpeechRecognizer, DivaGetSpeechRecognizerResult

DivaGetSpeechRecognizerResult

DivaGetSpeechRecognizerResult retrieves the recognition result.

```
DWORD   DivaGetSpeechRecognizerResult ( DivaCallHandle              hdCall,
                                        char*                       pTextBuffer,
                                        DWORD*                      pTextSize,
                                        DWORD*                      pConfidence );
```

Parameter

hdCall

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

pTextBuffer

[out] The *pTextBuffer* parameter is a reference to a buffer that receives the text detected by the speech recognizer.

pTextSize

[in / out] The *pTextSize* parameter is a reference to a DWORD value that contains the size of the buffer referenced by *pTextBuffer* when the function is called and returned the amount of bytes written or needed on return. Refer to remarks for details.

pConfidence

[out] The *pConfidence* parameter is a reference to a DWORD value that receives the confidence level of the recognition. The parameter is optional, if the application does not need the confidence level the parameter may be set to NULL.

Return values

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

Remarks

The function retrieves the results of the speech recognition. The application passes a buffer to retrieve the detected text. The length of the buffer is passed as reference value. If the buffer is too small to receive the text the function returns DivaErrorInsufficientBuffer and the required size is places into the location specified by *pTextSize*. The application may call *DivaGetSpeechRecognizerResult* with the *pTextBuffer* parameter set to NULL to retrieve the needed size. If no recognized data is pending the function returns *DivaErrorNoDataAvailable.*

The recognition results may contain addition information like type of detected data, e.g. speech or digit, and the grammar that has been used. Detailed information is returned by the function *DivaGetSpeechRecognizerResultDetails*.

See Also

DivaEventSpeechRecognizerProgress,DivaOpenSpeechRecognizer,DivaCloseSpeechRecognizer,DivaStartSpeechRecognizer,DivaStopSpeechRecognizer,DivaGetSpeechRecognizerResultDetails

DivaGetSpeechRecognizerResultDetails

DivaGetSpeechRecognizerResultDetails retrieves detailed information about the recognition result.

```
DWORD   DivaGetSpeechRecognizerResultDetails ( DivaCallHandle                  hdCall,
                                               DivaSpeechRecognizerResultType  Type,
                                               char*                           pBuffer,
                                               DWORD*                          pSize );
```

Parameter

hdCall

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

Type

[out] The *Type* parameter identifies the type of results to retrieve. For available types refer to the remarks section.

pBuffer

[out] The *pBuffer* parameter is a reference to a buffer that receives the information specified by the *Type* parameter..

pSize

[in / out] The *pSize* parameter is a reference to a DWORD value that contains the size of the buffer referenced by *pBuffer* when the function is called and returned the amount of bytes written or needed on return. Refer to remarks for details.

Return values

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

Remarks

The function retrieves the detailed results of the speech recognition. The application passes a buffer to retrieve the textual information. The length of the buffer is passed as reference value. If the buffer is too small to receive the text the function returns DivaErrorInsufficientBuffer and the required size is places into the location specified by *pSize*. The application may call *DivaGetSpeechRecognizerResultDetailst* with the *pBuffer* parameter set to NULL to retrieve the needed size. If no recognized data is pending the function returns *DivaErrorNoDataAvailable.*

The returned information depends on the requested type of information. Valid types are interpretation, grammar and content. For details refer to *DivaSpeechRecognizerResultType*.

See Also

DivaEventSpeechRecognizerProgress, DivaOpenSpeechRecognizer, DivaCloseSpeechRecognizer, DivaStartSpeechRecognizer, DivaStopSpeechRecognizer, DivaGetSpeechRecognizerResult

DivaSetSpeechRecognizerParameter

DivaSetSpeechRecognizerParameter sets or updates the parameter for speech recognition.

```
DWORD   DivaSetSpeechRecognizerParameter ( DivaCallHandle               hdCall,
                                           const char*                  Name,
                                           const char*                  Value );
```

Parameter

hdCall

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

Name

[in] The *Name* parameter specifies the name of the parameter.

Value

[in] The *Value* parameter specifies the value to be set for the parameter.

Return values

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

Remarks

The function sets the parameter for speech recognition for the call specified by the parameter *hdCall*. If the parameter already exists the value will be changed to the new value. If the parameter does not exist the parameter will be added to the end of the list of existing parameters.

The function can be called for an open speech recognizer session. If the recognizer session is not open or the speech recognition process is running the function will return *DivaErrorInvalidState.*

See Also

DivaEventSpeechRecognizerProgress, DivaOpenSpeechRecognizer, DivaCloseSpeechRecognizer, DivaStartSpeechRecognizer, DivaStopSpeechRecognizer, DivaSetSpeechRecognizerGrammar

DivaSetSpeechRecognizerGranmar sets or updates the grammar for speech recognition.

```
DWORD   DivaSetSpeechRecognizerGrammar ( DivaCallHandle            hdCall,
                                         const char*               ContentType,
                                         const char*               ContentId,
```

```
                                        const char*                 Grammar );
```

Parameter

hdCall

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

ContentType

[in] The *ContentType* parameter specifies the type of content provided by the *Grammar* parameter.

ContentId

[in] The *ContentId* parameter specifies the identifier to be sent with the grammar definition.

Grammar

[in] The *Grammar* parameter specifies the grammar to be used. The format depends on the *ContentType* parameter, e.g. a URI list or an XML based definition.

Return values

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

Remarks

The function sets the grammar for speech recognition for the call specified by the parameter *hdCall*. If the grammar is already defined it will be overwritten. The grammar and the type of content are passed unchanged to the speech recognizer.

The function can be called for an open speech recognizer session. If the recognizer session is not open or the speech recognition process is running the function will return *DivaErrorInvalidState.*

See Also

DivaOpenSpeechRecognizer, DivaCloseSpeechRecognizer, DivaStartSpeechRecognizer, DivaStopSpeechRecognizer

DivaSetSpeechRecognizerGrammar

DivaSetSpeechRecognizerGranmar sets or updates the grammar for speech recognition.

```
DWORD   DivaSetSpeechRecognizerGrammar ( DivaCallHandle             hdCall,
                                         const char*                ContentType,
                                         const char*                ContentId,
                                         const char*                Grammar );
```

Parameter

hdCall

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

ContentType

[in] The ContentType parameter specifies the type of content provided by the Grammar parameter.

ContentId

[in] The ContentId parameter specifies the identifier to be sent with the grammar definition.

Grammar

[in] The Grammar parameter specifies the grammar to be used. The format depends on the ContentType parameter, e.g. a URI list or an XML based definition.

Return values

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

Remarks

The function sets the grammar for speech recognition for the call specified by the parameter hdCall. If the grammar is already defined it will be overwritten. The grammar and the type of content are passed unchanged to the speech recognizer.

The function can be called for an open speech recognizer session. If the recognizer session is not open or the speech recognition process is running the function will return DivaErrorInvalidState.

See Also

DivaOpenSpeechRecognizer, DivaCloseSpeechRecognizer, DivaStartSpeechRecognizer, DivaStopSpeechRecognizer, DivaSetSpeechRecognizerParameter

DivaCreateSpeechRecognizer

DivaCreateSpeechRecognizer creates a speech recognizer instance and assigns common parameter for all sessions.

```
DWORD   DivaCreateSpeechRecognizer ( const char*            Name,
                                     DivaMrcpVersion        Version,
                                     const char*            ServerName,
                                     const char*            ServerIpAddress,
                                     DWORD                  ServerPort,
                                     const char*            DefaultMediaPath,
                                     const char*            LocalIpAddress );
```

Parameter

Name

[in] The *Name* parameter is a symbolic name that must be unique within the system. The name identifies the speech server for all following default parameter settings and for opening a session at this recognizer.

Version

[in] The *Version* parameter specifies the MRCP version used for communication with the speech recognizer.

ServerName

[in] The *ServerName* parameter specifies the network name of the server that is running the recognizer. The parameter is optional if a *ServerIpAddress* is specified. If no name is provided, the *ServerIpAddress* will be used.

ServerIpAddress

[in] The *ServerIpAddress* parameter specifies the IP address of the server running the recognizer. The parameter is optional if a *ServerName* is provided.

ServerPort

[in] The *ServerPort* parameter specifies the port number the server is listening for requests. If set to zero the default port for the MRCP version is used, e.g. 4900 for MRCPv1.

DefaultMediaPath

[in] The *DefaultMediaPath* parameter specifies any optional media path to be used to address the recognizer. The parameter is optional. If not specified the speech recognizer is addressed by the IP address or name and the port.

LocalIpAddress

[in] The *LocalIpAddress* parameter specifies the local IP address to be used. The parameter is optional. By default the IP address of the default network interface is used. The application may specify an address if multiple network interfaces are available in a system.

Return values

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

Remarks

The function creates an object for the given parameter. The object must have a unique symbolic name. If the name already exists the application will return *DivaErrorInvalidParameter*. The application may assign default parameter and default grammar to the object. If a session is created via *DivaOpenSpeechRecognizer* these default parameter are used to initialize the session.

See Also

DivaSetSpeechRecognizerDefaultParameter, DivaSetSpeechRecognizerDefaultGrammar, DivaOpenSpeechRecognizer

DivaSetSpeechRecognizerDefaultParameter

DivaSetSpeechRecognizerDefaultParameter sets the default parameter for this speech recognizer.

```
DWORD   DivaSetSpeechRecognizerDefaultParameter ( const char*   Name,
                                                  const char*   ParameterName,
                                                  const char*   ParameterValue );
```

Parameter

Name

[in] The *Name* parameter identifies the speech recognizer. The name has been assigned by a call to *DivaCreateSpeechRecognizer* or by an instance in the configuration file.

ParameterName

[in] The *ParameterName* parameter specifies the name of the parameter.

ParameterValue

[in] The *ParameterValue* parameter specifies the value to be set for the parameter.

Return values

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

Remarks

The function sets the default parameter for speech recognition. These parameters will be implicitly assigned to a recognition session when DivaOpenSpeechRecognizer is called using the Name of this recognizer.

See Also

DivaOpenSpeechRecognizer, DivaCreateSpeechRecognizer, DivaSetSpeechRecognizerDefaultGrammar

DivaSetSpeechRecognizerDefaultGrammar

DivaSetSpeechRecognizerDefaultGranmar sets the default grammar for this speech recognizer.

```
DWORD   DivaSetSpeechRecognizerDefaultGrammar ( const char*         Name,
                                                const char*         ContentType,
                                                const char*         ContentId,
                                                const char*         Grammar );
```

Parameter

Name

[in] The *Name* parameter identifies the speech recognizer. The name has been assigned by a call to *DivaCreateSpeechRecognizer*.

ContentType

[in] The *ContentType* parameter specifies the type of content provided by the *Grammar* parameter.

ContentId

[in] The *ContentId* parameter specifies the identifier to be used for the *Grammar*.

Grammar

[in] The *Grammar* parameter specifies the grammar to be used. The format depends on the *ContentType* parameter, e.g. a URI list of an XML based definition.

Return values

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

Remarks

The function sets the default grammar for speech recognition. These grammar parameters will be implicitly assigned to a recognition session when *DivaOpenSpeechRecognizer* is called using the Name of this recognizer.

See Also

DivaOpenSpeechRecognizer, DivaCreateSpeechRecognizer, DivaSetSpeechRecognizerDefaultParameter

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 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)
- DivaCompleteCallTransfer
- DivaBlindCallTransfer
- DivaAcceptCallTransfer
- DivaRejectCallTransfer
- DivaListenChannel
- DivaLIConnect
- DivaLIDisconnect
- DivaLIEnableRxData
- DivaHold
- DivaRetrieve
- DivaSendInfo
- DivaSendFlash

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 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 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 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_NoHoldBeforeTransfer

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 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 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

DivaAcceptCallTransfer

DivaAcceptCallTransfer accepts a call transfer request.

```
DWORD    DivaAcceptCallTransfer (    DivaCallHandle          hdCall );
```

Parameters

hdCall

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

Return values

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

Remarks

The function is only available on SIP calls. When a call transfer is invoked by the remote peer, the application will be notified about the request. By default, the notification is disabled and can be enabled via the call property *DivaCPT_TransferRequestNotification*.

If the call transfer notification is enabled, the event *DivaEventTransferRequested* is signaled to the application. The application may retrieve additional information about the transfer request via the call properties. To accept the call transfer, the application calls *DivaAcceptCallTransfer*. To reject a transfer request, the function *DivaRejectCallTransfer* can be used.

By default, the transfer notification is disabled and any transfer request is accepted. Once the call transfer has been performed, the event *DivaEventCallTransferredNotify* is signaled to the application.

See also

DivaRejectCallTransfer,DivaEventTransferRequested,DivaCPT_TransferRequestNotification,

DivaRejectCallTransfer

DivaRejectTransfer rejects a call transfer request from the remote peer.

```
DWORD    DivaRejectCallTransfer (    DivaCallHandle    hdCall,
                                     DWORD             Reason );
```

Parameters

hdCall

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

Reason

[in] The parameter *Reason* contains the reason for the reject. Valid reasons are specified in *DivaTransferRejectReasons*.

Return values

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

Remarks

The function is only available on SIP calls. For a description of call transfer notifications, accept and reject, refer to *DivaAcceptCallTransfer*.

See also

DivaEventTransferRequested, DivaCPT_TransferRequestNotification, DivaAcceptCallTransfer

DivaListenChannel

DivaListenChannel registers a incoming call notification on a specific channel.

```
DWORD    DivaListenChannel (    DivaApplHandle    hApp,
                                DivaListenType    ServiceType,
                                DWORD             LineDevice,
                                DWORD             Channel,
                                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] This parameter specifies the line device on which the listening should be enabled. Line defines are numbered from one to the maximum installed.

Channel

[in] This parameter specifies the channel on which the listening should be enabled. Channels are numbered from one to the maximum available on the line device.

pCalledNumber

[in] The 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 *DivaErrorInvalidHandle*, *DivaErrorLineDevice*, or *DivaErrorInvalidParameter*.

Remarks

The function behaves like *DivaListen* on a specific channel. Refer to *DivaListen* for more information on services and number filtering.

The function allows for reserving channels for outgoing calls by placing listens only on those channels that should be used for incoming calls.

See also

DivaRegister, DivaListen, DivaEventIncomingCall, DivaListenType

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 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. Both calls must have an assigned data channel and by in the call state DivaCallStateRinging, DivaCallStateOffering, DivaCallStateAnswered, DivaCallStateProceeding or DivaCallStateConnected. By defaullt no data traffic between the application and the calls take 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, DivaEventCustomToneDetected, 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 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, DivaEventCustomToneDetected, 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 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, DivaEventCustomToneDetected, DivaEventLIDisconnected

DivaHold

DivaHold puts the specified call on hold.

```
DWORD    DivaHold (    DivaCallHandle    hdCall );
```

Parameters

hdCall

[in] The *hdCall* parameter identifies the 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 *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 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 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 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 the following 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 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 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 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

DivaConferenceGetProperties provides information on members and status of a conference.

```
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 the following MWI functions:

- DivaMWIActivate
- DivaMWIDeactivate
- DivaMWIReport
- DivaMWIGetIndication

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

DivaMWIReport

DivaMWIReport enables or disables the reporting of message waiting indications.

```
DWORD    DivaMWIActivate (    DivaAppHandle    hApp,
                              DWORD            LineDeviceId,
                              BOOL             bEnable );
```

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.

bEnable

[in] The parameter bEnable specifies whether the reporting of message waiting indications is enabled or disabled.

Return values

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

Remarks

The function enables or disables the reporting of message waiting indications. Received message waiting indications are reported via the event *DivaEventMWIIndicated*.

See also

- DivaEventMWIIndicated
- DivaMWIGetIndication

DivaMWIGetIndication

DivaMWIGetIndication returns detailed information about a received message waiting indication.

```
DWORD    DivaMWIGetIndication (  DivaAppHandle              hApp,
                                 DWORD                     LineDeviceId,
                                 DivaHandle                hMWIIndication,
                                 DivaMWIIndicationParams*  pParams,
                                 DWORD                     ParamSize );
```

Parameters

hApp

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

LineDeviceId

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

hMWIIndication

[in] The parameter *hMWIIndication* specifies a handle to address the message waiting information. See Remarks.

pParams

[out] The parameter *pParams* specifies a pointer to a user-supplied buffer of type DivaMWIIndicationParams that receives the information.

ParamSize

[in] The parameter *ParamSize* specifies the length of the user-supplied buffer.

Return values

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

Remarks

The function retrieves information about the message waiting indication. The availability of message waiting information is signaled via the event *DivaEventMWIIndicated*. With this event, a handle is provided that can be used as the parameter *hMWIIndication*. If the parameter *hMWIIndication* is set to zero, the most recent message waiting indication is returned.

See also

- DivaEventMWIIndicated
- DivaMWIReport

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 Dialogic® Diva® API Call Properties.

This section contains the following call properties:

- DivaSetCallProperties
- DivaGetCallProperties
- DivaDefaultCallProperties

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 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 Dialogic® Diva® API Call Properties 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, Dialogic® Diva® API Call Properties

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 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 Dialogic® Diva® API Call Properties 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, Dialogic® Diva® API Call Properties

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 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
- Message loop

For callback function and message loop signaling, the event is provided directly. For signaling of an event object, the event has to be retrieved by a function call.

If the application has selected that events should trigger an event object, the application has to create a thread waiting for this object. Typically, operating system-specific events do not allow parameters to be given to the thread when the event is signaled. Therefore, the application has to call *DivaGetEvent* to obtain the event.

The following functions are provided:

- Callback function
- CallbackEx function
- CallbackSignal function
- DivaGetEvent
- Message loop

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 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 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 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.

Message loop

If the application has selected event reporting to a Windows® message loop, it has to pass a Windows® handle and a unique message identifier to the Diva API.

No additional parameters are passed with the Windows® message. *lParam* and *wParam* are set to zero. The application has to call *DivaGetEvent* to retrieve the event.

Monitoring

This chapter contains the following monitoring functions:

DivaMonitorAttachToTimeslot

DivaMonitorAttachToTimeslot is used to create a link to the specified timeslot for audio recording. The function is used for monitoring audio only.

```
DWORD   DivaMonitorAttachToTimeslot (   DivaMonitorHandle   hdMonitor,
                                        DWORD               Timeslot,
                                        DivaCallHandle      *phdCall );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitorAudio.*

Timeslot

[in] The parameter *Timeslot* identifies the timeslot or B-channel.

phdCall

[in] The parameter *phdCall* points to a location that receives the Diva API-related handle for monitoring the line.

Return values

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

Remarks

The function is only valid on monitoring objects created via DivaCreateMonitorAudio. Only applications that retrieve the signaling information via a 3rd party source, e.g., via CSTA or SS7, use this function. If signaling information is provided by the Diva API, the attach is done implicitly and the handle is provided via the event-specific parameter.

The function returns immediately and provides a call handle that can be used for Diva API calls to monitor audio signals, e.g., DivaMonitorRecordAudio. The application does not need to detach the handle for each monitored call. The recording can be controlled on a per call basis using *DivaMonitorRecordAudio* and *DivaMonitorStopAudio*. When the application no longer needs any access to the line, the handle is freed by a call to *DivaMonitorDetachHandle*.

See also

DivaMonitorDetachHandle, DivaMonitorAttachToLine

DivaMonitorDetachHandle

DivaMonitorDetachHandle frees a handle returned by *DivaMonitorAttachToLine* or *DivaMonitorAttachToTimeslot*.

```
DWORD   DivaMonitorDetachHandle (   DivaMonitorHandle   hdMonitor,
                                    DivaCallHandle      *phdCall );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor* or *DivaCreateMonitorAudio*.

hdCall

[in] The parameter *hdCall* identifies the handle to be detached from the monitoring object. The handle is returned by *DivaMonitorAttachToLine* or *DivaMonitorAttachToTimeslot*.

Return values

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

Remarks

The function detaches the handle from the monitoring object. This implicitly closes all pending actions on the line or timeslot, e.g., a pending audio recording.

See also

DivaMonitorAttachToTimeslot, DivaMonitorAttachToLine

DivaMonitorSpecifyTone

Via *DivaMonitorSpecifyTone*, the application specifies a custom ringing or dial tone cadence.

```
DWORD   DivaMonitorSpecifyTone (   DivaMonitorHandle   hdMonitor,
                                   DWORD               Type,
                                   DWORD               Recurrences,
                                   DWORD               NumDefinitions,
                                   DivaToneDefinition* pDefinitions );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by DivaCreateMonitorAnalog.

Type

[in] The parameter *Type* identifies how the tone should be interpreted. Valid options are DivaRingingTone and DivaBusyTone.

Recurrences

[in] The parameter *Recurrences* specifies how often the tone and pause sequence specified by *pDefinitions* must be repeated to trigger the event specified by *Type*.

NumDefinitions

[in] The parameter *NumDefinitions* specifies the number of members in the definition table pointed to by *pDefinitions*.

pDefinitions

[in] The parameter *pDefinitions* points to an array containing elements of type DivaToneDefinition. These elements describe the frequencies, duration, pause, and variations of the tone.

Return values

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

Remarks

By default, the Dialogic® Diva® SDK will use a set of build-in definitions to detect ringing and busy conditions. The application may specify custom tones and cadences to be interpreted as ringing or busy tone. Each tone definition specified via *DivaToneDefinition* contains the specification about the tone, namely, whether the tone is single or dual, what the duration of the tone is, and optionally what the tone pause is. Multiple definitions can be used to define the cadence to be detected. If the cadence should occur multiple times before this is interpreted as detected tone, the parameter *Recurrences* can be used.

DivaMonitorAttachToLine

DivaMonitorAttachToLine is used to create a link to the specified line for audio recording if call progress analysis is disabled.

```
DWORD    DivaMonitorAttachToLine (  DivaMonitorHandle    hdMonitor,
                                    DWORD                Line,
                                    DivaCallHandle       *phdCall );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitorAnalog*.

Line

[in] The parameter *Line* identifies the analog line by an index starting from one.

phdCall

[in] The parameter *phdCall* points to a location that receives the Diva API related handle for monitoring the line.

Return values

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

Remarks

The function is only valid on monitoring objects created via DivaCreateMonitorAnalog using the option DivaMonitorOptionDisableCPA. Only applications that retrieve the signaling information via a 3rd party source, e.g., via CSTA, use this function. If signaling information is provided by the Diva API, the attach is done implicitly and the handle is provided via the event-specific parameter.

The function returns immediately and provides a call handle that can be used for Diva API calls to monitor audio signals, e.g., DivaMonitorRecordAudio. The application does not need to detach the handle for each monitored call. The recording can be controlled on a per call base using *DivaMonitorRecordAudio* and *DivaMonitorStopAudio*. When the application no longer needs any access to the line, the handle is freed by a call to *DivaMonitorDetachHandle*.

See also

DivaMonitorDetachHandle, DivaMonitorAttachToTimeslot

DivaCreateMonitor

DivaCreateMonitor creates an internal monitoring object for Q.931-based lines 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 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, DivaCreateMonitorR2

DivaCreateMonitorR2

DivaCreateMonitorR2 creates an internal monitoring object for E1 R2-based lines and initiates the monitoring.

```
DWORD    DivaCreateMonitorR2 (   DivaAppHandle          hApp,
                                 AppMonitorHandle       haMonitor,
                                 DivaMonitorHandle      *phdMonitor,
                                 DWORD                  LineDeviceA,
                                 DWORD                  LineDeviceB,
                                 DWORD                  R2Variant,
                                 void*                  R2GenericParams,
                                 DWORD                  Reserved );
```

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 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.

R2Variant

[in] The parameter R2Variant identifies the variant of the E1 R2 protocol of the monitored line. For supported variants, refer to DivaMonitorR2Variants.

R2GenericParams

The parameter is reserved for future use.

Reserved

The parameter is reserved for future use.

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 *DivaCreateMonitorR2* 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 extracts signaling information from channel 16 of the given line devices and creates the corresponding events. Based on the selected R2 variant, inband information for called and calling party number is extracted from B-channels and reported with the events.

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, DivaCreateMonitor

DivaCreateMonitorAudio

DivaMonitorCreateAudio creates an internal monitoring object for monitoring audio signal on all timeslots of a line. Signaling information is not retrieved by this object.

```
DWORD    DivaCreateMonitorAudio (  DivaAppHandle      hApp,
                                   AppMonitorHandle  haMonitor,
                                   DivaMonitorHandle *phdMonitor,
                                   DWORD             LineDeviceA,
                                   DWORD             LineDeviceB
```

Parameter

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 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*, *DivaErrorNotSupported*, *DivaErrorInvalidParameter*, and *DivaErrorInvalidHandle*.

Remarks

With a call to *DivaCreateMonitorAudio*, 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. On the monitoring object created by *DivaCreateMonitorAudio*, the application can attach to timeslots via *DivaMonitorAttachToTimeslot* and initiate the recording of audio. Signaling information is not handled on this object; all timeslots are available for audio recording. Applications that retrieve the signaling information via a 3rd party source, e.g., via CSTA or SS7, use this function.

See also

DivaMonitorDetachHandle, DivaMonitorAttachToTimeslot

DivaCreateMonitorAnalog

DivaCreateMonitorAnalog creates an internal monitoring object for "analog-based" lines and initiates the monitoring.

```
DWORD    DivaCreateMonitorAnalog (    DivaAppHandle            hApp,
                                      AppMonitorHandle         haMonitor,
                                      DivaMonitorHandle        *phdMonitor,
                                      DWORD                    LineDevice,
                                      DWORD                    Options,
                                      DivaMonitorAnalogParams  *pParams );
```

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 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.

LineDevice

[in] The parameter *LineDevice* identifies the analog line device connected to the lines to be monitored.

Options

[in] The parameter *Options* defines generic rules for the monitoring. See Remarks section.

pParams

[in] The parameter *pParams* allows applications to specify parameters for detecting a connect, default audio formats, and recording gain. If the application wants to use the default parameter, *pParams* may be set to zero.

Return values

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

Remarks

With a call to *DivaCreateMonitorAnalog*, the application creates a monitor object and accesses the given line device. The function returns right away. The start of the monitor is signaled by the event *DivaEventMonitorStatus* with the status set to DivaMonitorStarted. The Diva SDK will automatically attach to all lines of the Dialogic® Diva® Analog Media Board. When the monitor object detects an active layer 1, it will signal another status event for the line with the status set to DivaMonitorLayer1Up.

By default, the Dialogic® Diva® SDK will use standard detection for ring tones, human talker detection, and busy tones to generate the events *DivaEventMonitorCallConnected* and *DivaMonitorCallDisconnected*. Via the *Options* parameter, the application can overwrite this behavior.

The default detection waits for a human talker detection to signal the connected event. The Diva SDK will store the audio during detection such that the initial audio sequence is part of the recorded audio. The disconnect is generated when the standard busy signal is detected. The standard busy tone has the frequency of 425 Hz with a duration of about 160 milliseconds and a pause of 400 milliseconds. By default, three continuous occurrences of the tone are used to interpret this as a busy tone. By default, the Diva SDK will remove the busy tone from the recorded audio.

See also

DivaCreateMonitor, DivaCreateMonitorR2, DivaCreateMonitorAudio, DivaMonitorAnalogParams, DivaMonitorOptions

DivaCreateMonitorT1CAS

DivaCreateMonitorT1CAS creates an internal monitoring object for T1CAS-based lines and initiates the monitoring.

```
DWORD    DivaCreateMonitorT1CAS (    DivaAppHandle              hApp,
                                     AppMonitorHandle           haMonitor,
                                     DivaMonitorHandle          *phdMonitor,
                                     DWORD                      LineDeviceA,
                                     DWORD                      LineDeviceB,
                                     DWORD                      T1CASVariant,
                                     DWORD                      Options,
                                     DivaMonitorT1CASParams*    pParams );
```

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 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 lines to be monitored.

LineDeviceB

[in] The parameter *LineDeviceB* identifies the second line device connected to the lines to be monitored. See Remarks section.

T1CASVariant

[in] The parameter *T1CASVariant* defines the T1 CAS variant to be used. See DivaMonitorT1CASVariants for supported variants.

Options

[in] The parameter *Options* defines generic rules for the monitoring. See Remarks section.

pParams

[in] The parameter *pParams* allows applications to specify parameter to customize monitoring. If the application wants to use the default parameter, *pParams* may be set to zero.

Return values

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

Remarks

With a call to *DivaCreateMonitorT1CAS*, 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 cabling must ensure that LineDeviceA monitors the traffic from the user (TE) to the network (NT) and LineDeviceB monitors the traffic from the network to the user.

The monitor extracts signaling information from the upper bit of each timeslot and creates the corresponding events. The processing of the signaling information depends on the parameter *T1CASVariant*, which allows specifying the trunk types LoopStart, GroundStart, and WinkStart. In addition to the signaling information from the upper bit, the Diva SDK will process tones for call progress analyses. By default, the Diva SDK will use standard detection for ring tones, human talker detection, and busy tones to generate the events *DivaEventMonitorCallConnected* and *DivaMonitorCallDisconnected*. Via the *Options* parameter the application can overwrite this behavior.

The default detection waits for a human talker detection to signal the connected event. The Diva SDK will store the audio during detection to ensure the initial audio sequence is part of the recorded audio. The disconnect is signaled when the standard busy signal is detected. The standard busy tone has the frequency 425 Hz with a duration of about 160 milliseconds and a pause of 400 milliseconds. Three continuous occurrences of the tone are used to interpret this as a busy tone. The Diva SDK will remove the busy tone from the recorded audio if default options are used.

The application may add customized ring or busy tones via the function DivaMonitorSpecifyTone. Via the option DivaMonitorOptionDisableStandardTones the standard tone may be disabled.

See also

DivaCreateMonitor, DivaCreateMonitorR2, DivaCreateMonitorAudio, DivaDestroyMonitor

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 Dialogic® Diva® API Call Properties 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 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 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 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 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 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 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

DivaMonitorReportFrames

DivaMonitorReportFrames enables or disables the reporting of layer 2 or layer 3 signaling frames.

```
DWORD    DivaMonitorReportFrames (    DivaMonitorHandle      hdMonitor,
                                      DivaMonitorReportMode  Mode );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

Mode

[in] The parameter *Mode* specifies if reporting of frames should be enabled or disabled. If enabled, it also specifies if layer 2 or layer 3 frames should be reported. See DivaMonitorFrameReportMode for options.

Return values

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

Remarks

With a call to *DivaMonitorReportFrames*, the application controls if pure layer 2 or layer 3 frames should be reported to the application. By default, high level call control events are reported to the application, but no frames.

If enabled, the Dialogic® Diva® SDK will signal the event *DivaEventMonitorFrameReceived* for each received frame that matches the Mode parameter. The application retrieves the frame and additional information by the function *DivaMonitorGetFrame*.

See also

DivaCreateMonitor, DivaMonitorGetFrame, DivaEventMonitorFrameReceived

DivaMonitorGetFrame

DivaMonitorGetFrame retrieves a layer 2 or layer 3 frame from the monitoring object.

```
DWORD    DivaMonitorGetFrame (    DivaMonitorHandle      hdMonitor,
                                  DivaHandle             hFrame,
                                  DivaCallHandle         *phdCall,
                                  DWORD                  *pLineDevice,
                                  DivaTime               *pTimeStamp,
                                  unsigned char          *pBuffer,
                                  DWORD                  BufferLength,
                                  DWORD                  *pBytesUsed );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*.

hFrame

[out] The parameter *hFrame* identifies the frame to be retrieved. The application may set this parameter to zero to retrieve the most recent received frame. See the Remarks section for detailed information.

phdCall

[out] The parameter *phdCall* points to a location that receives the call handle of the call to which the frame belongs. If the frame does not belong to a call, INVALID_APP_CALL_HANDLE will be returned. If the application does not need the information, the parameter can be set to zero.

pLineDevice

[out] The parameter *pLineDevice* points to a location that receives the line device on which the frame was received. If the application does not need the information, the parameter can be set to zero.

pTimeStamp

[out] The parameter *pTimeStamp* points to a location that receives the time stamp when the frame was received. If the application does not need the information, the parameter can be set to zero.

pBuffer

[out] The parameter *pBuffer* points to a location that receives the signaling frame.

BufferLength

[in] The parameter *BufferLength* specifies the length (in bytes) of the buffer specified by the *pBuffer* parameter.

pBytesUsed

[out] The parameter *pBytesUsed* points to a location that receives the amount of bytes written to the buffer specified by the *pBuffer* parameter.

Return values

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

Remarks

The application retrieves plain layer 2 or layer 3 frames via a call to *DivaMonitorGetFrame* on monitoring objects that are Q.931-based and created with *DivaCreateMonitor*. The reception and reporting of layer 2 or layer 3 frames must be enabled via *DivaMonitorReportFrames*.

In addition to the high level call control events, the Dialogic® Diva® SDK signals the event *DivaEventMonitorFrameReceived* for each frame that matches the mode specified by *DivaMonitorReportFrames*. The second parameter, delivered with the event *DivaEventMonitorFrameReceived*, specifies a handle that identifies the frame. The application may use this handle as parameter for *hFrame* to retrieve the frame. To retrieve the most recent frame, the application sets the parameter *hFrame* to zero.

See also

DivaCreateMonitor, DivaMonitorReportFrames, DivaEventMonitorFrameReceived

DivaMonitorReportDTMF

DivaMonitorReportDTMF enables or disables the DTMF detection for a monitored call.

```
DWORD   DivaMonitorReportDTMF ( DivaMonitorHandle          hdMonitor,
                                DivaCallHandle             hdCall
                                BOOL                       bEnable );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*, *DivaCreateMonitorR2*, *DivaCreateMonitorAnalog*, *DivaMonitorCreateMonitorT1CAS* or *DivaCreateMonitorAudio*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with one of the events *DivaEventMonitorCallInitiated*, *DivaEventMonitorCallConnected*, or *DivaEventMonitorCallInfo*. *DivaEventMonitorCallInfo*.

bEnable

[in] If the parameter *bEnable* is TRUE the detection of DTMF tones is enabled.

Return values

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

Remarks

The function enables or disables reporting of DTMF digits. Detected digits are signaled by the event *DivaMonitorEventDTMFDetected* and must be retrieved via the function *DivaMonitorGetDTMFInfo*.

See Also

DivaEventMonitorDTMFDetected, DivaMonitorGetDTMFInfo, DivaMonitorReportTone, DivaMonitorGetToneInfo, DivaEventMonitorToneDetected

DivaMonitorReportTone

DivaMonitorReportTone enables or disables the extended tone detection for a monitored call.

```
DWORD   DivaMonitorReportTone ( DivaMonitorHandle          hdMonitor,
                                DivaCallHandle             hdCall
                                BOOL                       bEnable );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*, *DivaCreateMonitorR2*, *DivaCreateMonitorAnalog*, *DivaMonitorCreateMonitorT1CAS* or *DivaCreateMonitorAudio*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with one of the events *DivaEventMonitorCallInitiated*, *DivaEventMonitorCallConnected*, or *DivaEventMonitorCallInfo*. *DivaEventMonitorCallInfo*.

bEnable

[in] If the parameter *bEnable* is TRUE the detection of extended tones is enabled.

Return values

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

Remarks

The function enables or disables reporting of extended tones. . For a list of extended tones refer to *DivaContinuousTones*. Detected tones are signaled by the event *DivaMonitorEventToneDetected* and must be retrieved via the function *DivaMonitorGetToneInfo*

See Also

DivaEventMonitorDTMFDetected, DivaEventMonitorToneDetected, DivaMonitorGetDTMFInfo, DivaMonitorReportTone, DivaMonitorGetToneInfo

DivaMonitorGetDTMFInfo

DivaMonitorGetDTMFInfo retrieves the DTMF digit information for a digit detected on a monitored call.

```
DWORD   DivaMonitorGetDTMFInfo ( DivaMonitorHandle          hdMonitor,
                                 DivaCallHandle             hdCall
                                 DivaMonitorDTMFInfo*       pInfo );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*, *DivaCreateMonitorR2*, *DivaCreateMonitorAnalog*, *DivaMonitorCreateMonitorT1CAS* or *DivaCreateMonitorAudio*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with one of the events *DivaEventMonitorCallInitiated*, *DivaEventMonitorCallConnected*, or *DivaEventMonitorCallInfo*. *DivaEventMonitorCallInfo*.

pInfo

[out] This parameter is a reference to a user-supplied buffer of the type *DivaMonitorGetDTMFInfo* that receives the information about the received DTMF digit and which side has sent the digit.

Return values

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

Remarks

The function retrieves the information about a DTMF digit detected on the monitored call. The Diva API notifies the application via the event *DivaEventMonitorDTMFDetected* that a digit is available. The application must retrieve the digit information via *DivaMonitorGetDTMFInfo*.

The information about the digit and the side that has sent the digit are returned via the data structure *DivaMonitorDTMFInfo*. If the digit is sent by the side that has initiated the call, the parameter *Source* is set to *DivaMonitorSourceOriginator*, if the side that has answered the call has sent the digit, the parameter *Source* is set to *DivaMonitorSourceAnswerer*. If the direction information is not available, e.g. on analog lines, the parameter *Source* is set to *DivaMonitorSourceUnknown*.

See Also

DivaMonitorReportDTMF, DivaEventMonitorDTMFDetected, DivaEventMonitorToneDetected, DivaMonitorReportTone, DivaMonitorGetToneInfo

DivaMonitorGetToneInfo

DivaMonitorGetToneInfo retrieves the extended tone information for a tone detected on a monitored call.

```
DWORD   DivaMonitorGetToneInfo( DivaMonitorHandle          hdMonitor,
                                DivaCallHandle             hdCall
                                DivaMonitorToneInfo*       pInfo );
```

Parameter

hdMonitor

[in] The parameter *hdMonitor* identifies the monitoring object. The handle is returned by *DivaCreateMonitor*, *DivaCreateMonitorR2*, *DivaCreateMonitorAnalog*, *DivaMonitorCreateMonitorT1CAS* or *DivaCreateMonitorAudio*.

hdCall

[in] The parameter *hdCall* identifies the call. The handle has been provided with one of the events *DivaEventMonitorCallInitiated*, *DivaEventMonitorCallConnected*, or *DivaEventMonitorCallInfo*. *DivaEventMonitorCallInfo*.

pInfo

[out] This parameter is a reference to a user-supplied buffer of the type *DivaMonitorToneInfo* that receives the information about the received tone and which side has sent the digit.

Return values

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

Remarks

The function retrieves the information about an extended tone detected on the monitored call. The Diva API notifies the application via the event *DivaEventMonitorToneDetected* that a tone is available. The application must retrieve the digit information via *DivaMonitorGetToneInfo*.

The information about the tone and the side that has sent the tone are returned via the data structure *DivaMonitorDTMFInfo*. If the tone is sent by the side that has initiated the call, the parameter *Source* is set to *DivaMonitorSourceOriginator*, if the side that has answered the call has sent the tone, the parameter *Source* is set to *DivaMonitorSourceAnswerer*. If the direction information is not available, e.g. on analog lines, the parameter *Source* is set to *DivaMonitorSourceUnknown*.

See Also

DivaMonitorReportTone, DivaEventMonitorDTMFDetected, DivaEventMonitorToneDetected, DivaMonitorReportDTMF, DivaMonitorGetDTMFInfo, DivaMonitorReportTone, DivaMonitorGetToneInfo

IP Media Channel Access

This chapter contains the following IP media channel functions:

- DivaCreateIPMediaChannel
- DivaConnectIPMediaChannel
- DivaDisconnectIPMediaChannel
- DivaCloseIPMediaChannel

DivaCreateIPMediaChannel

The *DivaCreateIPMediaChannel* function creates a virtual call object to control an IP Media Channel.

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

Parameters

hApp

[in] The parameter *hApp* 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 event is generated by the Dialogic® Diva® SDK. You can use it to pass an index or a pointer to a structure or class, to help you keep track of multiple IP media channels in the same application.

phdCall

[out] This parameter points to a location of type *DivaCallHandle* that receives the call handle on successful return. This call handle must be used when making Diva API calls to control the media channel.

Return values

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

Remarks

The function creates a virtual call object for plain media access. Only IP media streaming will be possible on this object, and certain call properties will be ignored. The application may set certain IP streaming-related call properties before calling *DivaOpenIPMediaChannel*. Once the IP Media Channel is confirmed via the event *DivaEventIPMediaChannelStatus*, the application may use functions to stream audio, or to record audio or conferencing functions.

Note: This function does not handle any signaling, e.g., SIP. The application must ensure that the remote endpoint is aware about port and media information.

See Also

DivaConnectIPMediaChannel, DivaDisconnectIPMediaChannel, DivaCloseIPMediaChannel, DivaEventIPMediaChannelStatus

DivaConnectIPMediaChannel

DivaConnectIPMediaChannel initiates an IP media channel for streaming only.

```
DWORD    DivaConnectIPMediaChannel (    DivaCallHandle    hdCall,
                                        char*             LocalIpAddress,
                                        DWORD             LocalPort,
                                        char*             RemoteIpAddress,
                                        DWORD             RemotePort,
                                        DWORD             CodecMask,
                                        DWORD             Options );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the media channel at the Diva API. The handle is returned by *DivaCreateIPMediaChannel*.

LocalIpAddress

[in] The parameter *LocalIpAddress* specifies the local IP address to be used in RTP packets.

LocalPort

[in] The parameter *LocalPort* specifies the local UDP port to listen for incoming RTP packets.

RemoteIpAddress

[in] The parameter *RemoteIpAddress* specifies the IP address to which RTP packets should be sent.

RemotePort

[in] The parameter *RemotePort* specifies the remote UDP port to be used as the destination port in RTP packets.

CodecMask

[in] The parameter *CodecMask* specifies the codecs to enable, for example, a-Law, µ-Law and/or DTMF. Refer to DivaCodecMask for available codecs.

Options

[in] The parameter *Options* is reserved for future use.

Return values

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

Remarks

With a call to *DivaConnectIPMediaChannel*, the application initiates RTP streaming to an endpoint defined by the given parameter. The result of the operation is reported by the event *DivaEventIPMediaChannelStatus*. Once the event is received, the RTP streaming between the local and the remote endpoint is initiated and the application may use the voice related function to stream and record audio.

Note: This function does not handle any signaling, e.g., SIP. The application must ensure that the remote endpoint is informed about port and media information.

See Also

DivaCreateIPMediaChannel, DivaDisconnectIPMediaChannel, DivaCloseIPMediaChannel, DivaEventIPMediaChannelStatus

DivaDisconnectIPMediaChannel

The *DivaDisconnectIPMediaChannel* function stops the RTP streaming for the media channel..

```
DWORD    DivaDisconnectIPMediaChannel (    DivaCallHandle        hdCall);
```

Parameter

hdCall

[in] The parameter *hdCall* identifies the media channel at the Diva API. The handle is returned by *DivaCreateIPMediaChannel*.

Return values

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

Remarks

With a call to *DivaDisconnectIPMediaChannel*, the application terminates the RTP streaming between the local and remote endpoint. The termination of the RTP streaming is indicated via the event *DivaEventIPMediaChannelStatus*.

See Also

DivaCreateIPMediaChannel, DivaConnectIPMediaChannel, DivaCloseIPMediaChannel, DivaEventIPMediaChannelStatus

DivaCloseIPMediaChannel

The *DivaCreateIPMediaChannel* function creates a virtual call object to control an IP Media Channel.

```
DWORD    DivaCloseIPMediaChannel (    DivaCallHandle        hdCall );
```

Parameters

hdCall

[in] The parameter *hdCall* identifies the media channel at the Diva API. The handle is returned by *DivaCreateIPMediaChannel*.

Return values

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

Remarks

With a call to *DivaCloseIPMediaChannel*, the application terminates the RTP streaming between the local and remote endpoint and frees the object created via *DivaCreateIPMediaChannel*. After the function returns, the handle is no longer valid.

See Also

DivaCreateIPMediaChannel, DivaConnectIPMediaChannel, DivaDisconnectIPMediaChannel, DivaEventIPMediaChannelStatus

Audio provider

This chapter contains the following 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 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 the following timer handling functions:

- DivaStartCallTimer
- DivaStopCallTimer
- DivaStartApplicationTimer
- DivaStopApplicationTimer

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 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 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 available:

- DivaEnableTrace
- DivaSetTraceFile
- DivaLogPrintf

DivaEnableTrace

DivaEnableTrace controls the trace level at runtime.

```
BOOL    DivaEnableTrace (    DWORD    nLevel );
```

Parameter

nLevel

[in] This parameter sets the level of tracing.

Return values

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 );
```

Parameters

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.

Return values

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 );
```

Parameters

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.

Return values

None

Remarks

The function allows an application to trace into the standard Diva SDK trace environment.

See also

No references.

Static and dynamic initialization functions

IP-based line devices are configured via *DivaSetDeviceInitParameter*. Note that the Diva SDK places the Dialogic® Diva® Media Board devices in front of IP-based line devices by default. Therefore, the line device numbers used to create virtual IP boards may differ from line device numbers after *DivaInitialization*. The Diva API provides the line device of the first virtual IP board via *DivaGetFirstIPLineDevice*.

The dynamic parameters can be updated at any time after the initialization via *DivaInitialize* succeeded. Most of the dynamic parameters are updated asynchronously and the result is reported via an event. In order to receive events, the application must register via *DivaRegister*. Note that only one registration via *DivaRegister* is recommended per process. Examples for asynchronous dynamic parameters are registrations for SIP registrar and H.323 gatekeeper via *DivaRegisterSIPRegistrar* and *DivaRegisterH323Gatekeeper*.

The following functions are available:

- DivaSetInitParameter
- DivaSetDeviceInitParameter
- DivaGetFirstIPLineDevice
- DivaRegisterSIPRegistrar
- DivaReleaseSIPRegistrar
- DivaRegisterH323Gatekeeper
- DivaReleaseH323Gatekeeper
- DivaGetRegistrationResult
- DivaSetH323Gateway
- DivaCloseRegistration

DivaSetInitParameter

DivaSetInitParameter sets the initialization parameter to be used during *DivaInitialize*.

```
DWORD    DivaSetInitParameter (   DivaInitParameterTypes    Type,
                                  DivaInitParameterValue*   pValue );
```

Parameters

Type

[in] The parameter specifies the type of the value to be set. Valid values are defined in *DivaInitParameterTypes*.

pValue

[in] The *pValue* parameter points to a location, where the parameter value is located. The value and the length depend on the *Type*. See Remarks.

Return values

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

Remarks

The function sets a certain parameter for initialization. All parameters must be set before *DivaInitialize* is called.

The parameter value depends on the parameter *Type*. In general, the value has the data type *DivaInitParameterValue* and the length is given by the type. If shorter values of basic types, e.g., Boolean, are used, the reference to this parameter can be passed directly by using the cast operator. Below is a sample code for setting a Boolean value directly.

```
BOOL              bSet = TRUE;
DivaSetInitParameter (DivaParamDisableTDMDevices, (DivaInitParameterValue*) &bSet );
```

See also

DivaInitialize, DivaInitParameterTypes

DivaSetDeviceInitParameter

DivaSetDeviceInitParameter sets the initialization parameter for a line device to be used during *DivaInitialize*.

```
DWORD    DivaSetDeviceInitParameter (  DWORD                          LineDevice,
                                       DivaDeviceInitParameterTypes   Type,
                                       DivaDeviceInitParameterValue*  pValue );
```

Parameters

LineDevice

[in] This parameter specifies the line device. This is an index starting with one. See remarks.

Type

[in] This parameter specifies the type of the value to be set. Valid values are defined in *DivaDeviceInitParameterTypes*.

pValue

[in] The *pValue* parameter points to a location, where the parameter value is located. The value and the length depend on the *Type*. See Remarks.

Return values

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

Remarks

The function sets a device specific parameter for initialization. All parameters for all devices to be created need to be set before *DivaInitialize* is called.

The parameter value depends on the parameter type. In general, the value has the data type *DivaDeviceInitParameterValue* and the length is given by the type. If shorter values of basic types, e.g., Boolean, are used, the reference to this parameter can be passed directly by using the cast operator.

The line device parameter needs to start with one and is incremented for each device to be created. The maximum amount of IP-based line devices to be created is specified via the *DivaSetInitParameter*.

Note that the Dialogic® Diva® SDK by default arranges the line devices as TDM devices first and then the IP-based devices. Therefore, the application needs to call *DivaGetFirstIPLineDevice* to align the devices initialized via *DivaSetDeviceInitParameter* to the real device numbers after initialization.

See also

DivaInitialize, DivaDeviceInitParameterTypes

DivaGetFirstIPLineDevice

DivaGetFirstIPLineDevice provides the number of the first line device that belongs to a virtual IP device.

```
DWORD    DivaGetFirstIPLineDevice (  DWORD      pLineDevice );
```

Parameters

pLineDevice

[out] The *pLineDevice* parameter specifies a location that receives the line device number of the first virtual IP device. See Remarks.

Return values

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

Remarks

When the application configures the virtual IP-based line devices, it numbers them starting from one. During initialization, the Diva API enumerates TDM-based line devices first and adds the configured virtual IP-based devices. *DivaGetFirstIPLineDevice* returns the number of the first device that belongs to a virtual IP device. All IP-based line devices are numbered continuously. If no virtual IP device is available, the function returns *DivaErrorNotSupported*.

See also

DivaInitialize, DivaSetDeviceInitParameter

DivaRegisterSIPRegistrar

DivaRegisterSIPRegistrar initiates the registration at a SIP registrar server.

```
DWORD   DivaRegisterSIPRegistrar (  DivaAppHandle            hApp,
                                    DWORD                    LineDevice,
                                    DivaHandle               hAppRegistrar,
                                    DivaHandle*              phDivaRegistrar,
                                    DivaSIPRegistrarParams * pParams );
```

Parameters

hApp

[in] The *hApp* parameter specifies the application instance created via *DivaRegister* for event notification. See remarks.

LineDevice

[in] The parameter specifies the line device. The line device is an index starting with one up to the maximum number of line devices.

hAppRegistrar

[in] The value of *hAppRegistrar* passed in here will be given back to the application whenever a callback or an event is generated by the Dialogic® Diva® SDK. You can use it to pass, e.g., an index or a pointer to a structure, to help you keep track of multiple registrations in the same application.

phDivaRegistrar

[out] The *phDivaRegsitar* parameter points to a location of type *DivaHandle* that receives the Diva API handle for this registration request.

pParams

[in] The *pParams* parameter points to a user supplied buffer that contains the registration parameter. Refer to DivaSIPRegistrarParams for details.

Return values

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

Remarks

The function initiates the registration at a SIP registrar server. The function is only valid on SIP-based line devices; if called on other line devices, the function returns *DivaErrorNotSupported*.

If the registration request was sent successfully, the function returns *DivaSuccess*. The status of the registration is reported via the event *DivaEventRegistrationStatus*. For assignment of events to registrations, the application may specify a handle that is signaled with the events. This handle may have any value and is not interpreted by the Dialogic® Diva® SDK. The function returns a handle for further actions related to this registration, e.g., to cancel the registration via *DivaReleaseSIPRegistrar*.

The parameters for the registration are passed in the data structure *DivaSIPRegistrarParams*. Refer to the description of DivaSIPRegistrarParams for more detailed information about the parameter.

The event notification is done on the context returned via *DivaRegister*. It is recommended for applications to register only once via *DivaRegister*. If multiple registrations via *DivaRegister* are done, the SIP registrations are valid for all instances registered with *DivaRegister*.

See also

DivaReleaseSIPRegistrar, DivaEventRegistrationStatus, DivaSIPRegistrarParams

DivaReleaseSIPRegistrar

DivaReleaseSIPRegistrar initiates the release of a registration at a SIP registrar server.

```
DWORD    DivaReleaseSIPRegistrar (   DivaHandle    hDivaRegistrar );
```

Parameter

hDivaRegistrar

[in] The *hDivaRegistrar* parameter contains a handle and identifies the registration. The handle has been returned by *DivaRegisterSIPRegistrar*.

Return values

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

Remarks

The function initiates the release of a previous registration at a SIP registrar server. The function is only valid on SIP-based line devices. If it is called on other line devices, the function returns *DivaErrorNotSupported*.

If the function returns success, the progress is reported by the event *DivaEventRegistrationStatus*. If no registration is active or pending, the function returns *DivaErrorInvalidState*.

Note that the handles remain valid until the application calls *DivaCloseRegistration*.

See also

DivaRegisterSIPRegistrar, DivaEventRegistrationStatus, DivaSIPRegistrarParams

DivaRegisterH323Gatekeeper

DivaRegisterH323Gatekeeper initiates the registration at a H.323 gatekeeper.

```
DWORD    DivaRegisterH323Gatekeeper (  DivaAppHandle              hApp,
                                       DWORD                     LineDevice,
                                       DivaHandle                hAppGatekeeper,
                                       DivaHandle*               phDivaGatekeeper,
                                       DivaH323GatekeeperParams *  pParams );
```

Parameters

hApp

[in] The *hApp* parameter specifies the application instance created via *DivaRegister* for event notification. See Remarks.

LineDevice

[in] The parameter specifies the line device. The line device is an index starting with one up to the maximum number of line devices.

hAppGatekeeper

[in] The value of *hAppGatekeeper* passed in here will be given back to the application whenever an event is indicated by the Dialogic® Diva® SDK for this request. You can use it to pass, e.g., an index or a pointer to a structure, to help you keep track of multiple registrations in the same application.

phDivaGateKeeper

[out] The *phDivaGatekeeper* parameter points to a location of type *DivaHandle* that receives the Diva API handle for this registration request.

pParams

[in] The *pParams* parameter points to a user supplied buffer that contains the registration parameter. For details, refer to DivaH323GatekeeperParams.

Return values

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

Remarks

The function initiates the registration at a H.323 gatekeeper. The function is only valid on H.323-based line devices; if called on other line devices, the function returns *DivaErrorNotSupported*.

If the registration request was successfully sent, the function returns *DivaSuccess*. The status of the registration is reported via the event *DivaEventRegistrationStatus*. For assignment of events to registrations, the application may specify a handle that is signaled with the events. This handle may have any value and is not interpreted by the Diva SDK. The function returns a handle for further actions related to this registration, e.g., to cancel the registration via *DivaReleaseH323Gatekeeper*.

The parameters for the registration are passed in the data structure DivaH323GatekeeperParams. Refer to the description of the structure for more detailed information about the parameter.

The event notification is done on the context returned via *DivaRegister*. Applications should register only once via *DivaRegister*. If multiple registrations via *DivaRegister* are done, the SIP registrations are valid for all instances registered with *DivaRegister*.

See also

DivaReleaseH323Gatekeeper, DivaEventRegistrationStatus, DivaH323GatekeeperParams, DivaSetH323Gateway

DivaReleaseH323Gatekeeper

DivaReleaseH323Gatekeeper initiates the release registration at a H.323 gatekeeper.

```
DWORD    DivaReleaseH323Gatekeeper (  DivaHandle      hDivaRegistrar );
```

Parameter

hDivaRegistrar

[in] The *hDivaRegistrar* parameter contains a handle and identifies the registration. The handle has been returned by *DivaRegisterH323Gatekeeper*.

Return values

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

Remarks

The function initiates the release of a previous registration at a H.323 gatekeeper. The function is only valid on H.323-based line devices, if called on other line devices the function returns *DivaErrorNotSupported*.

If the function returns success, the progress is reported by the event *DivaEventRegistrationStatus*. If no registration is active or pending, the function returns *DivaErrorInvalidState*.

The handles remain valid until the application calls *DivaCloseRegistration*.

See also

DivaRegisterH323Gatekeeper, DivaEventRegistrationStatus, DivaH323GatekeeperParams

DivaGetRegistrationResult

DivaGetRegistrationResult provides the result of the registration and optionally whether a retry is recommended.

```
DWORD    DivaGetRegistrationResult (    DivaHandle      hDivaRegistrar,
                                        DWORD*          pResult,
                                        BOOL*           pRetry )
```

Parameters

hDivaRegistrar

[in] The *hDivaRegistrar* parameter contains a handle and identifies the registration. The handle has been returned by *DivaRegisterSIPRegistrar* or *DivaRegisterH323Gatekeeper*.

pResult

[out] The *pResult* parameter points to a location of type DWORD that receives the result of the registration.

pReply

[out] The *pReply* parameter points to a location of type BOOL and receives the information whether a retry is recommended or not. The parameter is optional and may be set to zero.

Return values

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

Remarks

When a registration is completed, the event *DivaEventRegistrationStatus* provides information whether the registration was successful or failed. If the registration failed, the application may call *DivaGetRegistrationResult* to retrieve the reason of the failure. For possible failures, refer to *DivaRegistrationResults*. Optionally, the application may pass a pointer to a variable of type BOOL that receives the information whether a retry is recommended.

The function can be called if the status signaled via the event *DivaEventRegistrationStatus* is *DivaRegStatusRegistered*, *DivaRegStatusFailed*, or *DivaRegStatusReleased*. For all other states, the result *DivaErrorInvalidState* is returned.

See also

DivaRegisterH323Gatekeeper, DivaRegisterSIPRegistrar, DivaEventRegistrationStatus

DivaSetH323Gateway

DivaSetH323Gateway specifies a H.323 gateway to be used for outgoing calls.

```
DWORD   DivaSetH323Gateway (   DWORD          LineDevice,
                               const char *   pGatewayAddress );
```

Parameters

LineDevice

[in] This parameter specifies the line device. The line device is an index starting with one up to the maximum number of line devices.

pGatewayAddress

[in] The *pGatewayAddress* parameter specifies the gateway IP address and optional the port number.

Return values

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

Remarks

The function *DivaSetH323Gateway* stores the information about a default gateway to be used. If a gateway is specified, the Dialogic® Diva® SDK will add the gateway information to outgoing calls that are initiated with a phone number only. The gateway address is specified as a string in the format <IP address>[:port], e.g., 192.168.0.100:1010. To remove a previously set gateway, call this function with an empty gateway address.

Note that configuring a gateway is only valid if no registration at a gatekeeper is done.

See also

DivaRegisterH323Gatekeeper, DivaReleaseH323Gatekeeper, DivaEventRegistrationStatus, DivaH323GatekeeperParams

DivaCloseRegistration

DivaCloseRegistration frees all internal resources allocated to the registration handle.

```
DWORD   DivaCloseRegistration (   DivaHandle   hDivaRegistrar );
```

Parameter

hDivaRegistrar

[in] The *hDivaRegistrar* parameter contains a handle and identifies the registration. The handle has been returned either by *DivaRegisterSIPRegistrar* or *DivaRegisterH323Gatekeeper*.

Return values

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

Remarks

The function releases all resources allocated for the given handle. This function should be called when a registration has been released or has failed. The function is synchronous and returns immediately. When the function returns, the handle of the Dialogic® Diva® SDK is invalidated.

If the registration is still pending, the Diva SDK will implicitly release the registration. During this time, registration resources in the Diva SDK may be blocked and a new registration may not be possible immediately.

See also

DivaRegisterH323Gatekeeper, DivaRegisterSIPRegistrar, DivaEventRegistrationStatus

IP-specific functions

With Dialogic® Diva® SDK 5.5, several IP-specific features are introduced. These features will only be available on IP-based line devices. All IP-specific features will only be available on SIP-based line devices or on calls that are initiated or answered on these line devices.

All IP-specific features are optional, and, if not used, the default handling known from previous Diva SDK releases will take place.

- DivaRegisterSIPHeader
- DivaGetSIPHeader
- DivaSetSIPHeader

DivaRegisterSIPHeader

DivaRegisterSIPHeader registers a SIP header to be reported to the application if included in a received SIP message.

```
DWORD   DivaRegisterSIPHeader (   DWORD         LineDevice,
                                  const char    Name );
```

Parameters

LineDevice

[in] This parameter specifies the line device. The line device is an index starting with one up to the maximum number of line devices.

Name

[in] The *Name* parameter specifies the SIP header to be registered. This can be a standard SIP header, e.g., "Contact" or a private header either application- or PBX-specific. The maximum length for the name parameter is MAX_ADDR_LEN.

Return values

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

Remarks

By default, the Diva API abstracts the communication protocol-specific parameters, so that applications are independent from the underlying protocol. If applications require SIP-specific information, they may register to receive a certain SIP header. The SIP header may be a standard SIP header like "Contact" or any private header. The availability of a SIP header is signaled via the event *DivaEventSIPMessageReceived*. The registration for SIP header is done per line device and is valid for all calls processed on this line device.

See also

DivaGetSIPHeader, DivaSetSIPHeader

DivaGetSIPHeader

DivaGetSIPHeader retrieves a SIP header from the Diva API.

```
DWORD   DivaGetSIPHeader (   DivaCallHandle    hdCall,
                             DivaHandle        hMessage,
                             char*             pName,
                             char*             pBuffer,
                             DWORD             BufferSize,
                             DWORD*            pSizeNeeded );
```

Parameters

hdCall

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

hMessage

[in] The *hMessage* parameter specifies the SIP message for which the SIP header should be retrieved. A value of zero specifies that the oldest available SIP header is retrieved, independent from any SIP message identifier.

pName

[out] The *pName* parameter specifies a location that receives the name of the SIP header. The length of the buffer must be MAX_ADDR_LEN.

pBuffer

[out] The *pBuffer* parameter specifies a location that receives the SIP header information. For information about buffer size, refer to Remarks.

BufferSize

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

pSizeNeeded

[out] The *pSizeNeeded* parameter specifies a location of type DWORD that receives the required length. The parameter is optional and may be set to zero.

Return values

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

Remarks

The function retrieves the SIP header information for a specific call. The Diva API signals the available SIP header via the event *DivaEventSIPMessageReceived*, if enabled by the application via *DivaRegisterSIPHeader*. With the event, the application gets a message identifier that specifies the SIP message belonging to the event. The application may use this handle to retrieve the SIP headers for a specific SIP message.

If the application passes the value zero as message identifier *hMessage*, the oldest available SIP header is retrieved. Applications should call *DivaGetSIPHeader* in a loop until the result *DivaErrorNoDataAvailable* is returned.

Note: For SIP headers that are signaled with the initial INVITE message, the event *DivaEventSIPMessageReceived* is not signaled. The event *DivaEventIncomingCall* implicitly signals that a SIP header might be available. The application may retrieve the SIP header belonging to the initial INVITE using either *DivaSIPInitialMessageHandle* or zero as message identifier.

The Diva API buffers a certain amount of SIP headers before they are overwritten. Once a message is retrieved via *DivaGetSIPHeader*, the message is removed from the internal queue.

The application must specify two memory locations to retrieve a SIP header. The memory location that receives the name of the SIP header is expected to be MAX_ADDR_LEN long. The memory location that receives the header information may vary in length. The application may retrieve the required length by passing a valid pointer for the *pSizeNeeded* parameter. The Diva API will return the required length in *pSizeNeeded* if the buffer is not specified or the length of the buffer does not fit. If the length does not fit, the Diva API returns *DivaErrorInsufficientBuffer*.

See also

DivaRegisterSIPHeader, DivaSetSIPHeader, DivaEventSIPMessageReceived

DivaSetSIPHeader

DivaSetSIPHeader specifies a SIP header to be sent with the next SIP message.

```
DWORD   DivaSetSIPHeader (   DivaCallHandle   hdCall,
                             const char*      pName,
                             const char*      pHeader );
```

Parameters

hdCall

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

pName

[in] The *pName* parameter specifies the name of the SIP header to be set.

pHeader

[in] The *pHeader* parameter specifies the information to be set for the header specified by *pName*.

Return values

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

Remarks

The function sets the information for a SIP header to be included in the next SIP message sent to the peer. The message in which the SIP header is included depends on the call state, e.g., the header is send with the INVITE message if the SIP header is set before calling *DivaDial*. Once the specified header is included in a SIP message, it will be cleared from the internal storage. The application must set a new SIP header if this information should be included in the next SIP message.

See also

DivaRegisterSIPHeader, DivaGetSIPHeader

CHAPTER 5

Dialogic® Diva® API Events

This chapter describes the events reported by the 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.

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*.

Event Summary

The following tables summarize the Diva API events by group.

Connection Events

Event	Description
DivaEventIncomingCall	Signals an incoming call.
DivaEventCallProgress	Signals the progress of a call by providing information on the call setup steps.
DivaEventCallInfo	New information for the call is available since the last call to *DivaGetCallInfo.*
DivaEventCallConnected	An incoming or outgoing call is physically connected, and data transfer can be performed on this call.
DivaEventCallDisconnected	The call was disconnected.
DivaEventEarlyDataChannelConnected	Signals that voice information can be streamed. 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 have call type set to voice using the *DivaVoiceOptionEarlyDataChannel* option.
DivaEventCallDisconnectedNotify	Remote side initiated a disconnect. The event is only signaled in early data channel mode.

Call Transfer and Supplementary Services Events

Event	Description
DivaEventHoldCompleted	DivaHold initiated a hold request.
DivaEventRetrieveCompleted	DivaRetrieve initiated a retrieve request.
DivaEventSetupTransferCompleted	DivaSetupCallTransfer requested a call transfer.
DivaEventTransferCompleted	DivaCompleteCallTransfer or DivaBlindCallTransfer initiated a transfer request.
DivaEventCallHoldNotify	The remote end put the call on hold.
DivaEventCallRetrievedNotify	The remote end retrieved a call that was previously on hold.
DivaEventCallTransferredNotify	The call was transferred by the remote end.
DivaEventTransferRequested	The remote end requested a call transfer.
DivaEventMWICompleted	A message activation or deactivation request sent via *DivaMWIActivate* or *DivaMWIDeactivate* completed.
DivaEventMWIIndicated	A message waiting indication was detected.
DivaEventFlashCompleted	A hook flash initiated by *DivaSendFlash* was completed.
DivaEventLIConnectCompleted	Line Interconnect was established
DivaEventLIDisconnected	Line Interconnect was released

VoIP Events

Event	Description
DivaEventIPMediaChannelStatus	Signals whether access to an IP media channel via *DivaConnectIPMediaChannel* was successful or whether an error occurred.
DivaEventSIPMessageReceived	The application received a SIP message containing one or more headers for which it registered.

Data Transfer Events

Event	Description
DivaEventDataAvailable	New data is available.
DivaEventDataSent	The data passed for sending has been sent, and the buffer is free to be used by the application.
DivaEventDataChannelStatus	The status of the data channel has changed. This event is only signaled if the application enabled manual data channel mode via the call property *DivaCPT_ManualDataChannel*.
DivaEventDTMFInitialDigitTimeout	Signals information about the receive status of modem data, e.g. if reception of a data block is in progress. This event is only signaled if the application enabled the reporting of data status information via the call property *DivaCPT_EnableDataStatusReporting*.

Fax Events

Event	Description
DivaEventFaxPageSent	A fax page has been sent.
DivaEventFaxSent	All pages of a fax have been sent.
DivaEventFaxPageReceived	A fax page has been received.
DivaEventFaxReceived	All pages of a fax have been received.
DivaEventDetailedFaxStatus	Fax status information is available. This event is only signaled if the application enables the reporting of fax status information via the call property DivaCPT_EnableFaxStatusReporting.

Voice Events

DivaEventRecordVoiceEnded	The recording of the audio data has ended.
DivaEventSendVoiceEnded	The streaming of the audio data has ended.
DivaEventSendVoiceDone	(Obsolete) The streaming of the audio has ended.
DivaEventSendVoiceRestarted	The streaming of the audio data was restarted.
DivaEventSendVoiceCanceled	The streaming of the audio data was terminated by the application.
DivaEventDTMFReceived	A DTMF tone was detected.
DivaEventSendDTMFToneEnded	The streaming of a DTMF tone or of the last tone of a sequence initiated by DivaSendDTMF has ended.
DivaEventToneDetected	The start or stop of a continuous tone or a multi-frequency tone was detected. The detection must be enabled by calling DivaReportTones.
DivaEventSendToneEnded	The streaming of a tone or of the last tone of a sequence initiated by DivaSendTone or DivaSendContinuousTone has ended.
DivaEventDTMFMaxDigits	DTMF processing for the group *DivaProcessingGroupEvent* was enabled via via DivaSetDTMFProcessingRules, and the maximum amount of digits were received.

DivaEventDTMFTerminationDigit	DTMF processing for the group *DivaProcessingGroupEvent* was enabled via DivaSetDTMFProcessingRules, and one of the enabled termination digits was received.
DivaEventDTMFInterDigitTimeout	DTMF processing for the group *DivaProcessingGroupEvent* was enabled via DivaSetDTMFProcessingRules, and the inter digit timeout was reached.
DivaEventDTMFInitialDigitTimeout	DTMF processing for the group *DivaProcessingGroupEvent* was enabled, and no DTMF digit was received within the initial digit timeout set via DivaSetDTMFProcessingRules.
DivaEventDTMFMaxTimeout	DTMF processing for the group *DivaProcessingGroupEvent* was enabled via DivaSetDTMFProcessingRules, and no other part of the rules specified expired before the maximum timeout was detected.
DivaEventAnsweringMachineDetector	The answering machine detector, started with DivaEnableAnsweringMachineDetector, has finished the analyses and reports the result.
DivaEventGenericToneEnded	The generic tone generator, enabled via *DivaGenerateSingleTone* or *DivaGenerateDualTone*, has ended due to a timeout condition.
DivaEventGenericToneDetected	The generic tone detector, enabled via *DivaDetectSingleTone* or *DivaDetectDualTone*, has detected a tone that matches the detection parameter.
DivaEventGenericToneInfo	An answer to a low level generic tone request is available. The request was previously issued by the application via the function *DivaSendGenericToneRequest*.
DivaEventCustomToneDetected	A custom tone specified via DivaSpecifyCustomTone was detected.
DivaEventGenericToneDetected	The generic tone detector, enabled via *DivaDetectSingleTone* or *DivaDetectDualTone*, has detected a tone that matches the detection parameter.
DivaEventConferenceInfo	New information concerning the current conference is available.

Device Status Events

Event	Description
DivaEventDeviceStatusChanged	The status of a device has changed.
DivaEventRegistrationStatus	The status of a registration at a SIP registrar or H.323 gatekeeper changed.

Monitoring/Line Tapping Events

Event	Description
DivaEventMonitorCallInitiated	A call was initiated on the monitored line
DivaEventMonitorCallConnected	A call was connected at the signaling level.
DivaEventMonitorCallDisconnected	A call was disconnected at the signaling level.
DivaEventMonitorCallInfo	The state of a call changed or additional information was received as information messages.
DivaEventMonitorFrameReceived	A layer 2 or layer 3 frame is available on the monitored line.
DivaEventMonitorRecordEnded	Recording stopped.
DivaEventMonitorStatus	The status of a monitor object created by DivaCreateMonitor changed.
DivaEventMonitorAudioData	Audio data for the monitored call is available can be retrieved by the application.
DivaEventMonitorDTMFDetected	DTMF was detected on the monitored call.
DivaEventMonitorToneDetected	An extended tone or human talker was detected on the monitored call.

Timer Events

Event	Description
DivaEventCallTimer	A timer started with DivaStartCallTimer expired.
DivaEventApplicationTimer	A timer started with *DivaStartApplicationTimer* expired.

SMS Events

Event	Description
DivaEventSms1MsgReceived	An SMS message arrived.
DivaEventSmsError	SMS detected an error.

The remainder of this chapter each Diva API event. The events are listed in alphabetical order

Speech Recognizer Events

Event	Description
DivaEventSpeechRecognizerStatus	An event signals the status of a speech recognizer session of a call.
DivaEventSpeechRecognizerProgress	An event signals the progress of a speech recognizer session of a call.

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.

DivaEventApplicationTimer

The event *DivaEventApplicationTimer* is signaled when a timer started with *DivaStartApplicationTimer* expires. There is no event specific parameter for this event.

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 Diva API. This parameter should be used to close the call via DivaCloseCall.

Note: There is no application-specific call handle and no disconnect event is sent if the application has rejected a call.

DivaEventCallDisconnectedNotify

The event *DivaEventCallDisconnectNotify* is only signaled in early data channel mode. The event indicates that the remote side has initiated the disconnect. The Diva SDK will not release the call and applications may still listen to the data channel, e.g., to hear the busy tone. The call is finally released when the application calls *DivaDisconnect*.

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.

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*.

DivaEventCallRetrievedNotify

The event *DivaEventCallRetrieveNotify* is signaled when the remote end retrieves a call that was previously on hold. The *EventSpecific1* parameter contains the call handle of the application.

DivaEventCallTimer

The event *DivaEventCallTimer* is signaled when a timer started with DivaStartCallTimer expires. The parameter *EventSpecific1* 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.

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 Diva API when the conference was created.

DivaEventCustomToneDetected

The event *DivaEventCustomTone* is signaled if the application has specified one or more customer tones via DivaSpecifyCustomTone. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the identifier of the tone assigned, with the tone definition passed via DivaSpecifyCustomTone.

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.

DivaEventDataChannelStatus

The event *DivaEventDataChannelStatus* signals that the status of the data channel has changed. The event is only signaled if the application has enabled manual data channel mode via the call property *DivaCPT_ManualDataChannel*. The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the new state of the data channel. For possible data channel states, refer to DivaEventDataChannelStatus.

DivaEventDataFrameStatus

The *DivaEventDataFrameStatus* event signals information about the receive status of modem data, e.g., if reception of a data block is in progress. The signaling of this event must be enabled via the call property *DivaCPT_EnableDataStatusReporting*. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* contains information about the available status information. Refer to DivaDataFrameStatus for details about available data status options.

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.

DivaEventDetailedFaxStatus

The *DivaEventDetailedFaxStatus* event signals the availability of detailed fax status information. The signaling of this event must be enabled via the call property *DivaCPT_EnableFaxStatusReporting*. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* contains information about the available status information. Refer to DivaFaxStatusType for details about available fax status types.

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*.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

DivaEventFaxSent

The *DivaEventFaxSent* event is signaled when all pages have been sent. At this moment, the remote side can automatically disconnect. 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.

DivaEventFSKDataDetected

The event *DivaEventFSKDataDetected* is signaled when the FSK detector has been enabled via DivaDetectFSKData. The parameter *EventSpecific1* contains the application call handle. The *EventSpecific2* parameter contains the detected character or type. The lower 16 bits contain the character, the upper 16 bits contain the event type. For valid types, refer to DivaFSKEventTypes. In 64 bit systems, the upper 32 bits are unused.

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.

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.

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.

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*.

DivaEventIncomingCall

The *DivaEventIncomingCall* event signals a new incoming call. The *EventSpecific1* parameter contains the 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.

DivaEventIPMediaChannelStatus

The *DivaEventIPMediaChannelStatus* event signals whether access to an IP media channel via *DivaConnectIPMediaChannel* was successful or whether an error occurred. The parameter *EventSpecific1* contains the virtual call handle of the application, which is provided with a call to *DivaCreateIPMediaChannel*. The parameter *EventSpecific2* contains status information. Refer to DivaMediaChannelStatus for details about available status information.

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.

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.

DivaEventMonitorCallConnected

The event *DivaEventMonitorCallConnected* is signaled when a call was connected at the signaling level.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the 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 was disconnected at the signaling level.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the 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 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.

DivaEventMonitorCallInitiated

The event *DivaEventMonitorCallInitiated* is signaled when a call was initiated on the monitored line.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the 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.

DivaEventMonitorFrameReceived

The event *DivaEventMonitorFrameReceived* is signaled when a layer 2 or layer 3 frame is available on the monitored line.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the Diva API when the monitor is created via DivaCreateMonitor. The parameter *EventSpecific2* contains the handle of the frame that may be used to retrieve the frame information via DivaMonitorGetFrame.

DivaEventMonitorRecordEnded

The event *DivaEventMonitorRecordEnded* is signaled when the recording stops.

The parameter *EventSpecific1* contains the monitor handle of the application passed to the Diva API when the monitor is created. The parameter *EventSpecific2* contains the reason for the stopping, typically user initiated.

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 Diva API when the monitor is created. The parameter *EventSpecific2* contains the new status. See the DivaMonitorStatus for possible status messages.

DivaEventMonitorDTMFDetected

The event DivaEventMonitorDTMFDetected is signaled when a DTMF tone has been detected on the monitored call. The parameter EventSpecific1 contains the monitor handle of the application passed to the Diva API when the monitor is created. The parameter EventSpecific2 contains the handle of a call object. The application could retrieve the information about the DTMF digit via DivaMonitorGetDTMFInfo.

DivaEventMonitorToneDetected

The event DivaEventMonitorToneDetected is signaled when an extended tone has been detected on the monitored call. The parameter EventSpecific1 contains the monitor handle of the application passed to the Diva API when the monitor is created. The parameter EventSpecific2 contains the handle of a call object. The application could retrieve the information about the extended tone via DivaMonitorGetToneInfo.

DivaEventMWICompleted

The event *DivaEventMWICompleted* is signaled when a message activation or deactivation request sent 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.

DivaEventMWIIndicated

The *DivaEventMWIIndicated* event is signaled when a message waiting indication is detected. The *EventSpecific1* parameter contains the line device ID of the device that received the message waiting indication. The *EventSpecifc2 parameter* contains a handle to retrieve the message waiting information via the DivaMWIGetIndication function. The reporting of Message waiting indications must be enabled via the DivaMWIReport.

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.

DivaEventRegistrationStatus

The *DivaEventRegistrationStatus* event is signaled when the status of a registration at a SIP registrar or H.323 gatekeeper changes. The *EventSpecific1* parameter contains the handle of the application passed by the application to *DivaRegisterSIPRegistrar* or *DivaRegisterH323Gatekeeper*. The *EventSpecific2* parameter contains the new status of the registration. For valid values, refer to the enumeration DivaRegistrationStatus.

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*.

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.

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.

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.

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.

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*.

DivaEventSIPMessageReceived

The *DivaEventSIPMessageReceived* event is signaled when a SIP message is received containing one or more headers that the application has registered for. The *EventSpecific1* parameter contains the application call handle. The *EventSpecific2* parameter contains an identifier to retrieve the SIP header information belonging to this SIP message via DivaGetSIPHeader.

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-byte integer.

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 DivaReportTones.

The *EventSpecific1* parameter contains the call handle of the application. The *EventSpecific2* parameter contains the signaled tone. The continuous tones are defined in DivaContinuousTones, and the multi-frequency tones are defined in DivaMultiFrequencyTones.

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 due to the hold request or the transfer itself.

DivaEventTransferRequested

The event *DivaEventTransferRequested* is signaled when the remote party requests a call transfer. The event is only signaled if the call property *DivaCPT_TransferRequestNotification* is enabled. *Param1* contains the application call handle. *Param2* is unused and will always contain zero.

DivaEventSpeechRecognizerStatus

The *DivaEventSpeechRecognizerStatus* event signals the status of a speech recognizer session of a call. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* contains information about the status. For possible values refer to DivaSpeechRecognizerStatus.

DivaEventSpeechRecognizerProgress

The *DivaEventSpeechRecognizerProgress* event signals the status of a speech recognizer session of a call. The parameter *EventSpecific1* contains the call handle of the application. The parameter *EventSpecific2* contains information about the progress. For possible values refer to DivaSpeechRecognizerProgress.

CHAPTER 6

Dialogic® Diva® API Call Properties

The Diva API call properties allow application to set or retrieve information. All call properties are optional and allow for enhancing functionality and for keeping the Diva API interface compatible with previous versions. The following lists the available call property types by category.

Common Call Properties for All Call Types

Property	Value	Definition
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. Refer to DivaCallType for infromation on available call types.
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 Diva API automatically selects a line device. On reading, the actual selected line device is reported.
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.
DivaCPT_ManualDataChannel	Set	The *DivaCPT_ManualDataChannel* is a write only property to specify that the application handles the data channel manually. Refer to DivaEnableDataChannel for more information.
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.

Property	Value	Definition
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_CalledNumberParams	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_CallingNumberParams	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_RedirectionNumber	Get	This is a read only parameter that provides the redirection number if a call is transferred by the remote party.
DivaCPT_RedirectReason	Get Set	The *DivaCPT_RedirectReason* is a read write property to set or retrieve the redirect reason.
DivaCPT_SecondCallingNumber	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_SecondCallingNumber Params	Get	The parameter is read only and provides the parameter of a second calling party number. For information about this parameter, see DivaNumberInformation. A second calling party number may be signaled by SMS gateways.
DivaCPT_CallingName	Get Set	The parameter is read and write. 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.
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_OriginalCalledNumber	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_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_SignaledLineDiscReason	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*.

Property	Value	Definition
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.

Voice, Streaming, VAD, Talker and Tone Detection

Property	Value	Definition
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_VoiceEarlyDataChannel	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_VoiceRecordSilenceTimeout	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.*
DivaCPT_VoiceRecordStartTones= 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_SendDuration	Set	The property is write only and specifies the duration (in milliseconds) of generated DTMF tones.
DivaCPT_VoiceDTMF_SendPause	Set	The property is write only and specifies the pause (in milliseconds) of generated DTMF tones.
DivaCPT_VoiceDTMF_DetectDuration	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_DetectPause	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_VoiceRemoveDTMFFromStream	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_VoiceEarlyDataDiscOnInfo	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_EchoCancellerEnableNLP	Set	The property is write only and enables the non-linear processing for the echo canceller.
DivaCPT_EchoCancellerAutoDisable1	Set	The property is write only and bypasses the echo canceller upon detection of phase reversed 2100 Hz (operation according to G.165).
DivaCPT_EchoCancellerAutoDisable2	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_EchoCancellerTailLength	Set	The property is write only. Echo canceller time span in milliseconds; default is implementation-specific.
DivaCPT_EchoCancellerPreDelay	Set	The property is write only. Echo canceller pre-delay before starting.
DivaCPT_EnableDTMFTrailingEdge	Set	The property is a write only and enables the reporting of the training edge of a DTMF tone. The default is disabled.
DivaCPT_DataCodec	Set	The *DivaCPT_DataCodec* specifies that the data between the application and the Diva API is exchanged in compressed mode. For supported codecs, refer to DivaDataCodec. The property is only valid for calls handled on Diva Media Boards.

Property	Value	Definition
DivaCPT_DataCodecOptions	Set	The *DivaCPT_DataCodecOptions* allows options to be specified for a data codec selected via *DivCPT_DataCodec*.
DivaCPT_DataCodecSamplesPerPacket	Set	*DivaCPT_DataCodecSamplesPerPacket* allows for setting the maximum amount of bytes that are collected before received audio is indicated to the application. Note that the maximum packet size is also defined by the registration parameter set with DivaRegister.
DivaCPT_DisableMFDetection	Set	The property is write only and specifies if MF tone should be reported by the tone detector. By default, the tone detector, enabled via *DivaReportTones*, reports MF tones. This property allows to disable MF tone detection. The property must be set before calling *DivaReportTone*.
DivaCPT_EnableHookSignaling	Set	The property is write only and specifies if hook signaling tones should be reported by the tone detector. By default, the tone detector, enabled via *DivaReportTones*, does not report hook signaling tones. This property allows enabling detection of hook signaling. The property must be set before calling *DivaReportTone*. Refer to DivaContinuousTones for information on hook signaling tones.
DivaCPT_EnableR2ForwardDetection	Set	The property is write only and specifies if MF R2 Forward tones should be reported by the tone detector. By default, the tone detector, enabled via DivaReportTones, does not report MF R2 Forward tones. This property allows enabling detection of MF R2 Forward tones. The property must be set before calling *DivaReportTone*. Refer to DivaR2Tones for information on MF R2 Forward tones.
DivaCPT_EnableR2BackwardDetection	Set	The property is write only and specifies if MF R2 Backward tones should be reported by the tone detector. By default, the tone detector, enabled via DivaReportTones, does not report MF R2 Backward tones. This property allows enabling detection of MF R2 Backward tones. The property must be set before calling *DivaReportTone*. Refer to DivaR2Tones for information on MF R2 Backward tones.
DivaCPT_FSKData	Get	The property is read only and provides the received FSK Data. The application must enable FSK detection via *DivaDetectFSKData*. The received FSK data is cleared after the application has read the data.
DivaCPT_DTMFMode	Set	The property sets the preferred DTMF mode for an IP-based call. By default, the Diva API selects the DTMF mode automatically via the following rules: If possible, DTMF is exchanged via RFC 2833. If RFC 2833 is not possible, a fallback to inband is done. If the used codec does not allow inband SIP, Info messages are used on SIP-based devices. The options to switch the DTMF mode are shown in DivaDTMFMode. When the property is read, the currently active DTMF mode is returned.
DivaCPT_EnableTransparentLI	Set	The property is write only and sets the line interconnect mode to transparent. The property should be enabled if modem or digital data calls are interconnected (tromboned).
DivaCPT_HumanTalkerThreshold	Set	The property is write only and sets the threshold for the human talker detector. The threshold can be set in the range of -127 dBm to 127 dBm. The recommended range is -48 dBm to 0 dBm; the default value is -43 dBm.
DivaCPT_VoiceActivityThreshold	Set	The property is write only and sets the threshold for the voice activity detector. The threshold can be set in the range of -127 dBm to 127 dBm.
DivaCPT_VoiceDTMF_TxLevelGroup	Set	The property is write only and sets the transmit level for DTMF tones. The level can be set in the range -124 to +127.
DivaCPT_SingleToneOffDuration	Set	The property is write only and sets the off time for the generic single tone detector. By default, the single tone off event is signaled if a previously detected tone is not detected for 64 milliseconds. The application may specify a customized timeout in the range 32 to 8000 milliseconds.
DivaCPT_DualToneOffDuration	Set	The property is write only and sets the off time for the generic dual tone detector. By default, the dual tone off event is signaled if a previously detected dual tone is not detected for 64 milliseconds. The application may specify a customized timeout in the range 32 to 8000 milliseconds.
DivaCPT_HookSignalingOffHookTime	Set	The property is a write only property used to configure the minimum time for offhook detection of hook signaling. The time is specified in milliseconds, and the maximum value is 510 milliseconds. This parameter is only processed if hook signaling is enabled via the call property *DivaCPT_EnableHookSignaling*.

Property	Value	Definition
DivaCPT_HookSignalingHookFlashTime	Set	The property is a write only property used to configure the minimum time for hook flash detection of hook signaling. The time is specified in milliseconds, and the maximum value is 510 milliseconds. This parameter is only processed if hook signaling is enabled via the call property *DivaCPT_EnableHookSignaling*.
DivaCPT_ToneDetectorFFTLength	Set	The call property *DivaCPT_ToneDetectorFFTLength* is a write only property of type DWORD. The property is used to specify the minimum length the FFT detector uses to identify a tone. The range is from 8 milliseconds to 64 milliseconds. The default is 64 milliseconds. The property is valid for the single and dual tone detector and must be set before calling *DivaDetectSingleTone* or *DivaDetectDualTone*.
DivaCPT_SingleToneDetectorMinFreq	Set	The call property DivaCPT_SingleToneDetectorMinFreq is a write only property of type DWORD. The property is used to specify the minimum frequency of a single tone that should be reported to the application or processed by the internal custom tone processing. By default all tones are reported. The property is valid for the single tone detector and must be set before calling DivaDetectSingleTone.
DivaCPT_SingleToneDetectorMaxFreq	Set	The call property DivaCPT_SingleToneDetectorMaxFreq is a write only property of type DWORD. The property is used to specify the maximum frequency of a single tone that should be reported to the application or processed by the internal custom tone processing. By default all tones are reported. The property is valid for the single tone detector and must be set before calling DivaDetectSingleTone.
DivaCPT_DataCodecSampleRate	Set	The call property DivaCPT_DataCodecSampleRate is a write only property of type DWORD. The parameter is used to set the sample rate when the data codec is set to DivaDataCodecPCM16. For valid sample rates refer to DivaSampleRates.
DivaCPT_EnableNoiceSuppression	Set	The call property DivaCPT_EnableNoiceSuppression is a write only property and enables the noise suppressor for the next call initiated or answered on this call handle.
DivaCPT_NoiseSuppressionActive	Get	The call property DivaCPT_NoiseSuppressionActive is a read only property that provides the information if the noise suppressor has been enabled and is active.

Fax Call Properties

Property	Value	Definition
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 text to be printed on top of every fax page to be sent. 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.

Property	Value	Definition
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_FaxRemoteFeatures	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_FaxRemoteMaxHorzRes	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_FaxRemoteMaxVertRes	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_FaxRemoteMaxSpeed	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_FaxProcedureInterrupt	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_FaxRemoteSupports Subaddr	Get	The property is read only and provides information about whether the remote party can handle secure fax protocols.
DivaCPT_FaxRemoteSupports Password	Get	The property is read only and provides information about whether the remote party can handle secure fax protocols.
DivaCPT_FaxSignalSubAddress	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_FaxRemoteSubAddress	Get	The property is read only and provides the sub address and password of the remote party negotiated during fax T.30 negotiation.
DivaCPT_FaxRemotePassword	Get	The property is read only and provides the sub address and password of the remote party negotiated during fax T.30 negotiation.

Property	Value	Definition
DivaCPT_FaxDisableFileBuffering	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_FaxUseTextForSending	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_FaxRemoteScanLineLength	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*.
DivaCPT_FaxPassDataOnNextPage Start	Set	If this option is set, the fax data provided via *DivaReceiveFaxToMemory* is signaled when the next page starts. By default, the data is signaled when the current page ends.
DivaCPT_FaxStartPage	Set	Specifies the page of a multi-page fax document with which the transmission starts. By default, transmission starts with the first page.
DivaCPT_FaxEnableClearChannel	Set	Enables the clear channel mode for IP-based fax. Refer to DivaFaxOptions for a detailed description.
DivaCPT_EnableFaxStatusReporting	Set	The call property *DivaCPT_EnableFaxStatusReporting* enables or disables the reporting of detailed fax status information like training results, retransmits, and scanline statistics. The property is write only and must be set before calling *DivaDial* or *DivaAnswer*. The availability of detailed fax status information is signaled via the event *DivaEventDetailedFaxStatus*.
DivaCPT_FaxReportDCS	Get	The call property *DivaCPT_FaxReportDCS* is a read only property of binary data, first byte is length. The property provides the DCS information negotiated by the peers.
DivaCPT_FaxReportTrainingResult	Get	The call property *DivaCPT_FaxReportTrainingResult* is a read only property of type boolean. The property provides the result of the fax training sequence. If true, the training was successful, if false the training failed.
DivaCPT_FaxReportTrainingStats	Get	The call property *DivaCPT_FaxReportTrainingStats* is a read only property of type DivaFaxTrainingStats. The property is only available in fax receive mode. For details on available training statistics refer to DivaFaxTrainingStats.
DivaCPT_FaxReportPageQuality	Get	The call property *DivaCPT_FaxReportPageQuality* is a read only property of type DivaFaxPageQualityDetails. The property is only available in receive mode and returns details about received scan lines and errors in scan lines.
DivaCPT_FaxReportPartialPage	Get	The call property *DivaCPT_FaxReportPartialPage* is a read only property of type DivaFaxPartialPageDetails. The property returns information on ECM results received from the peer or sent to the peer, depending on the call direction.
DivaCPT_FaxReportT30Timeout	Get	The call property *DivaCPT_FaxReportT30Timeout* is a read only property of type DWORD. The property returns the number of the expired T.30 timer.
DivaCPT_FaxT30Phase	Get	The call property *DivaCPT_FaxT30Phase* is a read only property of type DivaFaxPhase. The property returns the current T.30 phase.
DivaCPT_FaxResultReport	Get	The call property *DivaCPT_FaxReportT30Timeout* is a read only property that returns the hang up code (HUC) according to T.32 as a hexadecimal value.
DivaCPT_FaxEnableBinaryFile Transfer	Set	The property is write only and enables the binary file transfer option for an incoming call. The property must be set before calling *DivaAnswer*. The application must check if binary file transfer is negotiated by reading the call property *DivaCPT_FaxBinaryFileTransferActive,* when the event *DivaEventCallConnected* is received.
DivaCPT_FaxRequestBinaryFile Transfer	Set	The property is write only and requests the binary file transfer option for an outgoing call. The property must be set before calling *DivaDial*. The application must check if binary file transfer is negotiated by reading the call property *DivaCPT_FaxBinaryFileTransferActive*,when the event *DivaEventCallConnected* is received

Property	Value	Definition
DivaCPT_FaxBinaryFileTransfer Active	Get	The property is read only and returns the result of the binary file transfer negotiation. The property can only be negotiated, if the property *DivaCTP_FaxEnableBinaryFileTransfer* or *DivaCPT_FaxRequestBinaryFileTransfer* is enabled. The result of the negotiation is available when the event *DivaEventCallConnected* is signaled.
DivaCPT_FaxSelectManual Result	Set	The property is write only and selects manual result reporting for an incoming fax call using binary file transfer. By default, the Diva SDK will confirm a reception once all data is stored or passed to the application. If this property is set, the Diva SDK will delay the confirmation until the application calls *DivaDisconnect*. The application may set a result by setting the call property *DivaCPT_FaxManualResultValue* before calling *DivaDisconnect*.
DivaCPT_FaxManualResult Value	Set	The property is write only and only used in fax binary file transfer receive mode, when the call property *DivaCPT_FaxSelectManualResult* has been set by the application. A non-zero value is interpreted as an error and reported to the sending side when the application calls *DivaDisconnect*.
DivaCPT_FaxUseHeadlineFont	Set	This write-only boolean property enables the use of a user-supplied TrueType or OpenType font file for the fax headline text. The next Fax properties are only applicable if this property is enabled; otherwise they have no effect on the headline.
DivaCPT_FaxHeadlinePageInfo	Set	The property is write only; it points to a buffer containing the right-aligned text preceding the page number and total pages.
DivaCPT_FaxHeadlineFontFile	Set	The property is write only; it must contain the path to the user-supplied font file, when the use of a font file is enabled.
DivaCPT_FaxHeadlineFontFace	Set	The property is write only and is used to select, if multiple fonts are present in the font file, any other than the first font face.
DivaCPT_FaxHeadlineFontSizePoints	Set	The property is write only and is used to indicate the font size in points when headline font is enabled. Default size is 12 points; minimum is 8 points.
DivaCPT_FaxHeadlineCoding	Set	The property is write only and it selects the character coding for the headline text. If defaults to DivaFaxHeadlineCodingASCII (8-bit ASCII extension), and it can be set to either of the other two enum values: DivaFaxHeadlineCodingUtf8 (Utf-8 charcter coding), or DivaFaxHeadlineCodingUtf16LE (Utf-16 little endian character coding). The coding applies to the entire user-provided headline text, whether left or right aligned.
DivaCPT_FaxHeadlineOmitDateTime	Set	This write-only property is a boolean to omit date and time from the headline. Otherwise the date and time are shown, left aligned, with a date format determined automatically based on the companding setting (Law) on the board.
DivaCPT_FaxHeadlineOmitPageInfo	Set	This write-only property is a boolean to omit page number and total pages from the headline, otherwise are shown right aligned.

Modem Call Properties

Property	Value	Definition
DivaCPT_MaximumSpeed =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.
DivaCPT_DisableCompression =800	Set	The property is write only and disables any compression for an analog modem connection.

Property	Value	Definition
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_NegotiatedV42V42bis	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_NegotiatedMNP4MNP5	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_NegotiatedTransparent	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_NegotiatedSDLC	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 the SDLC protocol.
DivaCPT_NegotiatedCompression	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. 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.
DivaCPT_ParityAutoDetectMode	Set	The call property *DivaCPT_ParityAutoDetectMode* is a write only property and enables or disables parity auto detection for incoming modem connections. If the auto detection for parity is enabled by the application, the Diva SDK will analyze the first received data frame based on the enabled detection method. If the parity of all bytes of the data frame matches one of the enabled parities, this parity is selected. For valid detection options, refer to *DivaCPT_ParityAutoDetectMode*. If the parity DivaParityEven or DivaParityOdd is detected, the Diva SDK will clear the upper bit of each received byte before passing the data to the application and add the parity before sending data given by the application. Note that the detection process does not know the expected data and will be based on analyzing the parity of all received bytes of the first data frame. If the received data does not contain a parity bit but the data matches one of the enabled parities, this would be false interpreted as parity.
DivaCPT_ParityAutoDetectMinData	Set	The property *DivaCPT_ParityAutoDetectMinData* allows for specifying the minimum amount of data to collect before doing the detection. By default, the first received data frame is used, independent from the length of the frame.
DivaCPT_EnableDataStatus Reporting	Set	The write only property *DivaCPT_EnableDataStatusReporting* enables or disables the reporting of data status information, such as receive started or receive aborted. The property must be set before calling *DivaDial* or *DivaAnswer*. The availability of data status information is signaled via the event *DivaEventDataFrameStatus*.
DivaCPT_AnswerToneDuration	Set	The write only call property *DivaCPT_AnswerToneDuration* sets the length of the answer tone for modem protocols. By default, the property is set to zero and uses a default of 2400 milliseconds. The answer tone duration must be given in milliseconds.

Property	Value	Definition
DivaCPT_V29FCAnswerToneDuration	Set	The write only call property DivaCPT_V29FCAnswerToneDuration sets the length of the answer tone for the modulation V.29 Fast Connect. By default, the property is set to zero and uses the default of 2500 milliseconds. The V29FC answer tone duration must be given in milliseconds the maximum is 6000 milliseconds.
DivaCPT_EnableLargeFrames	Set	The write only call property *DivaCPT_EnableLargeFrames* is used to enable larger frame sizes as alllowed by the registration parameter and buffer size limitation of the *DivaRegister*. This requires a protocol that is able to fragment and recombine data frames, and is only used if the call type is *DivaCallTypeModem* using SDLC.
DivaCPT_DetectedParity	Get Set	The call property *DivaCPT_DetectedParity* can be read and written. On read, it reports the detected parity if parity auto detection is enabled via *DivaCPT_ParityAutoDetectMode*. The application may overwrite a detected parity by setting the property. For valid values, refer to *DivaParity*.
DivaCPT_NegotiatedSDLC	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 the SDLC protocol.

Extended Modem Call Properties

Property	Value	Definition
DivaCPT_DisableV42Detection	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV29FDX	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV33	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV90APCM	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV22FS	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV29FS	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV23_1	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV23_2	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV22bisFS	Set	The property is write only and enables the V.22 bis Fast Setup modulation. This allows short negotiation periods for 2400 baud connections.
DivaCPT_EnableModulationBell202CID	Set	The property is write only and enables the modem modulation Bell 202 CID.
DivaCPT_EnableModulationBell202POS	Set	The property is write only and enables the modem modulation Bell 202 POS.
DivaCPT_EnableModulationBell103SIA	Set	The property is write only and enables the modem modulation Bell 103 SIA.
DivaCPT_EnableModulationV21Bits10	Set	The property is write only and enables the modem modulation V.21 Bits 10.
DivaCPT_EnableModulationV23Reverse	Set	The property is write only and enables the modem modulation V.23 reverse.
DivaCPT_EnableModulationFSK	Set	The property is write only and enables the FSK modulation using the default frequencies 1300 and 2100 Hz.
DivaCPT_EnableFSKExtendedMode	Set	The property is write only and enables the FSK modulation using the frequencies 1300 and 1900 Hz. If the property DivaCPT_DisableAnswerTone is not set, an initial answer tone with the frequencies 1550 and 1900 Hz is sent before the first data packet. The length of the answer tone is by default 700 milliseconds, if the property DivaCPT_EnableShortAnswerTone is set the answer tone is 200 milliseconds.
DivaCPT_EnableModulationECall	Set	The property is write only and enables the eCall modem modulation.

Property	Value	Definition
DivaCPT_ModemBitTransparentMode	Set	The call property DivaCPT_ModemBitTransparentMode enables a mode that doesn't apply any error correction or framing to the data. All bits of each data byte from the application are passed LSB first to the modulator. Bits received from the demodulator are concatenated LSB first to form data bytes for the application. This mode can be used with any modulation. If there is no data from the application in time, bytes with all '1' bits are inserted.
DivaCPT_DisableModulationV21	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV22	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV22bis	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV23	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV32	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV32bis	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV34	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV90DPCM	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationBell103	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationBell212A	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationAllFS	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationK56Flex	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationX2	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34TxLevelReduction	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34PreCoding	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34PreEmphasis	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34Shaping	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34NonLinearEncoding	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34ManualReduction	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34Training16Point	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate2400	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate2743	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate2800	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate3000	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate3200	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate3429	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

Property	Value	Definition
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_ModemLeasedLine	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_Modem4WireOption	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableDiscOnBusyTone	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableCallingTone	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableAnswerTone	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableDialToneDetect	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_EnableShortAnswerTone	Set	The property is write only and enables the short answer tone for low speed modem connections. This allows to shorten the modem negotiation process.
DivaCPT_DisableFlushTimer	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableEmptyFrames	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableMultimoding	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_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_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_DisableReliableIf1200	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_BufferDuringV42Detection	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV42SelectivReject	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.

Property	Value	Definition
DivaCPT_TransparentModeIfV22bis	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_TransparentModeIfV32bis	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_ModemEarlyConnect	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_ModemPassIndication	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.
DivaCPT_SDLCInitiateFastEstablishment	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCSingleDataPackets	Set	The property is write only. If set, the "right to send" is passed to the remote side after each data packet. If multiple packets are waiting, this allows the remote side to answer before the next packet is sent. For POS applications, this option should be set.
DivaCPT_SDLCExplicitDataAck	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCFastPollRecovery	Set	The property is write only and controls how the sending is initiated. If the property is set and data is pending, the "right to send" is requested from the peer rather than waiting for the peer to initiate the send. For POS applications, the option should be set.
DivaCPT_ModemDisconnectTimeout	Set	The property is write only and specifies a maximum time to wait until the modem data connection is disconnected before the signaling channel disconnects. The value is given in seconds. If the property is set to modem, then the data connection and the signaling connection are disconnected at the same time. The default value is zero.
DivaCPT_Modem7BitsPerChar	Set	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*.
DivaCPT_Modem2StopBits	Set	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*.
DivaCPT_ModemParityOdd	Set	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*.
DivaCPT_ModemParityEven	Set	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*.
DivaCPT_ModemEnableAutoMode	Set	If this option is set, the modem will negotiate the connection speed and mode automatically. This is currently only available for synchronous modes.
DivaCPT_ModemSDLCEnable	Set	If this option is set, the connection will use SDLC on top of the synchronous modem connection.
DivaCPT_ModemSyncV22Normal	Set	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.

Property	Value	Definition
DivaCPT_ModemSyncEnableV22bis	Set	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.
DivaCPT_ModemSyncEnableV22Fast Setup	Set	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.
DivaCPT_ModemSyncEnableV22bisFast Setup	Set	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.
DivaCPT_ModemSyncEnableV29Fast Setup	Set	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.
DivaCPT_ModemSyncSDLCExtModulo Mode	Set	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.
DivaCPT_DisableV42Detection	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV29FDX	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV33	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV90APCM	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV22FS	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV29FS	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV23_1	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV23_2	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableModulationV22bisFS	Set	The property is write only and enables the V.22 bis Fast Setup modulation. This allows short negotiation periods for 2400 baud connections.
DivaCPT_EnableModulationBell202CID	Set	The property is write only and enables the modem modulation Bell 202 CID.
DivaCPT_EnableModulationBell202POS	Set	The property is write only and enables the modem modulation Bell 202 POS.
DivaCPT_EnableModulationBell103SIA	Set	The property is write only and enables the modem modulation Bell 103 SIA.
DivaCPT_EnableModulationV21Bits10	Set	The property is write only and enables the modem modulation V.21 Bits 10.
DivaCPT_EnableModulationV23Reverse	Set	The property is write only and enables the modem modulation V.23 reverse.
DivaCPT_DisableModulationV21	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV22	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV22bis	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV23	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV32	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV32bis	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV34	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationV90DPCM	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationBell103	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

DivaCPT_DisableModulationBell212A	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationAllFS	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationK56Flex	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableModulationX2	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34TxLevelReduction	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34PreCoding	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34PreEmphasis	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34Shaping	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34NonLinearEncoding	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34ManualReduction	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34Training16Point	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate2400	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate2743	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate2800	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate3000	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate3200	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV34SymbolRate3429	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
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_ModemLeasedLine	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_Modem4WireOption	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableDiscOnBusyTone	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableCallingTone	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableAnswerTone	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableDialToneDetect	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_EnableShortAnswerTone	Set	The property is write only and enables the short answer tone for low speed modem connections. This allows to shorten the modem negotiation process.
DivaCPT_DisableFlushTimer	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

DivaCPT_EnableEmptyFrames	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_EnableMultimoding	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_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_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_DisableReliableIf1200	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_BufferDuringV42Detection	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_DisableV42SelectivReject	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_TransparentModeIfV22bis	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_TransparentModeIfV32bis	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_ModemEarlyConnect	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_ModemPassIndication	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.
DivaCPT_SDLCInitiateFastEstablishment	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCSingleDataPackets	Set	The property is write only. If set, the "right to send" is passed to the remote side after each data packet. If multiple packets are waiting, this allows the remote side to answer before the next packet is sent. For POS applications, this option should be set.
DivaCPT_SDLCExplicitDataAck	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

DivaCPT_FastFallbackToTransparent	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_MaxGarbageBytes	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_MaxTimeInDetectionPhase	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf
DivaCPT_SDLCFastPollRetryTimer	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_AnswerToneDelay	Set	The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.
DivaCPT_SDLCFastPollRecovery	Set	The property is write only and controls how the sending is initiated. If the property is set and data is pending, the "right to send" is requested from the peer rather than waiting for the peer to initiate the send. For POS applications, the option should be set.
DivaCPT_ModemDisconnectTimeout	Set	The property is write only and specifies a maximum time to wait until the modem data connection is disconnected before the signaling channel disconnects. The value is given in seconds. If the property is set to modem, then the data connection and the signaling connection are disconnected at the same time. The default value is zero.
DivaCPT_Modem7BitsPerChar	Set	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*.
DivaCPT_Modem2StopBits	Set	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*.
DivaCPT_ModemParityOdd	Set	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*.
DivaCPT_ModemParityEven	Set	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*.
DivaCPT_ModemEnableAutoMode	Set	If this option is set, the modem will negotiate the connection speed and mode automatically. This is currently only available for synchronous modes.
DivaCPT_ModemSDLCEnable	Set	If this option is set, the connection will use SDLC on top of the synchronous modem connection.
DivaCPT_ModemSyncV22Normal	Set	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.
DivaCPT_ModemSyncEnableV22bis	Set	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.
DivaCPT_ModemSyncEnableV22Fast Setup	Set	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.
DivaCPT_ModemSyncEnableV22bisFast Setup	Set	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.
DivaCPT_ModemSyncEnableV29Fast Setup	Set	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.
DivaCPT_ModemSyncSDLCExtModulo Mode	Set	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.

Modulation V.18 Call Properties

Property	Value	Definition
DivaCPT_V18Selected	Set	The property is write only and enables the V.18 mode.
DivaCPT_V18ProbingSequence	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_V18CountryProbingSequence	Set	Pre-defined country probing sequences. For available countries, refer to DivaV18DefProbings. The property is write only.
DivaCPT_V18ProbingMessage	Set	Text of the message string used for probing. The property is write only.
DivaCPT_V18ReinitializeOnSilence	Set	The property is write only and enables or disables the reinitialization on silence.
DivaCPT_V18RevertToAnswerMode	Set	The property is write only and enables or disables the revert to answer mode on timeout.
DivaCPT_V18DisconnectOnBusy	Set	The property is write only and enables or disables disconnect on busy detection.
DivaCPT_V18AutomodingMonitorMode	Set	The property is write only and enables or disables automoding monitor.
DivaCPT_V18TextProbingForCarrierMode	Set	The property is write only and enables or disables continuous carrier probing with the message.
DivaCPT_V18TXPSpaceParityInOrigMode	Set	The property is write only and enables or disables the sending of TXP with space parity in origination mode.
DivaCPT_V18EnableV18OriginationMode	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_V18EnableV18AnswerMode	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_V18EnableV21OriginationMode	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_V18EnableV21AnswerMode	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_V18EnableBell103OrigMode	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_V18EnableBell103AnswerMode	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_V18EnableV23OriginationMode	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_V18EnableV23AnswerMode	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_V18EnableEDTMode	Set	The property is write only and enables EDT mode (980/1180 Hz 110 bit/s).
DivaCPT_V18EnableBAUDOT45Mode	Set	The property is write only and enables BAUDOT 45 mode (1800/1400 Hz 22 ms/bit).
DivaCPT_V18EnableBAUDOT47Mode	Set	The property is write only and enables BAUDOT 47 mode (1800/1400 Hz 21 ms/bit).
DivaCPT_V18EnableBAUDOT50Mode	Set	The property is write only and enables BAUDOT 50 mode (1800/1400 Hz 20 ms/bit).
DivaCPT_V18EnableDTMFMode	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_V18AsyncFormatV21	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_V18AsyncFormatV23	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_V18AsyncFormatBell103	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_V18AsyncFormatEDT	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.

Property	Value	Definition
DivaCPT_V18AsyncFormatBAUDOT	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_V18TimerTcTimeout	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_V18TimerTmTimeout	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_V18CleanCarrierTime	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_V18EchoSupressTime	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.

Call Properties for Low Level Signaling Access

Property	Value	Definition
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_SpecialInfoElement	Get Set	The property is available for read and write and is used for setting-specific elements.
DivaCPT_ChannelInfoElement	Get	The property is read only and provides the channel information element as received from the line.
DivaCPT_ProgressIndElement	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_GlobalConfiguration	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_ReverseDataChannelConnect	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_CauseInfoElement	Get	*DivaCPT_CauseInfoElement* is a read only property and provides the cause information element as signaled on the underlying network.

Property	Value	Definition
DivaCPT_SendingComplete	Get Set	The property may be set for an outgoing call. A value of true indicates that the provided dial information is complete, and a value of false indicated that additional information will follow. For an incoming call, the property delivers the information if more dial information can be expected from the remote peer.
DivaCPT_LastMessage	Get	The property is read only and provides the last signaling message received in the signaling cahnnel. The availablility depends on teh underlying signaling protocol.
DivaCPT_LineInfoElement	Get	The property is read only and returns the line info element. The line info element is supported by US protocols like NI 2 and includes the service indicator. Applications that require only the service indicator may use the property DivaCPT_ServiceIndicator.
DivaCPT_ServiceIndicator	Get	The property is read only and retrieves the service indicator from a line info element. The information is only available on US protocols and it depends on the switch if the information is passed.
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.

Digital Data Call Properties

Property	Value	Definition
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_CalledAddress	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_CallingAddress	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_ReverseRestart	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.
DivaCPT_AutoDetectMode=2400	Set	The property is reserved for future use.
DivaCPT_AutoDetectX75ForceX25	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_AutoDetectMaxFrames	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_AutoDetectMaxSeconds	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 Supplementary Service Call Properties

Property	Value	Definition
DivaCPT_UseSameChannelForTransfer =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_NoHoldBeforeTransfer	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
DivaCPT_TransferRequestNotification	Set	If this option is set, the application is notified via the event *DivaEventTransferRequested* when the remote party requests a call transfer. Refer to DivaEventTransferRequested for more information.
DivaCPT_AutoDiversion	Set	The *DivaCPT_AutoDiversion* enables the diversion of IP-based calls if requested by the called peer. By default, the call will be disconnected if the peer requests a diversion.

Passive Monitoring Call Properties

Property	Value	Definition
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.
DivaCPT_MonitorIncomingCall	Get	The property is read only and only available for call monitoring on the Diva Analog Media Boards. If the property returns true, the monitored call has been identified as incoming call based on the ring detection.

RTP Call Properties

The RTP parameters are used on Diva Media Boards to convert from TDM to RTP packets.

Property	Value	Definition
DivaCPT_RTPPayloadProtocol	Set	The property is write only and 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.
DivaCPT_RTPPayloadOptions	Set	The property is write only and defines the options to be used in the data channel. For available options and a detailed description, refer to DivaPayloadOptions.
DivaCPT_RTPMaxPacketLateRate	Set	The property is write only and and defines the maximum period of time, in milliseconds, that a packet may be delayed before it is discarded.
DivaCPT_RTPMaxDejitterDelay	Set	The property is write only and defines the maximum size of the anti-jitter buffer in milliseconds.
DivaCPT_RTPSSRC	Set	The property is write only and becomes 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.
DivaCPT_RTPLocalIpAddress	Set	The property is write only and is used to define the local IP address for RTP streaming, when call type *DivaCallTypeRTPGwMode* is enabled. When the parameter is set, the Diva SDK will check if the parameter is a valid IP address. If the string cannot be converted to an IP address, *DivaSetCallProperties* will return *DivaErrorInvalidParameter*.
DivaCPT_RTPLocalPort	Set	The property is write only and is used to define the local port for RTP streaming, when call type *DivaCallTypeRTPGwMode* is enabled.

Property	Value	Definition
DivaCPT_RTPRemoteIpAddress	Set	The property is write only and is used to define the remote IP address for RTP streaming, when call type *DivaCallTypeRTPGwMode* is enabled. When the parameter is set, the Diva SDK will check if the parameter is a valid IP address. If the string cannot be converted to an IP address, *DivaSetCallProperties* will return *DivaErrorInvalidParameter*.
DivaCPT_RTPRemotePort	Set	The property is write only and is used to define the remote port for RTP streaming, when call type *DivaCallTypeRTPGwMode* is enabled.
DivaCPT_RTPCodecMask	Set	The property is write only and is used to define the codec to be used for RTP streaming, when call type *DivaCallTypeRTPGwMode* is enabled. The options are defined in DivaCodecMask.
DivaCPT_RTPMinDejitterDelay	Set	The property is a write only and is used to specify the minimum delay to compensate packet arrival jitter, when *DivaCallTypeRTPGwMode* is enabled.

CHAPTER 7

Dialogic® Diva® API Data Structures and Defines

This chapter contains the Diva API data structures and defines.

DivaCallType

```
typedef enum
{
        DivaCallTypeVoice,
        DivaCallTypeFax,
        DivaCallTypeModem,
        DivaCallTypeDigitalData,
        DivaCallTypeX75,
        DivaCallTypeV120,
        DivaCallTypeGSM,
        DivaCallTypeVoIP,
        DivaCallTypeX25,
        DivaCallTypeSMS,
        DivaCallTypeRTPGwMode,
        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.

DivaCallTypeRTPGwMode

The call is processed as a voice call and the audio signal is send as RTP packets to the peer specified by the call properties *DivaCPT_RTPRemoteIpAddress* and *DivaCPT_RTPRemotePort*. Note that the audio is processed inside the Dialogic® Diva® System Release Software. The application cannot send or record audio in this mode. This call type is useful for applications implementing a gateway that have their own IP signaling stack.

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 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 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 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,
        DivaEventModeOSEvent,
        DivaEventModeMsgLoop,
        DivaEventModeCallbackEx,
        DivaEventModeCallbackSignal
} DivaEventModes;
```

DivaEventModeCallback

The application receives the event reported by the Diva API via a callback function. For more information on the callback function, refer to Callback function.

DivaEventModeOSEvent

The application receives the event reported by the Diva API via an event object that is signaled by the Diva API. The event object is operating-system-specific. For more information on the event object, refer to DivaGetEvent.

DivaEventModeMsgLoop

The application receives the event reported by the Diva API as a message that is sent to a message loop. For more information on this event mode, refer to Message loop.

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 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,
        DivaDRFaxSecureNotSupported,
        DivaDRFaxPartialErrors
} 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*.

DivaDRFaxSecureNotSupported

The application requested a secure fax connection by enabling *DivaCPT_FaxEnableSecurity*, but the remote peer did not support authentication. Therefore, the connection was disconnected.

DivaDRFaxPartialErrors

One or more pages of a received fax document contain errors. If a multi page document is received and at least one page was received with good quality, the Diva SDK indicates a successful reception via the event *DivaEventFaxReceived*. The disconnect reason informs the application that the document contains errors. Note that applications can retrieve the quality for each page via the call property *DivaCPT_FaxPageQuality* when the event *DivaEventFaxPageReceived* is signaled.

DivaRedirectReason

```
typedef enum
{
        DivaRedirectReasonUnknown = 0,
        DivaRedirectReasonBusy = 1,
        DivaRedirectReasonNoReply = 2,
        DivaRedirectReasonCallDeflection = 4,
        DivaRedirectReasonDTEOutOfOrder = 9,
        DivaRedirectReasonByCalledDTE = 10,
        DivaRedirectReasonUnconditional = 15
        DivaRedirectReasonUnavailable = 256,
        DivaRedirectReasonTimeOfDay,
        DivaRedirectReasonDoNotDisturb
        DivaRedirectReasonFollowMe,
        DivaRedirectReasonAway
} DivaRedirectReason;
```

DivaRedirectReasonUnknown

There is no information on the reason for the redirection reported by the remote peer.

DivaRedirectReasonBusy

The call was redirected due to a busy condition of the dialed destination.

DivaRedirectReasonNoReply

The call was redirected because the dialed destination did not answer the call.

DivaRedirectReasonCallDeflection

The call was redirected because the dialed destination has deflected the call.

DivaRedirectReasonDTEOutOfOrder

The call was redirected because the dialed destination is out of order.

DivaRedirectReasonByCalledDTE

The call was redirected by the dialed destination.

DivaRedirectReasonUnconditional

The call was redirected without a special condition.

DivaRedirectReasonUnavailable

The call has been redirected because the peer is not available. This reason is only available for IP-based line devices.

DivaRedirectReasonTimeOfDay

The call has been redirected because the peer is not available at this time. This reason is only available for IP-based line devices.

DivaRedirectReasonDoNotDisturb

The call has been redirected because the peer does not want to be disturbed. This reason is only available for IP-based line devices.

DivaRedirectReasonFollowMe

The call has been redirected to reach the dialed peer at another device. This reason is only available for IP-based line devices.

DivaRedirectReasonAway

The call has been redirected because the peer is not available. This reason is only available for IP-based line devices.

DivaSignalledCallType

```
typedef enum
{
        DivaSignalledCallTypeUnknown = 0,
        DivaSignalledCallTypeAnalog,
        DivaSignalledCallTypeDigital,
        DivaSignalledCallTypeGSM,
        DivaSignalledCallTypeFax
} 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.

DivaSignalledCallTypeGSM

The call is signaled as an asynchronous V.110 call for GSM connectivity.

DivaSignalledCallTypeFax

The call is signaled as fax call. This option may occur if an incoming SIP call contains only the media information for T.38.

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 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.
DivaErrorNoDevice	27	No Dialog® Diva® resources could be detected. There might be no resources installed or the ressources may not be configured or started correctly.

Name	Value	Description
DivaErrorInsufficientBuffer	28	The provided buffer space is not sufficient to place the requested information.
DivaErrorRejected	29	The call transfer request was rejected by the remote peer. This result code is passed with the event *DivaEventTransferCompleted*. *DivaErrorRejected* can only occur on a line device that uses SIP as a signaling protocol.
DivaErrorNoResources	30	The requested function or operation requires a resource that is not available. This may occur on Dialogic® communication platforms that have licensed components.

DivaFaxFormat

```
typedef enum
{
        DivaFaxFormatAutodetect,
        DivaFaxFormatDefault,
        DivaFaxFormatTIFF_ClassF = 1,
        DivaFaxFormatTIFF_ClassFSymmetric,
        DivaFaxFormatSFF,
        DivaFaxFormatTIFF_G3,
        DivaFaxFormatTIFF_G3Symmetric,
        DivaFaxFormatTIFF_G4,
        DivaFaxFormatTIFF_G4Symmetric,
        DivaFaxFormatColorJPEG,
        DivaFaxFormatASCII,
        DivaFaxFormatBinaryFile
} 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 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.

DivaFaxFormatBinaryFile

The data provided should be interpreted as plain binary data. This requires that the Diva SDK reported that binary file transfer is active via the call property *DivaCPT_FaxBinaryFileTransferActive*.

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 Diva API negotiates only formats up to fine.

DivaLineCodec

```
typedef enum

{

        LineAudio_ALaw      = 0,    // G.711 ALaw
        LineAudio_uLaw      = 1,    // G.711 uLaw

        /*
        * Codecs only available in direct access mode line configuration.
        */
        LineAudio_G722      = 10,   // G.722 codec
        LineAudio_G722_G711   = 11,   // G.722 - G 711 Hybrid
        LineAudio_PCM16_8KHz  = 20,   // PCM codec 16 Bit 8 KHz
        LineAudio_PCM16_16KHz = 21,   // PCM codec 16 Bit 16 KHz
        LineAudio_PCM16_32KHz = 23,   // PCM codec 16 Bit 32 KHz
        LineAudio_PCM16_48KHz = 24,   // PCM codec 16 Bit 48 KHz

} 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

LineAudio_G722

The G.722 audio codec is used if the line is configured for HD Audio using G.722. Standard voice calls will use the G.722 line and data codec. The line codec is only available on lines configured for direct access mode (no signaling).

LineAudio_G722_G711

The LineAudio_G722_G711 audio codec is only available if the line is configured in direct access mode. The audio signal will be automatically detected between G.722 and G.711 for DTMF detection, audio will be handled as G.722 like for LineAudio_G722. The line codec is only available on lines configured for direct access mode (no signaling).

LineAudio_PCM16_8KHz

The LineAudio_PCM16_8KHz audio codec is used if the line is configured for linear PCM data with 16 bit and a sample rate of 8 KHz. This requires two channels / timeslots per audio stream. Standard voice calls will use the PCM 16 data codec by default. The line codec is only available on lines configured for direct access mode (no signaling).

LineAudio_PCM16_16KHz

The LineAudio_PCM16_16KHz audio codec is used if the line is configured for linear PCM data with 16 bit and a sample rate of 16 KHz. This requires four channels / timeslots per audio stream. Standard voice calls will use the PCM 16 data codec by default. The line codec is only available on lines configured for direct access mode (no signaling).

LineAudio_PCM16_32KHz

The LineAudio_PCM16_32KHz audio codec is used if the line is configured for linear PCM data with 16 bit and a sample rate of 32 KHz. This requires eight channels / timeslots per audio stream. Standard voice calls will use the PCM 16 data codec by default. The line codec is only available on lines configured for direct access mode (no signaling).

LineAudio_PCM16_48KHz

The LineAudioPCM16_48KHz audio codec is used if the line is configured for linear PCM data with 16 bit and a sample rate of 48 KHz. This requires twelve channels / timeslots per audio stream. Standard voice calls will use the PCM 16 data codec by default. The line codec is only available on lines configured for direct access mode (no signaling).

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,
        DivaAudioFormat_G723_6_4          =  12,
        DivaAudioFormat_G723_5_3          =  13,
    /*
         * 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_Raw_GSM_610,
        DivaAudioFormat_Raw_G729,
        DivaAudioFormat_Raw_ILBC,
        DivaAudioFormat_Raw_AMR_4_75,
        DivaAudioFormat_Raw_AMR_5_15,
```

```
    DivaAudioFormat_Raw_AMR_5_9,
    DivaAudioFormat_Raw_AMR_6_7,
    DivaAudioFormat_Raw_AMR_7_4,
    DivaAudioFormat_Raw_AMR_7_95,
    DivaAudioFormat_Raw_AMR_10_2,
    DivaAudioFormat_Raw_AMR_12_2
    DivaAudioFormat_Raw_G723_6_4        = 212,
    DivaAudioFormat_Raw_G723_5_3        = 213,
    DivaAudioFormat_Raw_G722             = 214,
    DivaAudioFormat_Raw_PCM_16K16Bit    = 215,
    DivaAudioFormat_Raw_PCM_32K16Bit    = 217,
    DivaAudioFormat_Raw_PCM_48K16Bit    = 216,
} DivaAudioFormat;
```

} 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.

DivaAudioFormat_Raw_GSM_610

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_G729

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_ILBC

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_AMR_4_75

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_AMR_5_15

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_G723_6_4

The format DivaAudioFormat_G723_6_4 specifies that the file format is a wave file and the coding is according to G.723 using compression ratio 6.4 Kbits per second. 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_G723_5_3

The format DivaAudioFormat_G723_5_3 specifies that the file format is a wave file and the coding is according to G.723 using compression ratio 5.3 Kbits per second. 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_G723_6_4

The format DivaAudioFormat_Raw_G723_6_4 specifies that the file format is a raw file and the coding is according to G.723 using compression ratio 6.4 Kbits per second. 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_G723_5_3

The format DivaAudioFormat_Raw_G723_5_3 specifies that the file format is a wave file and the coding is according to G.723 using compression ratio 5.3 Kbits per second. 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_G722

The format DivaAudioFormat_Raw_G722 specifies that the file format is a raw and the coding is according to G.722. Please note that this format is only available if a call is initiated via the data codec G.722 or the line is configured for line codec G.722.

DivaAudioFormat_Raw_PCM16K16Bit

The format DivaAudioFormat_Raw_PCM16K16Bit specifies that the file format is a raw and the coding is according to PCM using 16 bit samples with a sample rate of 16 KHz. Please note that this format is only available if a call is initiated via the data codec PCM 16 with a sampling rate of 16 KHz or the line is configured for line codec PCM 16 bit 16 KHz.

DivaAudioFormat_Raw_PCM32K16Bit

The format DivaAudioFormat_Raw_PCM32K16Bit specifies that the file format is a raw and the coding is according to PCM using 16 bit samples with a sample rate of 32 KHz. Please note that this format is only available if a call is initiated via the data codec PCM 16 with a sampling rate of 32 KHz or the line is configured for line codec PCM 16 bit 32 KHz.

DivaAudioFormat_Raw_PCM48K16Bit

The format DivaAudioFormat_Raw_PCM48K16Bit specifies that the file format is a raw and the coding is according to PCM using 16 bit samples with a sample rate of 48 KHz. Please note that this format is only available if a call is initiated via the data codec PCM 16 with a sampling rate of 48 KHz or the line is configured for line codec PCM 16 bit 48 KHz.

DivaAudioFormat_Raw_AMR_5_9

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_AMR_6_7

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_AMR_7_4

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_AMR_7_95

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_AMR_10_2

The format is only available when using the function *DivaMonitorRecordAudio*.

DivaAudioFormat_Raw_AMR_12_2

The format is only available when using the function *DivaMonitorRecordAudio*.

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,
        DivaFaxOptionEnableClearChannel
} 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*.

DivaFaxOptionEnableClearChannel

If this option is set, the clear channel fax mode is enabled. By default, only T.38 is used.

DivaFaxResolution

```
typedef enum
{
DivaFaxResolutionStandard = 1,
        DivaFaxResolutionFine,
        DivaFaxResolutionSuperFine,
        DivaFaxResolutionUltraFine
} DivaFaxResolution;
```

DivaFaxDocumentProperties

```
typedef struct
{
        DWORD               Size;
        DWORD               Pages;
        DivaFaxResolution   FaxResolution;
        DWORD               ImageWidth;
        DWORD               ImageLength;
        DWORD               HorizontalResolution;
        DWORD               VerticalResolution;
        DWORD               Compression;
        DivaFaxFormat       Format;
        BOOL                AllPagesHaveSameFormat;
} DivaFaxDocumentProperties;
```

Size

The parameter *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 Diva API additional parameter may be added and the size may grow.

Pages

The parameter *Pages* contains the amount of pages included in the fax document. If the number of pages cannot be calculated the parameter is set to zero.

FaxResolution

The parameter *FaxResolution* contains the information if the resolution is standard, fine, superfine or ultrafine.

ImageWidth

The parameter *ImageWidth* contains the information about the horizontal amount of pixel per line. If the information is not available from the document header the parameter will be set to zero.

ImageLength

The parameter *ImageLength* contains the information about the amount of lines. If the document contains multiple pages, this is the length of the first page. . If the information is not available from the document header the parameter will be set to zero.

HorizontalResolution

The parameter *HorizontalResolution* contains the information about the horizontal resolution in dpi. . If the information is not available from the document header the parameter will be set to zero.

VerticallResolution

The parameter *VerticalResolution* contains the information about the vertical resolution in dpi. . If the information is not available from the document header the parameter will be set to zero.

Compression

The parameter *Compression* contains the information about the compression format. The parameter is only available for TIFF documents and will be set to zero for all other formats. The supported compression formats are RLE (2), Fax G3/T4 (3), Fax G4/T6 (4) and LZW (5). . If the information is not available from the document header the parameter will be set to zero.

FaxFormat

The parameter *FaxFormat* contains the information about the document format. This depends on the type of document (SFF or TIFF) and the compression.

AllPagesHaveSameFormat

The parameter *AllPagesHaveSameFormat* specifies if the format is the same on all pages or not.

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 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,
        DivaModemOptionDisableRetrain = 0x10,
        DivaModemOptionDisableCompression = 0x20,
        DivaModemOptionDisableV42 = 0x40,
        DivaModemOptionDisableMNP = 0x80,
        DivaModemOptionForceReliable = 0x100,
        DivaModemOptionDisableAnswerTone = 0x00001000,

        /*
        *Modem options for synchronous modems. The options following *DivaModemOptionSynchronous are
        only valid if this bit is set.
        */
        DivaModemOptionSynchronous = 0x80000000
} 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.

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.

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.

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

If this option is used in combination with the option *DivaTransferOptionCallDeflection*, the Diva SDK signals the calling party number set by the call property *DivaCPT_CallingNumber* when deflecting the call. If this option is used when transferring an incoming call via *DivaBlindCallTransfer* without the option *DivaTransferOptionCallDeflection* the calling number signaled with the incoming call is used as calling number for the implicit consultation 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
{

Tone	Value	Capability	Description
DivaEndOfTone	0x80	Detect	Indicates that a previously detected tone has ended.
DivaUnknownTone	0x81	Detect	Energy has been detected, but not a specific tone.
DivaDialTone	0x82	Detect/Generate	Standard continuous dial tone
DivaPBXInternalDialTone	0x83	Detect/Generate	Internal dial tone used by PBX. Three burst tone followed by pause.
DivaSpecialDialTone	0x84	Detect/Generate	Dial tone with interruptions, stutter dial tone.
DivaSecondDialTone	0x85	Detect/Generate	Dial tone with cadence according to TBR 21
DivaRingingTone	0x86	Detect/Generate	Standard ringback tone
DivaSpecialRingingTone	0x87	Detect/Generate	Special ringback tone. Dual burst tone followed by pause.
DivaBusyTone	0x88	Detect/Generate	Standard busy tone
DivaCongestionTone	0x89	Detect/Generate	Fast busy tone indicating congestion
DivaSpecialInformationTone	0x8A	Detect/Generate	Three tone (950 Hz, 1400 Hz, 1800 Hz) indicating that the called party cannot be reached for another reason than busy or congestion.
DivaComfortNoise	0x8B	Generate	Comfort tone indicating that the call is processed e.g. during post dial period.
DivaHoldTone	0x8C	Generate	Caller has been placed on hold
DivaRecordTone	0x8D	Generate	Tone used to indicate to the caller that recording begins
DivaCallerWaitingTone	0x8E	Generate	Called party is busy but call has been indicated that call is waiting
DivaCallWaitingTone	0x8F	Generate	Other party is calling
DivaPayTone	0x90	Generate	Reminder to deposit additional coins in a payphone
DivaPositiveIndicationTone	0x91	Generate	Supplementary service has been activated
DivaNegativeIndicationTone	0x92	Generate	Supplementary service could not be activated
DivaWarningTone	0x93	Generate	Notification that this call is being recorded
DivaIntrusionTone	0x94	Generate	Notification that this call is being monitored by an operator
DivaCallingCardServiceTone	0x95	Generate	Calling card service tone
DivaPayphoneRecognitionTone	0x96	Generate	A payphone is used for the call.
DivaCPEAlertingSignal	0x97	Generate	Dual tone alerting signal for Call waiting in PSTN environment (2130 Hz + 2750 Hz).
DivaOffHookWarningTone	0x98	Generate	Howler that may be sent if the PSTN equipment has been left off hook for an extended period of time.
DivaSITTone0	0xA0	Detect/Generate	Special information tone (three tone short 950 Hz, short 1400 Hz, short 1800 Hz)
DivaSITTone1	0xA1	Detect/Generate	Special information tone (three tone long 950 Hz, short 1400 Hz, short 1800 Hz)
DivaSITTone2	0xA2	Detect/Generate	Special information tone (three tone short 950 Hz, long 1400 Hz, short 1800 Hz)
DivaSITTone3	0xA3	Detect/Generate	Special information tone (three tone long 950 Hz, long 1400 Hz, short 1800 Hz)
DivaSITOperatorIntercept	0xA4	Detect/Generate	Special information tone for operator intercept (three tone short 950 Hz, short 1400 Hz, long 1800 Hz).

Tone	Value	Capability	Description
DivaSITVacantCircuit	0xA5	Detect/Generate	Special information tone for vacant circuit (three tone long 950 Hz, short 1400 Hz, long 1800 Hz).
DivaSITRecoder	0xA6	Detect/Generate	Special information tone for recording (three tone short 950 Hz, long 1400 Hz, long 1800 Hz)
DivaSITNoCircuitFound	0xA7	Detect/Generate	Special information tone for no circuit found (three tone long 950 Hz, long 1400 Hz, long 1800 Hz)
DivaSignalingSystem5ToneF1	0xAB	Detect / Generate	Special tone used in signaling systems according to System 5.
DivaSignalingSystem5ToneF2	0xAC	Detect / Generate	Special tone used in signaling systems according to System 5.
DivaSignalingSystem5ToneF1F2	0xAD	Detect / Generate	Special tone used in signaling systems according to System 5.
DivaInterceptTone	0xAF	Generate	Intercept tone
DivaModemCallingTone	0xC0	Detect/Generate	Standard modem calling tone (1300 Hz) indicating outbound modem calls
DivaFaxCallingTone	0xC1	Detect/Generate	Standard fax calling tone (1100 Hz) indicating outbound fax calls
// These tones can be detected but not generated.			
DivaAnswerTone	0xC2	Detect/Generate	Modem or fax answer tone (2100 Hz) used to disable echo suppressors in the PSTN.
DivaAnswerTonePhaseReversal	0xC3	Generate	Modem answer tone (2100 Hz) with regular phase reversals to disable echo suppressors and echo cancellers.
DivaANSam	0xC4	Detect/Generate	Modified answer tone according to ITU-T V.8 (modulated 2100 Hz) indicating V.8 capability.
DivaANSamPhaseReversal	0xC5	Generate	Modified answer tone according to ITU-T V.8 (modulated 2100 Hz) with phase reversals.
DivaBell103AnswerTone	0xC6	Detect/Generate	Continuous 2225 Hz answer tone used with Bell 103 modulation.
DivaFaxFlags	0xC7	Detect/Generate	HDLC flag sequences in modulation scheme ITU-T V.21 channel 2 of a Fax Preamble have been detected.
DivaFaxG2GroupId	0xC8	Detect/Generate	1850 Hz tone of group 2 FAX machines indicating the ability to receive.
DivaHumanSpeech	0xC9	Detect	The signal has been detected as a human talker.
DivaAnsweringMachineTone	0xCA	Detect/Generate	Answering machine tone 390Hz.
DivaToneAlertingSignal	0xCB	Detect/Generate	Dual tone alerting signal for Caller ID in PSTN environment (2130 Hz + 2750 Hz).

} DivaContinuousTones;

DivaMultiFrequencyTones

```
typedef enum
{
        DivaToneMF1 = 0xF1,
        DivaToneMF2,
        DivaToneMF3,
        DivaToneMF4,
        DivaToneMF5,
        DivaToneMF6,
        DivaToneMF7,
        DivaToneMF8,
        DivaToneMF9,
        DivaToneMF0,
        DivaToneMFStart = 0xFD,
        DivaToneMFStop = 0xFF
} DivaMultiFrequencyTones;
```

DivaR2Tones

The Diva SDK supports the detection and generation of R2 forward and backward tones. The following tones are supported.

```
typedef enum
{
        DivaR2ForwardToneOff        = 0xD0,
        DivaR2ForwardTone1          = 0xD1,
        DivaR2ForwardTone2          = 0xD2,
        DivaR2ForwardTone3          = 0xD3,
        DivaR2ForwardTone4          = 0xD4,
        DivaR2ForwardTone5          = 0xD5,
        DivaR2ForwardTone6          = 0xD6,
        DivaR2ForwardTone7          = 0xD7,
        DivaR2ForwardTone8          = 0xD8,
        DivaR2ForwardTone9          = 0xD9,
        DivaR2ForwardTone10         = 0xDA,
        DivaR2ForwardTone11         = 0xDB,
        DivaR2ForwardTone12         = 0xDC,
        DivaR2ForwardTone13         = 0xDD,
        DivaR2ForwardTone14         = 0xDE,
        DivaR2ForwardTone15         = 0xDF,
        DivaR2BackwardToneOff       = 0xB0,
        DivaR2BackwardTone1         = 0xB1,
        DivaR2BackwardTone2         = 0xB2,
        DivaR2BackwardTone3         = 0xB3,
        DivaR2BackwardTone4         = 0xB4,
        DivaR2BackwardTone5         = 0xB5,
        DivaR2BackwardTone6         = 0xB6,
        DivaR2BackwardTone7         = 0xB7,
        DivaR2BackwardTone8         = 0xB8,
        DivaR2BackwardTone9         = 0xB9,
        DivaR2BackwardTone10        = 0xBA,
        DivaR2BackwardTone11        = 0xBB,
        DivaR2BackwardTone12        = 0xBC,
        DivaR2BackwardTone13        = 0xBD,
        DivaR2BackwardTone14        = 0xBE,
        DivaR2BackwardTone15        = 0xBF
} DivaR2Tones;
```

DivaToneDefinition

Via *DivaToneDefinition*, a single or dual tone with a specific duration is specified; optionally, a pause can be specified as well. Multiple definitions can be combined to a cadence.

```
typedef struct
{
    DWORD    Frequency1;
    DWORD    Frequency1Variation;
    DWORD    Frequency2;
    DWORD    Frequency2Variation;
    DWORD    Duration;
    DWORD    DurationVariation;
    DWORD    Pause;
    DWORD    PauseVariation
} DivaAnalogParams;
```

Frequency1

Frequency1 defines the frequency (in Hz) for a single tone or the first frequency of a dual tone.

Frequency1Variation

Frequency1Variation defines the variation (in Hz) for the frequency specified by *Frequency1*.

Frequency2

Frequency2 defines the frequency (in Hz) for the second frequency of a dual tone. If the tone definition refers to a single tone, *Frequency2* must be set to zero.

Frequency2Variation

Frequency2Variation defines the variation (in Hz) for the frequency specified by *Frequency2*.

Duration

Duration defines the time (in milliseconds) that the tone must be available. The maximum duration for a tone is 8000 milliseconds, including the variation.

DurationVariation

DurationVariation defines the variation (in milliseconds) for the time specified by *Duration*.

Pause

Pause defines the expected time of silence after the tone. The time must be specified in milliseconds. The maximum pause is 8000 milliseconds, including the variation. If no silence is expected after the tone, the parameter is set to zero.

PauseVariation

PauseVariation defines the variation (in milliseconds) for the time specified by *Pause*.

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 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® Media Board. This mode does not require any conference features 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 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.

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 Dialogic® Diva® API Call Properties.

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 Dialogic® Diva® API Call Properties.

```
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          =39,
        ConnNormV23ETS300659_1          =40,
        ConnNormBell202CID              =42,
        ConnNormV21Bits10               =43,
        ConnNormBell202POS              =44,
        ConnNormV23Reverse              =46,
        ConnNormFSK                     =47,
        ConnNormECall                   =59,
        ConnNormBell103SIA              =45,
        ConnNormV22FastSetup            =72,
        ConnNormV22bisFastSetup         =73,
        ConnNormV29FastSetup            =74
} DivaConnectedNorm;
```

DivaMonitorSource

```
typedef enum
{
        DivaMonitorSourceUnknown = 0,
        DivaMonitorSourceOriginator,
        DivaMonitorSourceAnswerer,
        DivaMonitorDeviceAToDeviceB,
        DivaMonitorDeviceBToDeviceA,
        DivaMonitorSourceBoth,
        DivaMonitorSourceMixed
} DivaMonitorSource;
```

DivaMonitorSourceUnknown

For DTMF or tone detection on analog lines the value DivaMonitorSourceUnknown will be returned in DivaMonitorDTMFInfo or DivaMonitorToneInfo. This value cannot be used by the application for recording functions.

DivaMonitorSourceOriginator

The audio streamed from the originator of a call is recorded.

DivaMonitorSourceAnswerer

The audio streamed from the answerer of a call is recorded.

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.

DivaMonitorSourceBoth

The audio streamed from both parties is recorded to an audio file with two channels.

DivaMonitorSourceMixed

The audio from both parties is mixed and recorded to an audio file with one channel.

DivaMonitorStatus

```
typedef enum
{
        DivaMonitorStarted = 1,
        DivaMonitorStopped,
        DivaMonitorErrorStarting,
        DivaMonitorLayer1Up,
        DivaMonitorLayer1Down
} DivaMonitorStatus;
```

DivaMonitorDTMFInfo

```
typedef struct

{

        char                Digit;
        DWORD               Device;
        DivaMonitorSource   Source,

} DivaMonitorDTMFInfo;
```

Digit

The parameter Digit contains the received DTMF digits. Valid digits are '0123456789ABCD*#XY'. The value 'X' reports that a fax calling tone has been detected, the value 'Y' reports that a fax answer tone has been detected.

Device

The parameter Device contains the information on which Diva SDK line device the digit was detected. If the monitoring object was created via DivaCreateMonitorAudio, the Diva SDK has no information on the signaling direction. In such cases the application may calculate if the originator or answerer of the call has sent the digit from the line device information.

Source

The parameter Source contains the information which side has sent the digit. Valid options are DivaMonitorSourceOriginator, DivaMonitorSourceAnswerer or DivaMonitorSourceUnknown.

DivaMonitorToneInfo

```
typedef struct
{
   DWORD            Tone;
   DWORD            Device;
   DivaMonitorSource   Source,
} DivaMonitorToneInfo;
```

Tone

The parameter Tone contains the received tone digits. For valid tone refer to DivaContinuousTones.

Device

The parameter Device contains the information on which Diva SDK line device the tone was detected. If the monitoring object was created via DivaCreateMonitorAudio, the Diva SDK has no information on the signaling direction. In such cases the application may calculate if the originator or answerer of the call has sent the tone from the line device information.

Source

The parameter Source contains the information which side has sent the tone. Valid options are DivaMonitorSourceOriginator, DivaMonitorSourceAnswerer or DivaMonitorSourceUnknown.

DivaMonitorR2Variants

```
typedef enum
{
        DivaMonitorR2Basic = 1,
        DivaMonitorR2India,
        DivaMonitorR2Brazil,
        DivaMonitorR2Mexico
} DivaMonitorR2Variants;
```

DivaMonitorR2Basic

If this variant is selected, only signaling information is extracted. In this mode no called or calling number information will be available.

DivaMonitorR2India

If the R2 variant *DivaMonitorR2India* is selected, the inband information for called and calling party number is extracted based on the Indian R2 protocol.

DivaMonitorR2Brazil

If the R2 variant *DivaMonitorR2Brazil* is selected, the inband information for called and calling party number is extracted based on the Brazilian R2 protocol.

DivaMonitorR2Mexico

If the R2 variant *DivaMonitorR2Mexico* is selected, the inband information for called and calling party number is extracted based on the Mexican R2 protocol.

DivaMonitorFrameReportMode

The *DivaMonitorFrameReportMode* specifies which type of frames will be reported to the application.

```
typedef enum
{
        DivaMonitorFrameReportModeNone,
        DivaMonitorFrameReportModeLayer3,
        DivaMonitorFrameReportModeLayer2
} DivaMonitorFrameReportMode;
```

DivaMonitorFrameReportModeNone

If the option *DivaMonitorFrameReportModeNone* is specified, no frames are indicated to the application. This is the default mode. Note that high level call progress events are still signaled.

DivaMonitorFrameReportModeLayer3

If the option *DivaMonitorFrameReportModeLayer3* is specified, received layer 3 frames are indicated to the application.

DivaMonitorFrameReportModeLayer2

If the option *DivaMonitorFrameReportModeLayer2* is specified, received layer 2 frames are indicated to the application. This includes pure layer 2 frames and layer 2 frames with layer 3 content.

DivaMonitorOptions

The *DivaMonitorOptions* allow an application to overwrite the default behavior of the monitoring on analog media boards.

```
typedef enum
{
        DivaMonitorOptionPolarity = 0x0001,
        DivaMonitorOptionConnectOnEnergy = 0x0002,
        DivaMonitorOptionDisableCPAOnConnect = 0x0004,
        DivaMonitorOptionDisableCPAOnDisconnect = 0x0008,
        DivaMonitorOptionDisableStandardTones = 0x0010,
        DivaMonitorOptionDisableFSK = 0x0020,
        DivaMonitorOptionDisableCPA = 0x0080,
        DivaMonitorOptionDisableDigitProcessing = 0x0100
} DivaMonitorOptions;
```

DivaMonitorOptionPolarity

If the option *DivaMonitorOptionPolarity* is specified, the connect and disconnect detection is done based on polarity reversal detection. The options *DivaMonitorOptionConnectOnEnergy*, *DivaMonitorOptionDisableConnectDetect*, and *DivaMonitorOptionDisableDisconnectDetect* are ignored if this option is set.

DivaMonitorOptionConnectOnEnergy

By default, the Dialogic® Diva® SDK will detect the connect based on human talker detection and if a tone outside the range for signaling tones is detected. If the option *DivaMonitorOptionConnectOnEnergy* is set, any energy on the line is interpreted as connect. The energy level can be specified via *DivaMonitorAnalogParams*.

DivaMonitorOptionDisableCPAOnConnect

If the option *DivaMonitorOptionDisableCPAOnConnect* is set, the Diva SDK will not do any call progress analyses for detecting the connect. Applications that want to record the initial phase including ringing may use this option. The Diva SDK will report the connect once the on hook of the parties is detected.

DivaMonitorOptionDisableCPAOnDisconnect

If the option *DivaMonitorOptionDisableCPAOnDisconnect* is set, the Diva SDK will not do any call progress analyses for detecting the disconnect. Applications that want to record the whole call including busy tones may use this option. The Diva SDK will report the disconnect once the off hook of the parties is detected.

DivaMonitorOptionDisableFSK

By default, the Diva SDK will detect FSK data between the rings to retrieve calling party information. If the option *DivaMonitorOptionDisableFSK* is set, the FSK detection is disabled and the event *DivaEventMinitorCallInitiated* is reported with the first ring.

DivaMonitorOptionDisableStandardTones

By default, the Diva SDK will use standard definitions for ring and busy tones. The application may add definitions of custom tones and cadences. If the application wants to use only the custom tones, the option *DivaMonitorOptionDisableStandardTones* can be used to disable the standard tones.

DivaMonitorOptionDisableCPA

Applications that get signaling information via a 3rd party source, e.g., via CSTA, disable the CPA via the option *DivaMonitorOptionDisableCPA*. If this option is set, all other options are ignored. The application attaches to the analog lines for recording via *DivaMonitorAttachToLine*.

DivaMonitorOptionDisableDigitProcessing

The *DivaMonitorOptionDisableDigitProcessing* option disables the processing of DTMF separator digits to detect called and calling numbers. By default, the Diva SDK checks whether *DivaMonitorAnalogParams* is available. Then it detects the called number and calling number (ANI / DNIS), and stores them into separate call properties.

DivaMonitorAnalogParams

```
typedef struct
{
        DWORD           Size;
        int             RecordingGain;
        DWORD           ConnectEnergy;
        DWORD           ConnectSilenceTimeout;
        DivaAudioFormat DefaultAudioFormat;
} DivaMonitorAnalogParams;
```

Size

Size defines the length of the data structure. The application sets this value before calling any Diva API function that gets the structure as parameter. The *Size* is available for compatibility reasons. Depending on the version of the Dialogic® Diva® SDK, members may be added and indicated by a larger size.

RecordingGain

RecordingGain defines the energy (in dBm) to increase or decrease the volume for recording.

ConnectEnergy

ConnectEnergy defines the energy (in dBm) that is interpreted as connect event. The energy is given in the range of -127 to 127 dBm.

ConnectSilenceTimeout

ConnectSilenceTimeout defines the maximum amount of silence (in milliseconds) after the last ringing tone before a connect is indicated. A value of zero disables the silence timeout and the connection is detected based on a human talker detection or energy detection. The default value is zero.

DefaultAudioFormat

DefaultAudioFormat defines the audio format initially used when the data channel is established for call progress analyses. The Diva SDK may store data during the connect detection until detectors trigger human talker and uses the data when recording is initiated by the application. If the format given with the start of the recording does not match the default format, this stored data cannot be used.

DivaMonitorT1CASVariants

```
typedef enum
{
        DivaMonitorT1CASLoopStart = 1,
        DivaMonitorT1CASGroundStart,
        DivaMonitorT1CASWinkStart
} DivaMonitorT1CASVariants;
```

DivaMonitorT1CASLoopStart

This option configures the monitoring object for monitoring T1 CAS lines with the trunk type "Loop Start" and the signaling mode FXS/FXO.

DivaMonitorT1CASGroundStart

This option configures the monitoring object for monitoring T1 CAS lines with the trunk type "Ground Start" and the signaling mode FXS/FXO.

DivaMonitorT1CASWinkStart

This option configures the monitoring object for monitoring T1 CAS lines with the trunk type "Wink Start".

DivaMonitorT1CASParams

```
typedef struct
{
    DWORD           Size;
    int             RecordingGain;
    DWORD           ConnectSilenceTimeout;
    DWORD           DigitCountToSignalCall;
} DivaMonitorT1CASParams;
```

Size

Size defines the length of the data structure. The application sets this value before calling any Diva API function that gets the structure as parameter. The *Size* is available for compatibility reasons. Depending on the version of the Diva SDK members may be added and indicated by a larger size.

RecordingGain

RecordingGain defines the energy in dBm to increase or decrease the volume for recording.

ConnectSilenceTimeout

ConnectSilenceTimeout defines the maximum amount of silence in milliseconds after the last ringing tone before a connect is indicated. A value of zero disables the silence timeout and the connection is detected based on a human talker detection or energy detection. The default value is 4000 milliseconds.

DigitCountToSignalCall

DigitCountToSignalCall defines the amount of dial digits to be received before indicating the call to the application. By default, the call is indicated when the ring is detected.

DivaTime

The 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 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
{
        DivaSignalServiceUnknown = 0,
        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 or the service indicated for an incoming call.

DivaSignaServiceUnknown

The service of an incoming call is not reported by the network. This value cannot be used 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 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,
       DivaDSP_LineDeviceStatus,

        DivaDST_RedAlarm,
        DivaDST_BlueAlarm,
        DivaDST_YellowAlarm,

        DivaDST_ActiveInConnections,
        DivaDST_ActiveOutConnections,

        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_LineDeviceStatus

The type DivaDST_LineDeviceStatus provides the information if the device is enabled or disabled. Values are defined in DivaLineDeviceState.

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_ActiveInConnections

The amount of incoming calls that have been successfully connected since the last restart of the line device. The data type is DWORD.

DivaDST_ActiveOutConnections

The amount of outgoing calls that have been successfully connected since the last restart of the line device. 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 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,
        DivaPotsLineToneAlertingSignal,
        DivaPotsLineEndofCall,
        DivaPotsLineFlashDetected,
        DivaPotsLineInUse
} 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

The line is connected to the switch and no call is in progress. This is the idle state.

DivaPotsLineRing

A ring is detected at the line.

DivaPotsLinePolarityReverse

Reverse polarity has been detected on the line.

DivaPotsLineToneAlertingSignal

A Dual Tone Alerting signal (2130 Hz + 2750 Hz) has been detected on the idle line. It should be followed by the caller ID information.

DivaPotsLineEndofCall

A gap in the current loop has been detected in off hook state. This is usually the indication of the switch that the peer hung up.

DivaPotsLineFlashDetected

A hook flash from another device on the line has been detected in idle state.

DivaPotsLineInUse

In idle state, it has been detected that another device on the line went off hook.

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.

DivaLineDeviceState

```
typedef enum

{

        DivaLineDeviceStateEnabled = 0,
        DivaLineDeviceStateDisabled,

} DivaLineDeviceState;
```

DivaLineDeviceStateEnabled

The state DivaLineDeviceStateEnabled specifies that the line device is properly installed and driver software is started.

DivaLineDeviceStateDisabled

The state DivaLineDeviceStateDisabled specifies that the line device is either physically removed or the driver software for the device is not loaded.

DivaDeviceStatisticsType

```
typedef enum
{
        DivaST_Layer1Statistics = 1,
} DivaDeviceStatisticsType;
```

DivaDeviceStatisticsType specify the Line Device statistics type enumeration.

DivaST_Layer1Statistics

The layer 1 statistics is the only enumeration available; layer 1 statistics are only applicable to T1 and E1 Line Device types.

DivaLayer1Statistics[1]

```
typedef struct

{
```

```
    unsigned int        FramingErrors;
    unsigned int        CodeViolations;
    unsigned int        Crc4Errors;
    unsigned int        FrameSlips;
    unsigned int        InternalErrors;
    unsigned int        DmaOverloads;
    unsigned int        SampleOverruns;
    unsigned int        ProcessingOverloads;
    unsigned int        Load90Percent;
    unsigned int        Load80Percent;
    unsigned int        Load70Percent;
    unsigned int        Load60Percent;
    unsigned int        Load50Percent;
    unsigned int        PeakProcessingLoad;
} DivaLayer1Statistics;
```

DivaLayer1Statistics defines the counters available for the DivaST_Layer1Statistics enumeration in DivaDeviceStatisticsType.

FramingErrors

This counter increments each time a Framing error is detected in the T1 or E1 Line Device since the last clear; note that this counter is not available on Analog or BRI Line Devices.

CodeViolations

This counter increments each time a Code Violations is detected in the T1 or E1 Line Device since the last clear; note that this counter is not available on Analog or BRI Line Devices.

Note: Code Violations keeps track of either Bipolar Violations (BPV) or Excessive Zeros (EXZ) errors.

Crc4Errors

This counter increments each time a CRC-4 error is detected in the T1 or E1 Line Device since the last clear; note that this counter is not available on Analog or BRI Line Devices.

InternalErrors
DmaOverloads
ProcessingOverloads

These counters are normally zero as relate to a DMA engine which is not applicable to most Diva media Boards.

SampleOverruns

This counter is incremented each time the sample buffer between the TDM and the signal processing function was not able to compensate for CPU scheduling or processing backlog. Each increment means that a buffer of samples slipped which could cause a discontinuity or clicks in the audio stream. This counter is available on all Media boards and it might be useful in troubleshooting voice quality issues. Note that the value is only meaningful in the context of the first device Id of a physical board.

This entry keeps track of the number of internal CPU processing errors detected since the last clear; the counter is mostly used for troubleshooting purposes. Note that the value is only meaningful for the first device Id of a physical board.

1. These data structures along the DivaGetLineDeviceStatistic( ) are used in retrieving Layer 1 statistics functionality, which is not necessarily available in Windows

LoadXXPercent

This counter keeps track of the number of times the internal CPU processing load reached that XX level, or higher, since the last clear; the counter is useful in troubleshooting potential CPU overloads. Note that the value is only meaningful for the first device Id of a physical board.

PeakProcessingLoad;

This entry stores the actual peak CPU load measured; note that its resolution is 6%, thus the only values seen are 6, 12, 18, 24, 30, etc. Like the other internal counters, this value is only meaningful for the first device Id of a physical board.

DivaDeviceStatisticsValue

```
typedef union
{
    DivaLayer1Statistics        Layer1Statistics;
} DivaDeviceStatisticsValue;
```

DivaDeviceStatusEvents

```
typedef enum
{
        DivaDSE_Layer1,         = 0x00000001
        DivaDSE_Layer2,         = 0x00000002,
        DivaDSE_Alarms          = 0x00000004,
        DivaDSE_ServiceState    = 0x00000080,
} 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.

DivaDSE_ServiceState

The service state of the device has changed from enabled to disabled or vice versa.

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 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* parameter defines the length of the structure. The application sets this value before calling any function that gets this structure as parameter. Depending on the 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_L
                            EN];
    DivaTime                Time;

} DivaMWIDeactivateParams;
```

DivaMWIIndicationParams

```
typedef struct
{
    DWORD                   SizeUsed;
    DWORD                   Line;
    DWORD                   Handle;
    DWORD                   Service;
    DWORD                   NumMessages;
    DivaMessageStatus       Status;
    DWORD                   Reference;
    char                    ControllingUserNumber[MAX_ADDR_LEN];
    char                    ControllingUserProvidedNumber[MAX_ADDR_LEN];
    DivaTime                Time;
    char                    CalledNumber[MAX_ADDR_LEN];

} DivaMWIIndicationParams;
```

SizeUsed

The *SizeUsed* parameter defines the length of data written by the Diva SDK.

Line

The *Line* parameter identifies the line device on which the information was received. Line devices are continuously numbered by an index starting with one.

Handle

The *Handle* parameter is reserved for future use.

Service

The *Service* parameter specifies the service or media type of the message, e.g. voice or fax.

NumMessages

The *NumMessage* parameter pecifies the amount of messages that are waiting.

Status

Specifies whether the message was added or removed. For options see DivaMessageStatus.

Reference

The *Reference* parameter is only valid if *Status* is not set to *DivaMessageUnknown*.

ControllingUserNumber

The *ControllingUserNumber* parameter reports the number of the user that controlled the message waiting indication.

ControllingUserProvidedNumber

The *ControllingUserProvidedNumber* parameter reports the number used for authentication.

Time

The *Time* parameter specifies the time when the indication was created.

CalledNumber

The *CalledNumber* parameter is reserved for future use.

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. Note that the event is signalled when the remote peer has been identified as answering machine. At this time, the announcement of the answering machine may still be streamed.

DivaResultAnsweringMachineTone

The standard answering machine tone of 390 Hz has been detected. Note that the tone may still be active when the event is signaled.

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.

DivaTransferRejectReasons

The *DivaTransferRejectReasons* specify the reasons for a transfer reject. The values can be specified in a call to *DivaRejectTransfer* and retrieved if the application is the initiator of a transfer and receives the reject.

```
typedef enum
{
        DivaTRRUnknown = 0,
        DivaTRRBadRequest,
        DivaTRRUnauthorized,
        DivaTRRNotAcceptable,
        DivaTRRUserDefined
} DivaTransferRejectReasons;
```

DivaTRRUnknown

The transfer reject reason is unknown.

DivaTRRBadRequest

The transfer request was rejected by the remote peer with the reason "400 Bad Request".

DivaTRRUnauthorized

The transfer request was rejected by the remote peer with the reason "401 Unauthorized".

DivaTRRNotAcceptable

The transfer request was not accepted by the remote peer.

DivaInitParameterTypes

The enumeration *DivaInitParameterTypes* defines the parameter that can be set by *DivaSetInitParameter*. The following table specifies the types and describes the functionality.

Type Name	Data Type	Description
DivaParamNumIPDevices	DWORD	Specifies the maximum number of IP devices to be configured. Default is one. If set to zero, no IP devices will be created. The maximum number of devices that can be configured is eight.
DivaParamDisableTDMDevices	Boolean	If this parameter is set, the Dialogic® Diva® SDK will not use any TDM-based Diva Media Boards. Only software-based IP communication will be possible.
DivaParamDisableConfActiveTalker	Boolean	If set, active talker detection for conferences is disabled. By default, the active talker detection is enabled.
DivaParamConfActiveTalkerNotifyInterval	DWORD	Specifies the interval for notification of active talker on a conference.

DivaDeviceInitParameterTypes

The enumeration *DivaDeviceInitParameterTypes* defines the parameter that can be set by *DivaSetDeviceInitParameter*. The following table specifies the types and describes the functionality.

Type Name	Data Type	Descrption
DivaDeviceParamNetwork Interface	ASCII String	The parameter defines the local IP address. It is planned that in future versions also the network interface name can be specified. If not specified, the default local IP address is used. The maximum length is MAX_ADDR_LEN.
DivaDeviceParamMaxChannels	DWORD	Defines the maximum amount of channels to be used for this device. Note that the amount of licensed channels must be equal or greater than the amount of all configured channels of all devices.
DivaDeviceParamSignaling Protocol	DivaSignalingProtocols	Specifies the signaling protocol to be used. Currently, the following options are available: • DivaProtocolH323 • DivaProtocolSIP The default value is DivaProtocolSIP.
DivaDeviceParamSignalingPort	DWORD	Specifies the port to be used for signaling. The default value depends on the signaling protocol.
DivaDeviceParamSignaling Transport	DivaTransport Protocols	Specifies the signaling transport protocol to be used. Currently, the following options are available: • DivaTransportUDP • DivaTransportTCP
DivaDeviceParamMaxRegistrations	DWORD	Specifies the maximum registrations at SIP registrar servers. The parameter is valid for SIP only. By default, one registration is assumed. The type of the parameter is DWORD.
DivaDeviceParamProxy	ASCII String	Specifies the SIP proxy in the format {<IP address> | <DNS Name> }[:<port>]. The parameter is only valid if the signaling protocol is set to SIP. By default, no proxy is used. The type of the parameter is a zero terminated string.
DivaDeviceParamProxyUser	ASCII String	Specifies the user name to be used for authentication at the SIP proxy. The parameter is only valid if the signaling protocol is set to SIP. The type of the parameter is a zero terminated string.

Type Name	Data Type	Descrption
DivaDeviceParamProxyPassword	ASCII String	Specifies the password to be used for authentication at the SIP proxy. The parameter is only valid if the signaling protocol is set to SIP. The type of the parameter is a zero terminated string.
DivaDeviceParamProxyRealm	ASCII String	Specifies the realm/domain to be used for authentication at the SIP proxy. The parameter is only valid if the signaling protocol is set to SIP. The type of the parameter is a zero terminated string.
DivaDeviceParamEnableTranspAddr	Boolean	If enabled, the Dialogic® Diva® SDK will signal the received SIP or H.323 addresses without changes, unless a mapping exists. By default, the Diva SDK tries to extract a phone number from the SIP or H.323 address.
DivaDeviceParamDefaultFromAddress	ASCII String	If specified, the address will be used as default FROM address if no calling number has been specified when a call is initiated. The parameter is only valid for SIP-based line devices.

DivaSIPRegistrarParams

```
typedef struct
{
    DWORD    Size;
    char     RegistrarAddress[MAX_ADDR_LEN];
    char     DisplayName[MAX_ADDR_LEN];
    char     Domain[MAX_ADDR_LEN];
    char     User[MAX_ADDR_LEN];
    char     AuthorizationRealm[MAX_ADDR_LEN];
    char     AuthorizationUser[MAX_ADDR_LEN];
    char     Password[MAX_ADDR_LEN];
    DWORD    RefreshInterval;
} DivaSIPRegistrarParams;
```

Size

Size defines the length of the data structure passed by the application. The *Size* parameter allows for extending the data structure with future Dialogic® Diva® SDK versions, e.g., for security parameter. The Diva SDK will only use access fields in the data structure that are within the given size.

RegistrarAddress

RegistrarAddress is a mandatory parameter that contains the SIP registrar address as IP address or DNS name. Optionally, a port number can be added. The format is:
{ <IP address> | <DNS name> }[:<port number>], e.g., 192.168.0.100 or myregistrar.com:5060.

DisplayName

DisplayName is an optional parameter sent to the registrar server during registration.

Domain

Domain is a mandatory parameter containing the network part of the public SIP address.

User

User is a mandatory parameter containing the user part of the public SIP address.

AuthorizationRealm

AuthorizationRealm is an optional parameter that is only required if the SIP registrar uses a realm for authentication requests that is different from the *Domain*.

AuthorizationUser

AuthorizationUser is an optional parameter that is only required if the SIP registrar uses a different user name than specified as public user name for the authorization.

Password

Password is an optional parameter used for authentication at the SIP registrar. The parameter is required if the SIP registrar requests authentication.

RefreshInterval

The *RefreshInterval* parameter specifies the amount of seconds until a refresh of the registration is required.

DivaH323GatekeeperParams

typedef struct

{

```
DWORD                Size;
char                 GatekeeperAddress[MAX_ADDR_LEN];
DivaH323EndpointType LocalType;
char                 Number[MAX_ADDR_LEN];
char                 Aliases[MAX_ADDR_LEN];
char                 Prefixes[MAX_ADDR_LEN];
char                 Name[MAX_ADDR_LEN];
DWORD                RefreshInterval;
```

} DivaH323GatekeeperParams;

Size

The *Size* defines the length of the data structure passed by the application. The *Size* parameter allows to extend the data structure with future Dialogic® Diva® SDK versions, e.g., for security parameter. The Diva SDK will only use access fields in the data structure that are within the given size.

GatekeeperAddress

The *GatekeeperAddress* is a mandatory parameter that contains the IP address of the gatekeeper. Optionally, a port number can be added. The format is: <IP address> [:<port number>], e.g., 192.168.0.100:1010.

LocalType

The *LocalType* parameter specifies the type of the local endpoint. The parameter is of type *DivaH323EndpointType*, valid values are *DivaH323EndpointTerminal* and *DivaH323EndpointGateway*.

Number

The *Number* parameter contains one phone number or a comma separated list of phone numbers to be registered at the gatekeeper.

Aliases

The *Aliases* parameter contains one H.323 alias or a comma separated list of H.323 aliases to be registered at the gatekeeper.

Prefixes

The *Prefixes* parameter contains one or more comma separated prefixes to be registered at the gatekeeper.

Name

The *Name* parameter is optional and defines a symbolic name used to display at the endpoint.

RefreshInterval

The *RefreshInterval* parameter specifies the amount of seconds until a refresh of the registration is required.

DivaH323EndpointType

```
typedef enum
{
        DivaH323EndpointTerminal = 1,
        DivaH323EndpointGateway
} DivaH323EndpointType;
```

DivaH323EndpointTerminal

The value *DivaH323EndpointTerminal* specifies that the H.323 endpoint type is terminal.

DivaH323EndpointGateway

The value *DivaH323EndpointGateway* specifies that the H.323 endpoint type is gateway.

DivaRegistrationStatus

```
typedef enum
{
        DivaRegStatusIdle,
        DivaRegStatusInitiated,
        DivaRegStatusRegistered,
        DivaRegStatusFailed,
        DivaRegStatusReleasing,
        DivaRegStatusReleased
} DivaRegistrationStatus;
```

DivaRegStatusIdle

This is the initial state before the registration request has been sent.

DivaRegStatusInitiated

The registration request has been successfully sent. The response from the peer is pending.

DivaRegStatusRegistered

The registration has been confirmed by the peer.

DivaRegStatusFailed

The registration has been rejected by the peer.

DivaRegStatusReleasing

The release of a previously successful or pending registration has been initiated.

DivaRegStatusReleased

A previously successful registration has been released by the peer or has timed out.

DivaRegistrationResults

```
typedef enum
{
        DivaRegResultSuccess = 0,
        DivaRegResultNetworkError,
        DivaRegResultTimeout,
        DivaRegResultUnavailable,
        DivaRegResultUnknownUser,
        DivaRegResultPermissionDenied,
        DivaRegResultInvalidParameter,
        DivaRegResultRejected
} DivaRegistrationResults;
```

DivaRegResultSuccess

The registration was successful.

DivaRegResultNetworkError

The registration failed due to a network error.

DivaRegResultTimeout

The registration timed out. The timeout values can not be set. SIP and H.323 have different timeout values.

DivaRegResultUnavailable

The registrar server or gatekeeper was not available.

DivaRegResultUnknownUser

The registration failed because the user name given for authentication is unknown.

DivaRegResultPermissionDenied

The registration failed because the user could not be authenticated.

DivaRegResultInvalidParameter

The registration failed because the parameter, e.g., the address, is rejected by the peer.

DivaRegResultRejected

The registration was rejected by the registrar server or gatekeeper.

DivaCodec

The enumeration *DivaCodec* defines the list of predefined codecs. For these codecs, the Dialogic® Diva® SDK automatically adds information like payload and default frame size.

DivaCodec_G711_ALaw	Enables G.711 a-law.
DivaCodec_G711_ULaw	Enables G.711 μ-law.
DivaCodec_T38	Enables T.38 fax.
DivaCodec_G723	Enables all rates for G.723 codecs.
DivaCodec_G723_5_3	Enables G.723 at 5.3 kbps.
DivaCodec_G723_6_3	Enables G.723 at 6.3 kbps.
DivaCodec_G726	Enables all rates for the G.726 codecs.
DivaCodec_G726_16	Enables G.726 at 16 kbps.
DivaCodec_G726_24	Enables G.726 at 24 kbps.
DivaCodec_G726_32	Enables G.726 at 32 kbps.
DivaCodec_G726_40	Enables G.726 at 40 kbps.
DivaCodec_G729	G.729 at 8 kbps.
DivaCodec_G729A	G.729 with annex A at 8 kbps.
DivaCodec_G729B	G.729 with silence suppression.
DivaCodec_G729AB	G.729 with annex A at 8 kbps with silence suppression.
DivaCodec_GSM_FULL	GSM with full speech rate.
DivaCodec_GSM_HALF	GSM with half speech rate.
DivaCodec_GSM_EFR	GSM with enhanced full rate.
DivaCodec_GSM_AMR	GSM with adaptive multirate.

DivaDTMFMode

DivaDTMFMode is used to specify the DTMF handling on IP-based calls and to retrieve the negotiated mode.

DTMF Mode	Description
DivaDtmfModeAuto	Uses the automatic negotiation of the DTMF mode. The options RFC 2833 and inband are enabled. Based on the negotiation with the peer, RFC2833 or inband is used. If DTMF cannot be sent via RFC2833 or inband, then SIP info messages are used. Received DTMF are signaled to the application independent from the way they are received.
DivaDtmfModeRFC2833	Use RFC2833 if supported by the remote peer. An implicit fallback to inband DTMF mode is done if RFC2833 is not supported by the remote peer.
DivaDtmfModeSIPInfo	Only valid for transmitted DTMF tones. If this mode is set, DTMF chars are sent via SIP info messages. RFC2833 is not enabled.
DivaDtmfModeRFC2833SIPInfo	Use RFC2833 if supported by the remote peer. If DTMF tones cannot be transmitted via RFC2833, SIP Info messages are used. Incoming DTMF tones may be signaled as RFC2833, inband, or SIP Info, depending on the remote peer.

DivaDataCodec

Diva Media Boards support several compression codecs that can be used to convert G.711 audio used on the TDM line to compressed audio used by the application. The following options can be set via the call property *DivaCPT_DataCodec*.

```
typedef enum
{
        DivaDataCodecNone = 0,
        DivaDataCodecG729,
        DivaDataCodecG726,
        DivaDataCodecGSM,
        DivaDataCodecILBC,
        DivaDataCodecPCM16
        DivaDataCodecG722
} DivaDataCodec;
```

DivaDataCodecOptions

If an audio compression codec is selected, the codec options can be modified by the call property *DivaCPT_DataCodecOptions*. The following options are available:

```
typedef enum
{
        DivaDCO_None = 0x00,
        DivaDCO_EnableVAD = 0x01,
        DivaDCO_EnableComfortNoise = 0x02,
        DivaDCO_EnableDTMFDetection = 0x04,
        DivaDCO_EnableDTMFGeneration = 0x08,
        DivaDCO_All = 0x0F
} DivaDataCodecOptions;
```

DivaDCO_None

Use the default codec options.

DivaDCO_EnableVAD

If this option is enabled, the voice activity detection is activated. By default, the voice activity detection is off.

DivaDCO_EnableComfortNoise

If this option is enabled, the comfort noise generation is activated. By default, the comfort noise generation is off.

DivaDCO_EnableDTMFDetection

If this option is enabled, the DTMF detection is activated. By default, the DTMF detection is off.

DivaDCO_EnableDTMFGeneration

If this option is enabled, the DTMF generator is activated. By default, the DTMF generator is off.

DivaDCO_All

The value DivaDCO_All can be used to set all the above options.

DivaSampleRates

For the data codec DivaDataCodecPCM16 several sample rates are supported. The following sample rates can be set via the call property DivaCPT_DataCodecSampleRate. For all other codecs the sample rate is ignored.

```
typedef enum

{
        DivaSampleRate8000    =  0,
        DivaSampleRate16000    = 2,
        DivaSampleRate32000    = 4,
        DivaSampleRate48000    = 5,
```

```
} DivaSampleRates;
```

DivaSampleRate8000

The value DivaSampleRate8000 sets the sample rate for the data codec PCM 16 bit to 8000 KHz. This is the default sample rate.

DivaSampleRate16000

The value DivaSampleRate16000 sets the sample rate for the data codec PCM 16 bit to 16000 KHz.

DivaSampleRate32000

The value DivaSampleRate32000 sets the sample rate for the data codec PCM 16 bit to 32000 KHz.

DivaSampleRate48000

The value DivaSampleRate48000 sets the sample rate for the data codec PCM 16 bit to 48000 KHz.

DivaSamplingRate

```
typedef enum
{
        DivaSamplingRateMin = 1250,
        DivaSamplingRateNormal = 8000,
        DivaSamplingRateMax = 51200
} DivaSamplingRate;
```

DivaSamplingRateMin

Defines the minimum sampling rate supported by Diva Media Boards.

DivaSamplingRateNormal

Defines the normal or default-sampling rate used by Diva Media Boards.

DivaSamplingRateMax

Defines the maximum sampling rate supported by Diva Media Boards.

DivaDataChannelStatus

```
typedef enum
{
        DivaDataChannelDown = 0,
        DivaDataChannelUp
} DivaDataChannelStatus;
```

DivaDataChannelDown

The data channel has been disconnected based on a request initiated by the application via *DivaEnableDataChannel*.

DivaDataChannelUp

The data channel has been established based on a request initiated by the application via *DivaEnableDataChannel*.

DivaFSKModulation

```
typedef enum
{
        DivaFSKModulationV23 = 1
} DivaFSKModulation;
```

DivaFSKModulationV23

The FSK data is expected as V.23 modulation with 1200 Baud and frequencies 2100 and 1300.

DivaFSKEventTypes

```
typedef enum
{
        DivaFSKEventData = 0,
        DivaFSKEventCarrierOn = 14,
        DivaFSKEventCarrierOff = 15,
        DivaFSKEventPreamble = 200
} DivaFSKEventTypes;
```

DivaFSKEventData

The lower word of the parameter contains the data byte.

DivaFSKEventCarrierOn

The carrier for FSK data reception has been detected.

DivaFSKEventCarrierOff

The carrier for FRSK data reception is lost. The application may now retrieve all FSK data via the call property *DivaCPT_FSKData*.

DivaFSKEventPreamble

A preamble for the FSK data has been detected. This event type is not signaled if the application has enabled the transparent mode.

DivaFaxStatusType

DivaFaxStatusType defines the type of fax status indicated to the application. The information is signaled with the event *DivaEventDetailedFaxStatus*. Multiple information might be indicated with one event.

```
typedef struct
{
        DivaFaxStatusTrainingResult = 0x00000001,
        DivaFaxStatusTrainingStatistics = 0x00000002,
        DivaFaxStatusQualityReport = 0x00000004,
        DivaFaxStatusPartialPageReport  = 0x00000008,
        DivaFaxStatusTimeoutReport = 0x00000010,
        DivaFaxStatusResultReport = 0x00000020,
        DivaFaxStatusDISReport = 0x00000040,
        DivaFaxStatusDCSReport = 0x00000080,
        DivaFaxStatusPhaseReport = 0x00000100,
} DivaFaxStatusType;
```

DivaFaxStatusTrainingResult

The option *DivaFaxStatusTrainingResult* indicates that the training results can be retrieved by the call property DivaCPT_FaxReportTrainingResult.

DivaFaxStatusTrainingStatistics

The option *DivaFaxStatusTrainingStatistics* indicates that the training results can be retrieved by the call property DivaCPT_FaxReportTrainingStats.

DivaFaxStatusQualityReport

The option *DivaFaxStatusQualityReport* indicates that the detailed page quality information can be retrieved by the call property DivaCPT_FaxReportPageQuality.

DivaFaxStatusPartialPageReport

The option *DivaFaxStatusPartialPageReport* indicates that information for the fax Error Correction Mode (ECM) are received or sent. The information can be retrieved via the call property DivaCPT_FaxReportPartialPage.

DivaFaxStatusTimeoutReport

The option *DivaFaxStatusTimeoutReport* indicates that a T.30 timer has expired. Information about the timer can be retrieved via DivaCPT_FaxReportT30Timeout.

DivaFaxStatusResultReport

The option *DivaFaxStatusResultReport* indicates that phase E has been entered and provides the result code. The result code is coded as hexadecimal values according to T.32 .

DivaFaxStatusDISReport

The option *DivaFaxStatusDISReport* indicates that the DIS frame has been received from the remote peer. The raw DIS frame can be retrieved via the call property DivaCPT_FaxRemoteFeatures.

DivaFaxStatusDCSReport

The option *DivaFaxStatusDCSReport* indicates that the DCS frame has been received or sent. The raw DCS frame can be retrieved via the call property DivaCPT_FaxReportDCS.

DivaFaxStatusPhaseReport

The option *DivaFaxStatusPhaseReport* indicates that the fax phase has changed. The new phase can be retrieved via the call property DivaCPT_FaxT30Phase.

DivaFaxTrainingStats

DivaFaxTrainingStats provides information about the received training signal that can be retrieved.

```
typedef struct
{
    DWORD       GoodBytes;
    DWORD       ErrorBytes;
    DWORD       Noise;
} DivaFaxTrainingStats;
```

GoodBytes

The *GoodBytes* member contains the amount of training bytes received with all bits set to zero.

ErrorBytes

The *ErrorBytes* member contains the amount of training bytes that are not zero.

Noise

The *Noise* member contains the number of alternations between zero and non zero runs.

DivaFaxPageQualityDetails

DivaFaxPageQualityDetails provides information about scan lines, valid scan lines, and scan lines with errors can be retrieved.

```
typedef struct
{
    DWORD       TotalScanLines;
    DWORD       ErrorScanLines;
    DWORD       ConsecutiveErrors;
    DWORD       TotalBytes;
    DWORD       ErrorBytes;
    DWORD       ConsecutiveErrorBytes;
} DivaFaxPageQualityDetails;
```

TotalScanLines

The *TotalScanLines* member contains the total amount of scan lines received for this page.

ErrorScanLines

The *ErrorScanLines* member contains the amount of scan lines that contain errors.

ConsecutiveErrors

The *ConsecutiveErrors* member contains the amount of scan lines with errors received in a row.

TotalBytes

The *TotalBytes* member contains the total amount of bytes in the received scan lines for this page.

ErrorBytes

The *ErrorBytes* member contains the amount of bytes in scan lines that contain errors.

ConsecutiveErrorBytes

The *ConsecutiveErrorBytes* member contains the total amount of bytes in scan lines with errors received in a row.

DivaFaxPartialPageDetails

DivaFaxPartialPageDetails provides information about the ECM results that can be retrieved. Depending on the call direction, the returned information is either sent to the peer or received from the peer.

```
typedef struct
{
    DWORD       ECMFrameLength;
    BYTE        ECMState[32];
    BYTE        PPSFrameLength;
    BYTE        PPSFrame[255];
} DivaFaxPartialPageDetails;
```

ECMFrameLength

The *ECMFrameLength* member contains the length of an ECM data frame. Possible values are 64 or 256.

ECMState

The *ECMState* member contains the information about valid and invalid ECM frames. The bit 0 of byte 0 refers to ECM frame 0 and the bit 7 of byte 31 refers to frame 255. If a bit is set, the corresponding ECM frame is requested for a retransmit.

PPSFrameLength

The *PPSFrameLength* member contains the length of the following raw PPS frame.

PPSFrame

The *PPSFrame* member contains the raw PPS frame sent to the peer or received by the peer.

DivaFaxPhase

```
typedef enum
{
        DivaFaxPhaseA = 1,
        DivaFaxPhaseB,
        DivaFaxPhaseC,
        DivaFaxPhaseD,
        DivaFaxPhaseE
} DivaFaxPhase;
```

DivaFaxPhaseA

The fax phase *DivaFaxPhaseA* specifies that the connection establishment is active.

DivaFaxPhaseB

The fax phase *DIvaFaxPhaseB* specifies that the fax training is currently active.

DivaFaxPhaseC

The fax phase *DivaFaxPhaseC* specifies that the page data is transmitted or received.

DivaFaxPhaseD

The fax phase *DivaFaxPhaseD* specifies that the page results are exchanged.

DivaFaxPhaseE

The fax phase *DivaFaxPhaseE* specifies that the disconnect is initiated.

DivaDataFrameStatus

DivaDataFrameStatus defines the status options for processing modem data. The information is signaled with the event *DivaEventDataFrameStatus*.

```
typedef struct
{
        DivaDataFrameReceiveStarted,
        DivaDataFrameReceiveAborted,
} DivaDataFrameStatus
```

DivaDataFrameReceiveStarted

The option *DivaDataFrameReceiveStarted* indicates that the reception of data has started. When data reception is successfully finished, the application will receive the event *DivaEventDataAvailable*. If there is an error, the application will receive the event *DivaEventDataFrameStatus*.

DivaDataFrameReceiveAborted

The option *DivaDataFrameReceiveAborted* indicates that the reception of data has been aborted.

DivaCodecMask

DivaCodecMask defines the codecs that can be enabled for IP Media channel access. The *DivaCodecMask* can be used for the call type *DivaCallTypeRTPGwMode* or for plain IP media channel access via *DivaOpenIPMediaChannel*.

```
typedef struct
{
        DivaCodecMaskG711aLaw       = 0x00000001,
        DivaCodecMaskG711uLaw       = 0x00000002,
} DivaCodecMask;
```

DivaCodecMaskG711aLaw

The option *DivaCodecMaskG711ALaw* indicates that the codec G.711 a-Law should be enabled.

DivaCodecMaskG711uLaw

The option *DivaCodecMaskG711uLaw* indicates that the codec G.711 μ-Law should be enabled.

DivaMediaChannelStatus

DivaMediaChannelStatus defines the options for the status of the media channel reported by the event DivaEventMediaChannelStatus.

```
typedef struct
{
        DivaMediaChannelConnected = 0,
        DivaMediaChannelDown,
        DivaMediaChannelNoResources,
        DivaMediaChannelError,
} DivaMediaChannelStatus;
```

DivaMediaChannelConnected

The option *DivaMediaChannelConnected* indicates that the media channel can be used for audio streaming.

DivaMediaChannelDown

The option *DivaMediaChannelDown* indicates that the media channel has been disconnected and can no longer be used for audio streaming.

DivaMediaChannelNoResources

The option *DivaMediaChannelNoResources* indicates that the media channel could not be established, because no more IP channel resources are available.

DivaMediaChannelError

The option *DivaMediaChannelError* indicates that the media channel could not be established.

DivaMrcpVersion

The DivaMrcpVersion specifies the version of the MRCP protocol to use for communication to a speech engine, either a recognizer or synthesizer.

typedef enum

```
{
        DivaMrcpVersion1 = 1,
        DivaMrcpVersion2,
} DivaMrcpVersion;
```

DivaMrcpVersion1

The option DivaMrcpVersion1 indicates that the communication is done using MRCP version 1, which is based on RTSP.

DivaMrcpVersion2

The option DivaMrcpVersion2 indicates that the communication is done using MRCP version 2, which is based on SIP.

DivaSpeechRecognizerStatus

The DivaSpeechRecognizerStatus is signaled with the event DivaEventSpeechRecognizerStatus and specifies the status of the connection to a speech recognizer.

typedef enum

```
{
        DivaSpeechRecognizerStatusClosed = 0,
        DivaSpeechRecognizerStatusOpening,
        DivaSpeechRecognizerStatusOpened,
        DivaSpeechRecognizerStatusClosing,
        DivaSpeechRecognizerStatusOpenFailed,
} DivaSpeechRecognizerStatus;
```

DivaSpeechRecognizerStatusClosed

The status DivaSpeechRecognizerStatusClosed indicates that the status of the connection to the speech recognizer is closed.

DivaSpeechRecognizerStatusOpening

The status DivaSpeechRecognizerStatusOpening indicates that the connection to the speech recognizer is initiated by the application.

DivaSpeechRecognizerStatusOpened

The status DivaSpeechRecognizerStatusOpened indicates that the status of the connection to the speech recognizer is open, speech recognition may be started.

DivaSpeechRecognizerStatusClosing

The status DivaSpeechRecognizerStatusClosing indicates that the status of the connection to the speech recognizer is closing based on an application or recognizer request.

DivaSpeechRecognizerStatusOpenFailed

The status DivaSpeechRecognizerStatusOpenFailed indicates that the status of the connection to the speech recognizer is closed, the open request from the application failed.

DivaSpeechRecognizerProgress

The DivaSpeechRecognizerProgress is signaled with the event DivaEventSpeechRecognizerProgress and specifies the progress of a recognition initiated by DivaStartSpeechRecognizer.

typedef enum

```
{
        DivaSpeechRecognitionInProgress = 1,
        DivaSpeechRecognitionStartOfSpeech,
        DivaSpeechRecognitionCompleted,
        DivaSpeechRecognitionUserTerminated,
        DivaSpeechRecognitionNoMatch,
        DivaSpeechRecognitionNoInputTimeout,
        DivaSpeechRecognitionTimeout,
        DivaSpeechRecognitionGrammarError,
        DivaSpeechRecognitionGeneralError,
        DivaSpeechRecognitionSpeechTooEarly,
        DivaSpeechRecognitionSpeechTooLong,
        DivaSpeechRecognitionUnsupportedLanguage,
```

} DivaSpeechRecognitionProgress

DivaSpeechRecognitionInProgress

The progress DivaSpeechRecognitionInProgress is signaled when the speech recognition initiated by DivaStartSpeechRecognizer is active.

DivaSpeechRecognitionStartOfSpeech

The progress DivaSpeechRecognitionStartOfSpeech is signaled when the speech recognizer has detected that speech started. The application may use this status to stop playing an announcement.

DivaSpeechRecognitionCompleted

The progress DivaSpeechRecognitionCompleted is signaled when the speech recognizer has finished detection and the results can be retrieved via DivaGetSpeechRecognizerResults.

DivaSpeechRecognitionUserTerminated

The progress DivaSpeechRecognitionUserTerminated is signaled when the speech recognizer was stopped by a call to DivaStopSpeechRecognizer.

DivaSpeechRecognitionNoMatch

The progress DivaSpeechRecognitionNoMatch is signaled when the speech recognizer has finished the detection process but did not find any matches in the given grammar.

DivaSpeechRecognitionNoInputTimeout

The progress DivaSpeechRecognitionNoInputTimeout is signaled when the speech recognizer did not detect any speech for the given time.

DivaSpeechRecognitionTimeout

The progress DivaSpeechRecognitionTimeout is signaled when the speech recognizer has finished detection but no results were detected within the given recognition time.

DivaSpeechRecognitionGrammarError

The progress DivaSpeechRecognitionGrammarError is signaled when the speech recognizer has finished detection due to an error with the specified grammar.

DivaSpeechRecognitionGeneralError

The progress DivaSpeechRecognitionGeneralError is signaled when the speech recognizer has finished detection due to an error which was not specified in detail. Refer to the log files of the speech recognizer.

DivaSpeechRecognitionSpeechTooEarly

The progress DivaSpeechRecognitionSpeechTooEarly is signaled when the speech recognizer has finished detection but the result is not valid because speech has started too early.

DivaSpeechRecognitionSpeechTooLong

The progress DivaSpeechRecognitionSpeechTooLong is signaled when the speech recognizer has finished detection but the result is not valid because speech was too long to be processed.

DivaSpeechRecognitionUnsupportedLanguage

The progress DivaSpeechRecognitionUnsupportedLanguage is signaled when the speech recognizer detected a language that was not supported by the given grammar.

DivaSpeechRecognizerResultType

The DivaSpeechRecognizerResultType identifies the type of recognizer result information the application is requesting.

typedef enum

```
{
        DivaSpeechRecognizerResultInterpretation = 1,
        DivaSpeechRecognizerResultGrammar,
        DivaSpeechRecognizerResultContent
} DivaSpeechRecognizerResultType;
```

DivaSpeechRecognizerResultInterpretation

The type DivaSpeechRecognizerResultInterpretation is used by the application to retrieve the information how the recognizer interpreted the provided result. Options would be that the result is interpreted as speech or DTMF.

DivaSpeechRecognizerResultGrammar

The type DivaSpeechRecognizerResultGrammar is used by the application to retrieve the information about the grammar used for the recognition result.

DivaSpeechRecognizerResultContent

The type DivaSpeechRecognizerResultContent is used by the application to retrieve the recognition result as xml coded data.