Dialogic
Making Innovation Thrive

# Dialogic® Diva® Component API

# Developer's Reference Guide

Part of the Dialogic® Diva® Software Development Kit

April 2010 206-448-08

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

### Copyright

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

### License

Under the terms and conditions of this License Agreement:

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

### Upgrades

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

### Term

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

### Limited Warranty

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

### Customer Remedies

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

### No Other Warranties

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

### No Liability for Consequential Damage

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

### Limit of Liability

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

### Right to Audit

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

### Supplementary Software

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

### U.S. Government Restricted Rights

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

### Governing Law

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

### Contractor/ manufacturer is:

Dialogic CORPORATION.

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

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

# Contents

# CHAPTER 1

# About this Online Guide

## How to use this online guide

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

## Structure of this guide

This guide presents details and functional descriptions of all Dialogic® Diva® ActiveX Components. Examples, data structures, and return codes are 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 Dialogic® Diva® API for .NET. |
| Dialogic® Diva® Component API Overview | Introduction to and overview of the Diva ActiveX Components and their functionality |
| Operation Modes | Description of the operation modes provided with the Diva ActiveX Components |
| DivaCall Overview | Introduction to the component DivaCall. Description of methods, events, and properties of DivaCall |
| DivaCall References | Description of DivaCall references |
| DivaSystem References | Description of DivaSystem references |
| DivaInstance References | Description of DivaInstance references |
| DivaDevice References | Description of DivaDevice references |
| DivaConference References | Description of DivaConference references |
| DivaToneResult References | Description of DivaToneResult references |

# CHAPTER 2

# Dialogic® Diva® SDK Overview

The Dialogic Diva SDK can be used in combination with Dialogic® Diva® Media Boards and Dialogic® Host Media Processing (HMP) software. On these communication platforms, the Diva SDK provides the following application programming interfaces (APIs): the Dialogic® Diva® API, the Dialogic® Diva® Components API, and the Dialogic® Diva® API for .NET. For the Diva Media Boards, two additional APIs are available: the Dialogic® Diva® Management API and the Extended CAPI 2.0.

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

The Diva SDK includes the following components:

- Libraries providing functions to access the Dialogic® Diva® communication platforms
- DLLs containing the interfaces and component services
- Programming samples in source code
- Documentation explaining the functions of the Diva SDK

The components can be found as follows:

| Component | Path |
|---|---|
| Libraries of the Diva SDK | \SDK\BASIC\LIB\ |
| DLLs of the Diva SDK and compiled samples applications | \SDK\BASIC\BIN |
| Samples for the Diva SDK | \SDK\BASIC\SAMPLES\ |
| Libraries of the Diva Management API | \SDK\MANAGEMENT\LIB |
| Samples for the Management API | \SDK\MANAGEMENT\SAMPLES |
| Documentation | \SDK\DOC\ |

The Diva SDK is available for download from the Dialogic web site under http://www.dialogic.com/products/tdm_boards/development_tools/default.htm. After the download, extract the files to your hard disk and do not change the directory structure of the extracted files.

The 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

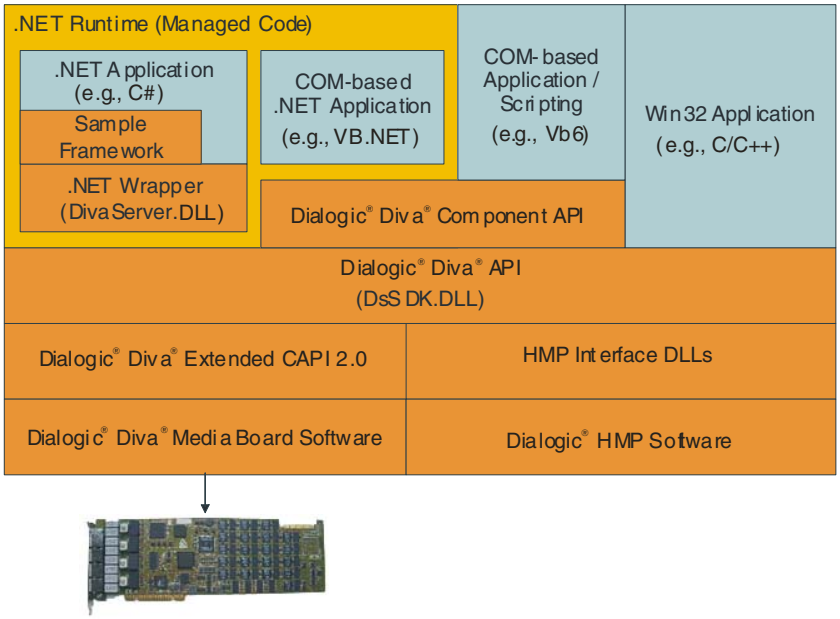
The five application programming interfaces (APIs) of the Dialogic® Diva® SDK represent different layers for the management and development of applications for Dialogic® Diva® communication platforms.

- Dialogic® Diva® API: It provides a high-level interface into the communication platforms that allows developers to implement communication applications. It also provides an additional library for data conversion like TIFF to SFF for fax applications.
- Dialogic® Diva® Component API: It provides a set of ActiveX components that allows developers to create new applications or to add telephony and communication features to existing applications. The Component API can be used from scripts and VB.NET and eliminates the need to write directly to a Microsoft® Visual C® / Visual C++® API.
- Dialogic® Diva® API for .NET: It is based on the Dialogic® Diva® API and provides access to the Diva API from .NET based applications.
- Extended CAPI 2.0 (only for Dialogic® Diva® Media Boards): It provides Dialogic-specific CAPI extensions that are fully CAPI 2.0 compliant.

- Dialogic® Diva® Management API: It provides direct hardware access for monitoring, security, and statistics. This API should only be used by applications using CAPI 2.0.

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

The following figure shows the architecture of the programming interfaces and the applications that may access them:




### Dialogic® Diva® API

The Diva API is a high-level interface into the Diva 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 thus can facilitate the development of applications for these areas. The modules are intended to be updated so as to offer development bases for an ever increasing range of 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. All 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.

### Dialogic® Diva® Component API

The Diva® SDK includes the Diva Component API. This interface provides a set of ActiveX components that allow for creating new applications or for adding telephony and communication features to existing applications. The components can be used in Windows®-based development environments that support ActiveX components. The Diva Component API provides functionalities including: extended call control, voice streaming and recording, change of media on existing calls, conferencing, call transfer, retrieve status information of the hardware, flexible tone detection, and answering machine detection.

### Dialogic® Diva® API for .NET

The Diva API for .NET is based on the Dialogic® Diva® API. The functionality is exactly the same and where possible a one to one mapping is done. The Diva API for .NET documentation describes the architecture, the mapping of data types, and the modifications that have been done due to the .NET and especially the C# requirements.

Applications written for .NET may be based on two APIs provided by the Diva SDK: the Dialogic® Diva® Component API and the Dialogic® Diva® API for .NET. Both APIs use the feature rich Diva API to access the underlying communication platform. The Diva Component API allows for synchronous processing and is therefore often used by script-oriented applications. The decision as to which Dialogic® Diva® API can be used depends on the application requirements.

### Extended CAPI 2.0

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

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

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

**Note:** If you are developing CAPI 2.0 applications based on Dialogic® Diva® for Windows® software, the CAPI 2.0 is part of the Diva API in the Dialogic® Diva® Configuration Manager.

### Dialogic® Diva® Management API

The Diva Management API is only available for Dialogic® Diva® Media Boards and only applications based on the CAPI 2.0 may use this interface. Applications using the Dialogic® Diva® API, the Dialogic® Diva® Component API, or Dialogic® Diva® API for .NET should not use this API.

The Diva Management API offers a range of functions to retrieve status and statistics information:

- Retrieve status of lowest level ISDN access
- Retrieve active calls
- Generating statistic information for number of calls, etc.
- Notification of status changes

Applications using the Dialogic® Diva® API do not need to access the Diva Management API, because the Diva API already has built-in support for the Diva Management API.

The Diva Management API only registers information and executes functions within your system. You cannot use it for data transfer. For data transfer, you should use the Diva API or CAPI interface.

The status information provided by the Diva Management API is structured as a sort of "virtual" file space. It contains nodes (similar to directories) and values (similar to files), where each node and value is defined by its path and name. The information can be read out using Diva Management API functions. The DLL providing these functions is part of the Dialogic® Diva® for Windows® software.

The functions provided by the Diva Management API are extended periodically; however, the Diva Management API will remain backwards compatible such that applications based on an earlier Diva Management API can continue to be used with new versions.

## Dialogic communication platform-related information

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

## Dialogic® HMP software configuration and licensing

The Dialogic HMP software is started either automatically when the system starts or manually depending on the configuration in the Dialogic® HMP Configuration Manager (DCM). With the initialization of the Dialogic® Diva® API by the application, the Dialogic HMP is initialized and configured. The configuration parameters are read from the file "dssdk.xml". Refer to the "Dialogic® HMP Software and Dialogic® Diva® SDK Installation and Configuration Guide" for details on those configuration parameters.

The Dialogic HMP features are based on licenses, and there are various options that can be combined. Based on the available licenses, Dialogic® Diva® API interface functions may return *DivaErrorNotSupported* if a requested function is not licensed or no more licenses are available. The following section provides detailed information.

The Diva API supports voice, conferencing, and fax on the Dialogic HMP software. In addition, mixed conferences between Dialogic® Diva® Media Boards and the Dialogic HMP software are supported. Tromboning, called Line Interconnect on the Diva API, is also supported between Diva Media Boards and the Dialogic HMP software.

The following license options are validated during startup of the Diva API:

- IP call control / RTP G.711
- voice
- speech integration
- conferencing
- fax

The Diva SDK allocates resources for the duration of a call. If a call is initiated as a voice call, a voice resource is allocated and assigned to this call. This resource remains allocated even if the application switches to fax mode later.

### IP call control / RTP G.711 resources

The amount of IP call control / RTP G.711 resources specifies the maximum number of channels. When started, the Dialogic® Diva® API creates the virtual line devices based on the configuration information that is specified in the configuration file. By default, only one line device is created for SIP-based communication using the available channels.

If line devices are configured and the amount of configured channels exceeds the licensed channels, the amount of channels for a line device or the amount of line devices may be limited. The application detects it in the information returned by *DivaGetNumLineDevices* and *DivaGetLineDeviceInfo*.

### Voice and speech integration resources

The Dialogic® HMP software provides voice resources for supporting features such as streaming audio, detecting DTMF tones, and generating DTMF tones. The capabilities of a voice resource depend on the license. If only "voice" is licensed, this resource can either play or record, but not both at the same time. A "speech integration" resource can play and record in parallel, and it supports an echo canceller. Based on the available voice resources,

the Dialogic® Diva® API supports three different modes and the interface functions *DivaRecordVoiceFile* and *DivaSendVoiceFile* may behave differently. During system start, the Dialogic® Diva® SDK enumerates the Dialogic HMP software resources and selects one of the following three modes:

- Two voice resources per channel
- One speech integration voice resource per channel
- One voice resource per channel

**Two voice resources per channel**

This mode allows to play and record in parallel and it is entered if two voice resources are available for each licensed IP-channel. The behavior of the functions *DivaRecordVoiceFile* and *DivaSendVoiceFile* is the same as on Dialogic® Diva® Media Boards.

When a call is established, the received audio is signaled to the application via the event *DivaEventDataAvailable* and can be retrieved by the application via *DivaReceiveAudio*.

**One speech integration voice resource per channel**

This mode is entered if one speech integration license per IP-channel is available and it allows to play and to record in parallel. The functions *DivaRecordVoiceFile* and *DivaSendVoiceFile* behave like on Diva Media Boards. For received audio, the echo canceller can be enabled.

When a call is established, the received audio is signaled to the application via the event *DivaEventDataAvailable* and can be retrieved by the application via *DivaReceiveAudio*.

This mode allows for retrieving audio in small buffer sizes and is the base for bridging between TDM and IP-based calls.

**One voice resource per channel**

This mode is entered if voice resources are licensed but none of the previously described modes can be selected. This mode allows to play or to record but not at the same time. If a recording is active, any *DivaSendVoice* function will fail.

When a call is established, the received audio is not automatically signaled to the application via the event *DivaEventDataAvailable*. If the application requires this event, it must enable this via the function *DivaEnableRxData* and ensure that no play is active.

**Conference resources**

The Dialogic® Diva® SDK uses the HMP conference resources when an IP-based call is added to a conference. Note that creating a conference via *DivaCreateConference* will always succeed, even if no conference resource is licensed. The conference resource is allocated when an IP-based call is added to a conference using *DivaAddToConference*.

For each IP-based call that is added to a conference, one conference party resource is allocated and released when the call is disconnected or removed from the conference. An additional conference party resource is required if TDM and IP calls are bridged or if a play or record operation on the conference object is initiated. Note that once this additional resource is allocated, it remains at the conference object until the last IP-based call is removed from the conference or the conference object is released using *DivaDestroyConference*.

Applications may record from any conference member. Playing to an IP-based call that is part of a conference is not possible. In this case, the function *DivaSendVoiceFile* and the other sent voice-related functions will return *DivaErrorInvalidState*. The same is valid for a call that is line interconnected (tromboned) to another call.

### Fax resources

The fax resources licensed for the Dialogic® HMP software support T.38 and clear channel fax, and the maximum supported speed is 14.400 bps. Fax resources are allocated when the application initiates a fax call or when the remote peer indicates a call as a fax call. For Dialogic HMP-based IP calls, the supported fax data format is TIFF. The following fax formats are supported:

- DivaFaxFormatTIFF_G3
- DivaFaxFormatTIFF_G4
- DivaFaxFormatColorJPEG

# CHAPTER 3

# Dialogic® Diva® Component API Overview

The Dialogic® Diva® SDK provides a set of ActiveX components that allows for creating new applications or for adding telephony and communication features to existing applications. Using the Dialogic® Diva® Component API eliminates the need to write directly to a Microsoft® Visual C® / Visual C++® API. It also abstracts the communication resources, so that users of the components do not need detailed understanding of communication architectures.

The Diva Component API can be used in any Windows®-based development environment that supports ActiveX components. The most common environment for the usage of ActiveX controls is Microsoft® Visual Basic®. The components of the Diva SDK are also designed to support scripting languages like Microsoft® Visual Basic® Script and Java Script.

The Diva Component API provides a flexible interface to create applications easily, and to set and retrieve applications that specify communication details. Moreover, it provides well defined functionalities that include methods, properties, and events for call control, voice streaming, fax communication, data communication, and supplementary services. The Diva SDK abstracts the underlying ISDN specifications and provides that the applications run in different environments without modification.

The communication channels of the installed Dialogic® Diva® Media Boards can be handled by the components, so that there is no need for the application to handle channel resources and call collisions.

## Functionality

The Dialogic® Diva® Component API provides the following functionalities:

- Simple Call Control
- Extended Call Control
- Voice Streaming and Recording
- Fax Communication including Fax Polling
- Change of Media of existing calls
- Detection and Generation of DTMF Digits
- Supplementary Services, Call Transfer, Hold / Retrieve, Conference
- Line Interconnect
- Retrieve information on installed hardware
- Retrieve status information of devices

## Components overview

The Dialogic® Diva® Component API provides the following components:

### DivaCall

DivaCall handles exactly one call. It is used for call setup as well as media-specific communication and also supplementary services. It enables you to write scripts or programs easily.

### DivaSystem

DivaSystem provides information on the number of installed Dialogic® Diva® communication resources and provides access to DivaDevice components for information on a specific board. In addition, instances are created that use DivaSystem.

**DivaInstance**

DivaInstance combines several DivaCall objects in one instance and is used for applications that process multiple calls at the same time. Call objects that are created based on DivaInstance get preset properties and the incoming call handling is optimized. The usage of DivaInstance is optional but recommended. DivaCall objects can also be created directly.

**DivaDevice**

DivaDevice represents a specific line with a well defined amount of channels and properties.

**DivaConference**

DivaConference allows for managing a conference with unlimited members. The conference, including mixing and automatic gain control, is handled by the underlying Dialogic® Diva® communication platform. Members of the conference are of the type DivaCall. The calls need to be connected before they can be added to the conference. The conference object is created based on DivaInstance. The method CreateConference returns an object of type DivaConference.

## Component install

The Dialogic® Diva® SDK contains a set of binaries, libraries, and sample applications. In general, the installation of the Diva SDK is done by copying the directory tree to the hard disk or network. Developers are free to copy the directory tree manually to any location.

The components need to be registered on the target computer. The Diva SDK does not automatically register the components. Developers may register the components using the configuration utility CONFIG.EXE available in the bin directory of the SDK, or they may register the components manually using the regsrv32 utility of the operating system. In the bin directory of the SDK exists a batch file for registration and deregistration.

# CHAPTER 4

# Operation Modes

The methods of a component can handle requests synchronously or asynchronously. Synchronous means that the call for accessing a method or property returns when the method has been executed. Asynchronous means that the result of the access is delivered later by an event or a different method.

All properties are handled synchronously and return the information directly. For those methods that originally work asynchronously, the operation mode can be either synchronous or asynchronous. The default is synchronous mode.

## Synchronous mode

If the synchronous mode is enabled, originally asynchronous methods are handled synchronously. In other words, they block the execution of the call until the requested operation is completed. This mode is also called blocking mode. It is selected by default if a component is created.

## Asynchronous mode

If the asynchronous mode is enabled, asynchronous methods initiate the requested method and return right away. The progress or result is either reported by events, if enabled, or by methods that wait until the called method is completed. This mode is also called non-blocking mode.

## Events

Events are optional and may be enabled by the application setting the *SignalEvents* property on the object. The ActiveX control exports an event interface that is conform to IDispatch. Applications may register to this interface and receive an event notification. Some scripting languages support the registration directly, e.g., in Microsoft® Visual Basic®, sub routines do the registration implicitly by writing to naming convention <object name>_<Event name>.

A separate thread signals the events. The Dialogic® Diva® SDK provides that thread blocking of the main application thread and the event signaling context is avoided.

## Switching operation mode

Once the object is created, the default operation mode is synchronous. The application may change the operation mode at any time.

**Note:** Operation mode and events are independent. An application may enable event reporting for incoming calls and run the call synchronously from the event function like a call script.

# CHAPTER 5

# DivaCall Overview

This chapter explains the creating, programming, and deleting of DivaCall objects.

## Creating DivaCall Objects

*DivaCall* can be created directly. The default is a voice call. All parameters are initialized to it. Applications that run one call per thread or process, e.g., a script, use a *DivaCall* object. A process may create several threads, each running one call.

A *DivaCall* object can also be created based on the component *DivaInstance*. *DivaCall* objects created on *DivaInstance* share common resources and have the same default properties, and they have to run in the same process space. Objects created on *DivaInstance* may use user-specific buffer sizes for communication. For applications serving several calls, it is recommended to create the *DivaCall* objects via *DivaInstance*.

## Deleting DivaCall Objects

Deleting objects in languages like Microsoft® Visual Basic® using the command "Object=Nothing" does not immediately delete the object itself. This is done later by the garbage collector. The application should remove the listen before deleting the object, to ensure that objects, which are no longer used by the application, do not take any calls. In general, it is recommended to create the objects once and use them over the runtime of the application.

## DivaCall programming

On the one hand, the methods and properties of *DivaCall* have been designed to provide easy call control and media streaming. On the other hand, they provide extended information and capabilities to applications that require this information.

A *DivaCall* object handles one call and the media-specific streaming. The communication resources and the channel can be selected automatically or by setting a specific line device. The used media, e.g., voice or fax, may be changed during the call. The different media modes are named call types. The following call types are supported:

- Voice (default)
- Fax
- Analog Modem
- Digital Data
- X.75, reliable digital data
- V.120, reliable digital data
- GSM, V.110 for mobile connections

The following tables provide an overview of the methods, properties, and events of *DivaCall*. The tables separate them by call orientation, media-specific and supplementary service-specific.

**Methods of DivaCall**

| Name | Comment |
|---|---|
| Connect | Establishes a connection using the call properties set previously to this call. |
| Disconnect | Disconnects a connection. |
| Listen | Only available if events are enabled. For non-event mode, use WaitForCall. |
| Alert | Sends an alert for an incoming call. |
| Answer | Answers an incoming call. Only available if events are enabled. For non-event mode, use WaitForCall. |
| Reject | Rejects an incoming call with a specific reason. |

| Methods of DivaCall | |
|---|---|
| **Name** | **Comment** |
| WaitForCall | Timed wait for incoming calls of the selected properties. |
| SetCallType | Changes the type of the call, e.g., from voice to fax. |
| WaitAsyncComplete | Waits for the last initiated asynchronous action to complete. |
| | |
| SendVoiceFile | Sends a single file. Asynchronous or synchronous depending on property AsyncMode. |
| SendVoiceFiles | Sends multiple files, e.g., announcement of time. Asynchronous or synchronous depending on the property AsyncMode. |
| SendVoiceFilesEx | Sends single or multiple files with the ability to return on certain digits. |
| StopSending | Terminates any pending streaming. |
| RecordVoiceFile | Records a voice file, options to limit time and silence. |
| StopRecording | Stops recording. |
| | |
| GetDigits | Waits for specific digits. |
| ClearDetectedDigits | Clears the internal digit buffer. |
| SendDigits | Sends single or multiple DTMF digits. |
| SendTone | Sends a single tone, optional continuous tone. |
| StopTone | Stops the sending of a continuous tone. |
| ClearDetectedTones | Clears internal tone buffer. |
| EnableSingleToneDetector | Enables the generic detector for a single tone. |
| EnableDualToneDetector | Enables the generic detector for dual tones. |
| DisableToneDetector | Disables the generic tone detector. |
| GetToneDetectorResult | Retrieves the results for a detected tone. |
| EnableAMD | Enables the answering machine detector. |
| DisableAMD | Disables the answering machine detector. |
| | |
| SendFax | Sends a single fax file. |
| SendFaxes | Sends multiple fax files to one receiver. |
| ReceiveFax | Receives a fax document to a file. |
| | |
| SendData | Sends plain data, depends on the call type. |
| ReceiveData | Receives plain data. |
| | |
| Hold | Puts a call on hold. |
| Retrieve | Retrieves a call. |
| BlindCallTransfer | Transfers a call to the given destination. |
| SetupSupervisedCallTransfer | Creates a consultation call for a supervised call transfer. |
| CompleteSupervisedCallTransfer | Completes a supervised call transfer. |
| CallDeflection | Delects a call to the given destination. |
| | |
| InterConnect | Trombones two calls with optional transaction recording. |
| InterDisconnect | Removes a previously set tromboning. |
| ConnectAudioProvider | Switches the audio stream for the call to an audio provider. |
| DisconnectAudioProvider | Disconnects the audio stream from the audio provider. |

| Properties of DivaCall | | | |
|---|---|---|---|
| **Name** | **Get** | **Put** | **Comment** |
| SignalEvents | x | x | If set, events are signaled. Please note that events and synchronous/ asynchronous mode are independent. |
| AsyncMode | x | x | Selects if following calls to methods are synchronous or asynchronous. |
| | | | |
| Device | x | x | Sets the line device to be used for an outgoing call. Reads the used line device. |
| LocalNumber | | x | Specifies the local number to be signaled with an outgoing call. |
| LocalSubAdress | | x | Specifies the local subaddress to be signaled with an outgoing call. |
| CallingNumber | x | | Calling number of an incoming call. |
| CalledNumber | x | | Called number of an incoming call. |
| Channel | x | | Used channel for the call. |
| RxSpeed | x | | Speed in receive direction. |
| TxSpeed | x | | Speed in send direction. |
| DialingComplete | x | x | Gets or sets the dialing complete state. |
| Compression | x | | Negotiates compression. Only if call type allows it. |
| DisconnectReason | x | | States the reason for the disconnection of the call. |
| | | | |
| DetectedDigits | x | | Returns the detected digits from the internal buffer. |
| | | | |
| EnableDTMFToneSplitting | | x | If true, store DTMF and tones in separate buffers. |
| DetectedTones | x | | Returns the detected tones. |
| SingleToneDetectorMinFrequency | | x | Sets the lower range to report detected tones. |
| SingleToneDetectorMaxFrequency | | x | Sets the upper range to report detected tones. |
| | | | |
| FaxLocalId | | x | Sets the local fax ID. |
| FaxHeadLine | | x | Sets the headline to be used for outgoing faxes. |
| FaxMaxSpeed | | x | Sets the maximum speed allowed. |
| FaxEnablePolling | | x | Enables the active or passive polling. |
| FaxDisableHighResolution | | x | Forces the fax to standard mode. |
| FaxDisableMR | | x | Disables Modified Read (MR) fax compression. |
| FaxDisableMMR | | x | Disables Modified Modified Read (MMR) fax compression. |
| FaxDisableECM | | x | Disables Error Correction Mode (ECM). |
| FaxMultipleDocument | | x | Specifies that the fax session will send multiple documents. |
| FaxReverseSession | | x | Fax session will change direction during a call. |
| FaxRemoteId | x | | ID of the remote fax. |
| FaxHighResolution | x | | Negotiated resolution, if true then high resolution. |
| FaxMRActive | x | | If true, the current call uses MR compression. |
| FaxMMRActive | x | | If true, the current call uses MMR compression. |
| FaxECMActive | x | | If true, the current call uses ECM. |
| FaxPollingActive | x | | If true, the fax session uses polling mode. |
| FaxPages | x | | Contains the amount of sent fax pages. |
| FaxEnableColor | | x | Enables the negotiation of the color fax mode. |
| FaxColorSelected | x | | Reads the result of the color fax negotiation. |
| FaxStoreMode | | x | Sets how received pages are stored. |

| Properties of DivaCall | | | |
|---|---|---|---|
| **Name** | **Get** | **Put** | **Comment** |
| | | | |
| VoiceEnableEchoCanceller | | x | If set, the echo canceller will be enabled for the connection. |
| VoiceEchoCancellerActive | x | | Specifies if the echo canceller is active. |
| EnableExtendedToneDetection | | | Enables or disables extended tone detection. |
| EnableDigitDetection | | | Enables the detection of DTMF and fax calling tones. |
| CallState | x | | Retrieves the call state. Valid states are defined in DivaCallState. |
| SignaledService | x | | Retrieves the service signaled for an incoming call. |
| Signal Service | | x | Sets the service to be signaled for an outgoing call. |
| RemoveDigitsFromStream | | x | Remove a received DTMF from a recorded audio signal. |
| X25CalledAddress | x | x | Sets the called address for an outgoing call. Read the called address for an incoming call. |
| X25CallingAddress | x | x | Sets the calling address for an outgoing call. Read the calling address for an incoming call. |
| X25NCPI | x | x | Sets or reads the plain NCPI for X.25. |
| TransferUseSameChannel | | x | Sets that the same channel is used for a consultation call created by BlindCallTransfer. |
| TranserfNoHold | | x | Sets that the main call is not placed on hold by BlindCallTransfer and SetupSupervisedCallTransfer. |
| TransferCompleteOnAlerting | | x | Sets that the transfer initiated by BlindCallTransfer is completed on alerting. |
| TransferCompleteOnProceeding | | x | Set that the transfer initiated by BlindCallTransfer is completed on proceeding. |
| TransferUseCallingNumber | | x | Specifies that the calling number should be signaled to the transfer destination. |
| TransferUseCallingName | | x | Specifies that the calling name should be signaled when a call is deleted. |
| InputVolume | | x | Sets the input volume. |
| OutputVolume | | x | Sets the output volume. |
| InputSamplingRate | | x | Sets the input sampling rate. |
| OutputSamplingRate | | x | Sets the output sampling rate. |
| RedirectNumber | x | | Reads the redirected number. |
| RedirectReason | x | | Reads the redirect reason. |
| CalledNumberType | x | x | Sets or reads the type of the called number. |
| CalledNumberId | x | x | Sets or reads the identifier of the called number. |
| CallingNumberType | x | x | Sets or reads the type of the calling number. |
| CallingNumberId | x | x | Sets or reads the identifier of the calling number. |
| CallingNumberPresentation | x | x | Sets or reads the calling number presentation. |
| CallingNumberScreening | x | x | Sets or reads the calling number screening. |

| Events of DivaCall | |
|---|---|
| **Name** | **Comment** |
| OnIncomingCall | Called if an incoming call is detected. Requires that SignalEvent is set to true and a listen has been placed. |
| OnCallProgress | Called when the call reaches the proceeding or alerting state. |
| OnConnected | Called if the call is connected. Requires that SignalEvent is set to true. |
| OnDisconnected | Called if the call is disconnected. Requires that SignalEvent is set to true. |
| OnToneReceived | Called if a tone, DTMF, or specific tone is received. The tone is provided with the call and remains also in the DigitBuffer. |
| | |
| OnVoiceStreamed | Called when the streaming of audio is finished. |
| OnRecordEnded | Called when the recording is finished. |
| | |
| OnFaxPageProcessed | Called when a fax page has been sent or received. |
| OnFaxProcessed | Called when a fax is successfully sent or received. |
| OnSuppServeCompleted | Called when a supplementary service operation is completed. |
| OnDataAvailable | Called when data for a data oriented call (not fax or voice) is available. |
| OnSingleToneDetected | Called when a single tone is detected. |
| OnDualToneDetected | Called when a dual tone is detected. |
| OnAMDFinished | Called when the answering machine detector has finished the analysis. |

### Outbound calls

Outbound calls are made by calling the *Connect* method on the *DivaCall* object. The application may set certain properties like call type and local number before calling *Connect*. With calling the *Connect* method the call type and the destination is selected.

### Inbound calls

An application may receive incoming calls on two ways. Either the application places a listen and is notified via event when the call comes in, or the application calls *WaitForCall*. The application can select the kind of services and an optional number filter for both options.

### Streaming and Recording Voice

Streaming and recording of audio can be handled by various methods using different audio formats. Audio streaming and recording is file-based, either as a wave file or as a so called raw file. Raw files contain only the audio data without any header information. The supported audio formats are defined by *DivaAudioFmt*. The following codecs are available:

- a-law, 8 KHz, 8 Bit Mono
- µ-law, 8 KHz, 8 Bit Mono
- PCM, 8 KHz, 8 Bit Mono
- PCM, 8 KHz, 16 Bit Mono

Streaming and recording can be done synchronously or asynchronously depending on the settings of the property *AsyncMode*. In synchronous mode, the method returns when the streaming or recording is finished. In asynchronous mode, the method returns when the streaming or recording is in process and the end is either signaled by event or the application uses *WaitAsyncComplete* to detect the end.

Audio can be streamed from a single file using *SendVoiceFile*. Multiple files can be streamed using *SendMultipleVoiceFiles*. For continuous streaming with or without automatic return upon receive of digits, *SendVoiceFilesEx* is available. Continuous streaming can be limited by a maximum duration.

When the end of a streaming is indicated, either by return of the method or an event, the audio is sent to the line.

For recording of audio the method, *RecordVoiceFile* is available. The method provides various options to limit the recording. All options are optional. The maximum record time can be specified as well as the maximum silence. Recording can be interrupted when receiving certain digits.

### Sending and Receiving Faxes

Sending and receiving fax documents is supported by high level functions. Faxes are sent from or received to files. Supported formats are TIFF, class F RLE and G3 compression, and the SFF format.

In synchronous mode, the methods return when the fax has been sent or received. This may take several minutes depending on the amount of pages.

In asynchronous mode, events are signaled, if enabled, for each page and for the completion of the document.

**Note:** The page event is not signaled for the last page of a document.

Multiple fax documents may be sent using one connection via the method *SendFaxFiles*. Using this function requires that the property *FaxMultipleDocument* is enabled before the connection is established.

### Switching Media Modes

Depending on the application it may be necessary to change the media mode of the data channel, while the physical call remains connected. The method *SetCallType* provides this function.

The application may set any data channel-related properties, e.g., fax properties before calling *SetCallType*.

### Detecting Digits and Tones

The Dialogic® Diva® Media Board supports the detection of DTMF digits and extended tones. Handling of DTMF digits and extended tones can be selected separately. The way of retrieving and processing detected digits or extended tones is done via the same set of methods and properties. By default, no detection is enabled. Two properties exist to enable and disable digit detection and extended tone detection:

- EnableDigitDetection
- EnableExtendedToneDetection

If one or both are enabled, the detected digits or tones are written to the internal buffer. They can be retrieved using the *DetectedDigits* property (read only).

**Note:** The detected digits or tones remain in the buffer until they are reset with the method *ClearDetectedDigits*.

The function *GetDigits* is used to set certain parameters and waits in synchronous mode for the settings. *GetDigits* implicitly enables the DTMF digit detection. If extended tone detection is enabled, these tones are also handled.

If the asynchronous mode is set, the application either uses *WaitAsyncComplete* to get the result from *GetDigits* or establishes an event method.

Applications may want to receive detected digits and tones in separate buffers. This can be enabled via the property *EnableDTMFToneSplitting*. If the property is set to true, the detected tones are placed in a separate buffer. This buffer can be retrieved by the property *DetectedTones*.

### Generating Digits and Tones

Digits and various tones can be generated. The method *SendDigits* sends one or more digits. The method is always synchronous and returns when the digits have been sent.

Tones can be generated as a single tone or as continuous tone. The method *SendTone* initiates sending of a tone. If the duration is set to non-zero, the tone is automatically stopped after the specified time. By setting the duration to zero, the application can control the length of the tone manually. In this case, the application has to call *StopTone* to stop the tone.

# CHAPTER 6

## DivaCall References

DivaCall Methods, DivaCall Properties, DivaCall Events, and DivaCall Result Codes are part of DivaCall References. This chapter describes these references in detail. For a description of DivaCall Methods, see below. You can find a detailed description of DivaCall Properties in the section DivaCall Properties on page 60. DivaCall Events are described in DivaCall Events on page 89. DivaCall Result Codes are described in DivaCall Result Codes on page 129.

## DivaCall Methods

This section contains the following DivaCall Methods:

- CompleteSupervisedCallTransfer
- SetCallProperties
- GetCallProperties
- ConnectAudioProvider
- DisconnectAudioProvider
- ReceiveAudio
- SendAudio
- ClearDetectedTones
- EnableSingleToneDetector
- EnableDualToneDetector
- DisableToneDetector
- GetToneDetectorResult
- EnableAMD
- DisableAMD
- ResetCodecList
- AddCodec

## Connect

Connects to a remote party.

retVal = object.Connect ( Destination, CallType )

**Parameter**

*Destination*

String value containing the number to dial.

*CallType* (optional)

Long value indicating the type of call. The parameter is optional. The default is voice.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

This is an asynchronous method. If the asynchronous mode is disabled (default), the method blocks until the call is completed or failed. The return code *DivaResultSuccess* indicates that the call has been established.

Note that call properties, e.g., FaxLocalId and headline in case of call type Fax, must be set prior to the call to Connect.

If asynchronous mode is enabled, the method initiates the call and returns. The return code *DivaResultSuccess* indicates that the call is in progress. The state of the call is either signaled as event *OnConnected* or the application may wait for asynchronous completion by *WaitAsyncComplete*.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ( "08154711" )
If ( retVal= DivaResultSuccess ) Then
  CallObj.Disconnect ( )
End If
```

**See also**

WaitAsyncComplete, OnConnected, OnCallProgress, OnAMDFinished

## Listen

Listens for a call on one or all devices.

retVal = object.Listen ( ServiceType, NumberFilter )

**Parameter**

*ServiceType* (optional)

Long value indicating the service to listen for. The parameter is optional, the default is *DivaListenServiceAll*.

*NumberFilter* (optional)

String value containing the filter for incoming calls. This can be a single number, or a list of numbers separated by semicolons. The numbers may contain wildcards for a single digit or a sequence. Ranges can be defined by two numbers separated by a "-" sign. The parameter is optional, by default no filter is active.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

This is an asynchronous method and works only if event reporting is enabled. Applications that disable event reporting mode need to use *WaitForCall*.

Incoming calls are checked against the service and the number filter. The event *OnIncomingCall* indicates if the call matches. When implementing *OnIncomingCall,* the allocation may process the call via synchronous functions.

The property *Device* sets the devices that should be used. By default, listen is enabled on all devices.

Example

The following Microsoft® Visual Basic® sample shows how incoming calls are signaled via event.

```
Dim WithEvents CallObj As DivaCall

Private Sub Form_Load ()
    Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
    retVal = CallObj.Listen ( DivaListenServiceAll )
End Sub

Sub CallObj_OnIncomingCall ( )
    'This will be called when the call is signaled.
    retVal = CallObj.Answer ( DivaCallTypeVoice )
    if ( retVal = DivaSuccess ) Then
      'Stream audio, then record and this is an answering machine.
    End If
EndSub
```

**See also**

WaitAsyncComplete, OnConnected

## WaitForCall

Waits for an incoming call for a maximum amount of time.

retVal = object.WaitForCall ( CallType, NumberFilter, MaxSecondsToWait )

### Parameter

*CallType* (optional)

A long value that indicates the type of calls to answer. The parameter is optional. The default is *DivaCallType_Voice*.

*NumberFilter* (optional)

String value containing the filter for incoming calls. This can be a single number, or a list of numbers separated by semicolons. The numbers may contain wildcards for a single digit or a sequence. Ranges can be defined by two numbers separated by a "-" sign.

The parameter is optional, by default no filter is active.

*MaxSecondsToWait* (optional)

A long value indicating the maximum amount of seconds to wait for an incoming call. If it is set to zero, no timeout is set.

### Returns

*DivaResultSuccess* (0) if a call is successfully connected. *DivaResultTimeout* if the waiting time is reached. If the call setup failed, an error code returns.

### Remarks

This is a purely synchronous method that is only available if asynchronous mode is disabled.

Incoming calls are checked against the service and the number filter. If the call matches, the call is automatically accepted using the given call type, and the method returns *DivaResultSuccess*. Depending on the call type, the application can start streaming or data transfer right away.

The *ListenDeviceMask* property sets the devices that should be used. By default, waiting for calls on all devices is enabled.

### Example

The following Microsoft® Visual Basic® sample shows how incoming calls are signaled via event.

```
Dim CallObj As DivaCall

Private Sub Form_Load ()
    Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
    retVal = CallObj.WaitForCall ( DivaCallType_Voice, "", 30 )
    If ( retVal = DivaResultSuccess ) Then
      CallObj.Disconnect ( )
    End If
End Sub
```

### See also

Disconnect, Listen, OnIncomingCall

## Alert

Sends alert for an incoming call.

retVal = object.Alert ( )

### Parameter

None

### Returns

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

### Remarks

An application may need some time to decide if the call should be answered. In order to keep the remote side ringing, an alert may be sent to the remote party.

*Alert* is a synchronous method. However, *Alert* can only be used if the application listens for calls using the *Listen* method. Applications using *WaitForCall* cannot use this method.

### Example

The following Microsoft® Visual Basic® sample shows how incoming calls are alerted.

```
Dim WithEvents CallObj As DivaCall

Private Sub Form_Load ()
    Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
    retVal = CallObj.Listen ( 0, "", 0 )
End Sub

Sub CallObj_OnIncomingCall ( )
    'This will be called when the call is signaled.
    Call CallObj.Alert ( )
EndSub
```

### See also

Listen, Answer, Reject

## Answer

Answers the call using the given call type.

retVal = object.Answer ( CallType )

### Parameter

*CallType* (optional)

Long value containing the call type to be used. The parameter is optional, the default value is *DivaCallType_Voice*.

### Returns

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

### Remarks

This is an asynchronous method and works only if asynchronous mode is enabled. Applications that disable asynchronous mode need to use *WaitForCall*.

The application needs to set the type of the call, e.g., voice or fax.

**Example**

The following Microsoft® Visual Basic® sample shows how incoming calls are signaled via event.

```
Dim WithEvents CallObj As DivaCall

Private Sub Form_Load ()
   Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
   retVal = CallObj.Listen ( )
End Sub

Sub CallObj_OnIncomingCall ( )
    'This will be called when the call is signaled.
    retVal = CallObj.Answer ( )
EndSub
```

**See also**

Listen, OnConnected, OnIncomingCall, Disconnect

## Reject

Rejects the call being signaled.

retVal = object.Reject ( Reason )

**Parameter**

*Reason* (optional)

The parameter specifies the reason for the call rejection. The default value is *DivaRejectNormalCallClearing*.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*Reject* is a synchronous method. It can only be used if the application listens for calls using the *Listen* method. Applications using *WaitForCall* cannot use this method.

The reason is sent to the network to inform the calling party why the call could not be answered. If the reason is set to *DivaAllowOthers*, the call is ignored. In certain environments, this allows other devices or applications to answer the call.

**Example**

The following Microsoft® Visual Basic® sample shows how incoming calls are signaled via event.

```
Dim WithEvents CallObj As DivaCall

Private Sub Form_Load ()
    Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
    retVal = CallObj.Listen ( 0, "", 0 )
End Sub

Sub CallObj_OnIncomingCall ( )
    'This will be called when the call is signaled.
   retVal = CallObj.Reject ( DivaRejectDestinationOutOfOrder )
EndSub
```

**See also**

Listen, OnConnected, OnIncomingCall, Disconnect, Alert, Answer

## Disconnect

Disconnects the call.

retVal = object.Disconnect ( )

**Parameter**

None

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*Disconnect* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the call is completed or failed. The return code *DivaResultSuccess* indicates that the call has been disconnected.

If asynchronous mode is enabled, the method initiates the disconnection and returns. The change of the call's state is either signaled with the event *OnDisconnected* or the application may wait for asynchronous completion by *WaitAsyncComplete*.

**Example**

The following Microsoft® Visual Basic® sample shows how incoming calls are signaled via event.

```
Dim WithEvents CallObj As DivaCall

Private Sub Form_Load ()
    Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
    retVal = CallObj.Listen ( 0, "", 0 )
End Sub

Sub CallObj_OnIncomingCall ( )
   'This will be called when the call is signaled.
   'Do processing then disconnect
   retVal = CallObj.Disconnect ( )
EndSub
```

**See also**

Listen, OnConnected, OnDisconnected, Connect, Answer

## SetCallType

Sets the call type for a connected call. This changes the media type of the call.

retVal = object.SetCallType ( NewCallType )

**Parameter**

*NewCallType*

Long value indicating the new type of the call.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SetCallType* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the call is completed or failed. The return code *DivaResultSuccess* indicates that the call type has been changed and the data channel is connected for the selected call type.

If asynchronous mode is enabled, the method initiates the change of the call type and returns. A return code of *DivaResultSuccess* indicates that the type change is in progress. The event *OnConnected* signals the state of the call, or the application may wait for asynchronous completion by *WaitAsyncComplete*.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ( "08154711", DivaCallTypeVoice )
If ( retVal= DivaResultSuccess ) Then
    'Call connected do some processing
    Call SendVoiceFile ( "GreetingMessage.wav" )
    retVal = SetCallType ( DivaCallTypeFax )
    If ( retVal = DivaResultSuccess )
      Call DivaSendFax ( "MyImage.tif" )
    End If
    CallObj.Disconnect ( )
End If
```

**See also**

WaitAsyncComplete, OnConnected

## WaitAsyncComplete

Waits for an asynchronous operation to complete.

retVal = object.WaitAsyncComplete ( MaxSeconds )

**Parameter**

*MaxSeconds*

Long value indicating the maximum time in seconds to wait.

**Returns**

The return value depends on the asynchronous operation that is currently in progress.

**Remarks**

*WaitAsyncComplete* is only available if asynchronous mode is enabled. It waits for the completion of one or more pending actions.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ( "08154711", DivaCallTypeVoice )
If ( retVal= DivaResultSuccess ) Then
    CallObj.AsyncMode = True
    Call SendVoiceFile ( "GreetingMessage.wav" )
    Call RecordVoiceFile ( "MyRecord.wav")
    WaitAsyncComplete ( 0 )
    CallObj.AsyncMode = False
    CallObj.Disconnect ( )
End If
```

**See also**

Connect, SendVoiceFile, RecordVoiceFile

## SendVoiceFile

Streams audio information from a file.

retVal = object.SendVoiceFile ( Filename, Format )

### Parameter

*Filename*

The parameter *Filename* is a string value that specifies the file to be streamed. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaAudioAutodetect*.

### Returns

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

### Remarks

*SendVoiceFile* opens the given file and streams the data. If the parameter *Format* is set to *DivaAudioAutodetect*, the format is automatically detected from the file header.

The file name may include drive / network share and path information. If only the file name is given, only the current directory is searched.

*SendVoiceFile* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the streaming is completed or the call is disconnected. A return code of *DivaSuccess* indicates that the streaming has finished.

### Example

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ("08154711")
If ( retVal= DivaResultSuccess ) Then
   retVal = SendVoiceFile ( "GreetingMessage.wav" )
   CallObj.Disconnect ( )
End If
```

### See also

SendVoiceFiles, SendVoiceFilesEx, StopSending, RecordVoiceFile

## SendVoiceFiles

Streams audio information from several audio files.

retVal = object.SendVoiceFiles ( Filenames, Format )

### Parameter

*Filename*

The parameter *Filename* is a string value that specifies the files to be streamed. The single files to be streamed are separated by commas. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaAudioAutodetect*.

### Returns

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SendVoiceFiles* streams the audio information from the files. No gap will occur between the different files. The parameter *Filename* contains the files to be streamed. The files are separated by commas.

If the parameter *Format* is set to *DivaAudioAutodetect*, the format is automatically detected from the file header. If the format is not set to *DivaAudioAutodetect*, the files need to have the same audio format.

The file names may include drive / network share and path information. If only the file name is given, only the current directory is searched.

*SendVoiceFiles* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the streaming is completed or the call is disconnected. A return code of *DivaSuccess* indicates that the streaming has finished.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ("08154711")
If ( retVal= DivaResultSuccess ) Then
   retVal = SendVoiceFiles ( "GreetingMessage.wav","Beep.wav" )
   CallObj.Disconnect ( )
End If
```

**See also**

SendVoiceFile, SendVoiceFilesEx, StopSending, RecordVoiceFile

## SendVoiceFilesEx

Streams audio information from one or several audio files and optionally detects and returns on digits.

retVal = object.SendVoiceFilesEx ( Filename, TonesToReturn, DetectedTone, Format, Continuous, MaxSeconds )

**Parameter**

*Filename*

The parameter *Filename* is a string value that specifies the files to be streamed. The single files to be streamed are separated by commas. See Remarks.

*TonesToReturn*

The parameter *TonesToReturn* is a string value that specifies a list of tones that complete an asynchronous operation.

*DetectedTone* (optional)

The parameter *DetectedTone* specifies the location to which the detected tone is written. The parameter is optional. The default value is zero.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional, the default value is *DivaAudioAutodetect*.

*Continuous* (optional)

The parameter *Continuous* is a Boolean value. If set, the audio streaming is repeated until the maximum time is reached or the call is disconnected. The parameter is optional. The default is false.

*MaxSeconds* (optional)

The parameter *MaxSeconds* is a long value that specifies the maximum time the audio should be streamed. A value of zero specifies no limitation. The parameter is optional. The default is no timeout.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SendVoiceFilesEx* streams the audio information from the files. Between the different files no gap will occur. The parameter *Filename* contains the files to be streamed. They are separated by commas.

If the parameter *Format* is set to *DivaAudioAutodetect*, the format is automatically detected from the file header. If the format is not set to *DivaAudioAutodetect*, the files need to have the same audio format.

The file names may include drive / network share and path information. If only the file name is given, only the current directory is searched.

*SendVoiceFilesEx* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the streaming is completed, one of the *TonesToReturn* is detected, or the call is disconnected. A return code of *DivaSuccess* indicates that the streaming is finished. If the return code is *DivaResultToneDetected*, the tone is reported via the parameter *DetectedTone*.

**Example**

```
Dim CallObj
Dim Detected as long
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ("08154711")
If ( retVal= DivaResultSuccess ) Then
    retVal = SendVoiceFilesEx ( "GreetingMessage.wav", "12", Detected )
    If ( retVal = DivaResultToneDetected ) Then
      If ( Detected = 49 ) Then
        'Action for tone '1'
      Else
        'Action for tone '2'
      End If
    End If
    CallObj.Disconnect ( )
End If
```

**See also**

SendVoiceFile, SendVoiceFiles, StopSending, RecordVoiceFile

## StopSending

Terminates the streaming of the data.

object.StopSending ( )

**Parameter**

None

**Returns**

None

**Remarks**

This is a synchronous method.

The method terminates any pending data streaming. If audio streaming is active, the pending audio buffers are discarded, and the streaming stops right away.

**Example**

```
CallObj.StopSending ()
```

**See also**

SendVoiceFile, SendVoiceFiles, SendVoiceFilesEx

## RecordVoiceFile

Records audio to a file. Optionally, it stops recording on certain criteria.

retVal = object.RecordVoiceFile ( Filename, Format, MaxSeconds, MaxSilence, TonesToReturn )

### Parameter

*Filename*

The parameter *Filename* is a string value that specifies the files to be streamed. The individual files to be streamed are separated by commas. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaAudioDefault*.

*MaxSeconds* (optional)

The parameter *MaxSeconds* is a long value and specifies the maximum recording length in seconds. A value of zero indicates no timeout. The parameter is optional. The default is no timeout.

*MaxSilence* (optional)

The parameter *MaxSilence* is a long value that specifies the maximum silence in seconds. If this timeout is reached, recording is finished. A value of zero specifies no silence detection. The parameter is optional. The default is no detection.

*TonesToReturn*

The parameter *TonesToReturn* is a string value that specifies a list of tones that complete a synchronous operation. See Remarks.

### Returns

In asynchronous mode, the return value is *DivaResultSuccess* (0) if successful. In case of an error, the method returns an error code. If the function is called with disabled asynchronous mode, the return value depends on the operation mode and the parameter. See Remarks.

### Remarks

The method records the audio information to the specified file. The file name may include drive / network share and path information.

If only the file name is given, the file is placed in the current directory. If the parameter *Format* is set to *DivaAudioDefault*, the format is set to PCM 8 KHz Mono.

*RecordVoiceFile* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until one of the following occurs:

- The call is disconnected. The return value is *DivaResultDisconnected.*
- *MaxSeconds* is set to non-zero and the timeout has been reached. The return value is *DivaResultTimeReached*.
- *MaxSilence* is set and the silence has been detected. The return value is *DivaResultSilenceDetected.*
- The tones to return are specified and one of those is detected. The return value is *DivaResultToneDetected.* The digit buffer contains the tones detected during recording.

In asynchronous mode, the method returns and the progress is signaled by events. In this mode, the TonesToReturn are not evaluated. Any received DTMF digit is signaled to the *OnToneReceived* function and the recording needs to be stopped from this function using *StopRecording*.

**Example**

```
'Simple answering machine

Dim CallObj
Dim Detected as long
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.WaitForCall ( )
If ( retVal= DivaResultSuccess ) Then
   retVal = SendVoiceFile( "GreetingMessage.wav" )
   If ( retVal = DivaResultSuccess ) Then
     Call RecordVoiceFile ( "MyMessage.wav" )
   End If
   CallObj.Disconnect ( )
End If
```

**See also**

SendVoiceFile, SendVoiceFiles, StopSending, StopRecording

## StopRecording

Terminates the recording of the data.

object.StopRecording ( )

**Parameter**

None

**Returns**

None

**Remarks**

The method terminates any pending audio recording.

*StopRecording* is a purely synchronous method. When the function returns, the audio recording has been finished and the audio file is closed.

**Example**

```
CallObj.StopRecording ()
```

**See also**

RecordVoiceFile

## GetDigits

Detects and waits for certain digits.

retVal = object.GetDigits ( MaxDigits, MaxTime, MaxSilence, Digits, DigitsToReturn )

**Parameter**

*MaxDigits*

The parameter *MaxDigits* is a long value that specifies the amount of digits to wait for.

*MaxTime* (optional)

The parameter *MaxTime* is a long value and specifies the maximum time to wait for digits in seconds. A value of zero indicates no timeout. The parameter is optional. The default is no timeout.

*MaxSilence* (optional)

The parameter *MaxSilence* is a long value that specifies the maximum time in seconds between two received digits. If the time is reached, the detection is finished. A value of zero specifies no timeout. The parameter is optional. The default is no detection.

*Digits* (optional)

The parameter *Digits* specifies the location where to place the detected digits. The parameter is optional. The default value is no buffer. The detected tones are also added to the digit buffer.

*DigitsToReturn* (optional)

The parameter *DigitsToReturn* is a string value that specifies a list of digits which completes an asynchronous operation.

**Returns**

In asynchronous mode, the return value is *DivaResultSuccess* (0) if successful. In case of an error, the method returns an error code.

If the function is called with disabled asynchronous mode, the return value depends on the operation mode and the parameter. See Remarks.

**Remarks**

The method initiates the detection of digits. The detected digits are placed in the digit buffer and can be retrieved by *GetDetectedDigits*.

**Note:** The digits remain in the buffer and need to be cleared using the *ClearDetectedDigits* method.

*GetDigits* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until one of the following occurs:

- The call is disconnected. The return value is *DivaResultDisconnected.*
- The specified amount of digits has been detected or one of the *DigitsToReturn* has been detected. The return value is *DivaResultToneDetected*.
- MaxSeconds is set to non-zero and the timeout has been reached. The return value is *DivaResultTimeReached*.
- MaxSilence is set and the timeout has been detected. The return value is *DivaResultSilenceDetected.*

**Example**

```
'Wait 10 seconds for a 3 digit extension.
  retVal = CallObj.GetDigits ( 3, 10 )
```

**See also**

ClearDetectedDigits, SendDigits, SendTone, StopTone

## ClearDetectedDigits

Removes any digits from the internal buffer.

object.ClearDetectedDigits ( )

**Parameter**

None

**Returns**

None

**Remarks**

*ClearDetectedDigits* deletes any detected tones or digits from the internal buffer.

It is a purely synchronous method.

**Example**

```
CallObj.ClearDetectedDigits ()
```

**See also**

GetDigits, SendDigits, SendTone, StopTone

## SendDigits

Sends the given digits.

retVal = object.SendDigits ( Digits )

**Parameter**

*Digits*

The parameter *Digits* is a string value that contains the digits to be sent.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SendDigits* sends the given digits. It is a purely synchronous method.

**Example**

```
retVal = SendDigits ( "998877" )
```

**See also**

GetDigits, ClearDetectedDigits, SendTone, StopTone

## SendTone

Sends the given tone.

retVal = object.SendTone ( Tone, Duration )

**Parameter**

*Tone*

The parameter *Tone* contains the tone to be sent and is a value of the type DivaTones.

*Duration* (optional)

The parameter *Duration* specifies the duration of the tone in milliseconds. A value of zero is the default duration. The parameter is optional.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SendTone* sends the given tone for the specified duration. It is a purely synchronous method.

**Example**

```
retVal = SendTone ( Diva_ToneFaxCalling )
```

**See also**

GetDigits, ClearDetectedDigits, SendDigits, StopTone

## StopTone

Stops the currently active continuous tone.

object.StopTone ( )

**Parameter**

None

**Returns**

None

**Remarks**

*StopTone* stops the sending of a continuous tone previously initiated by *SendTone* with asynchronous mode enabled.

It is a purely synchronous method.

**Example**

```
CallObj.StopTone ()
```

**See also**

GetDigits, SendDigits, SendTone

## SendFax

Sends the given fax using the current call.

retVal = object.SendFax ( Filename, Format )

**Parameter**

*Filename*

The parameter *Filename* is a string value that specifies the fax documents to be sent. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaFaxFmtAutodetect*.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SendFax* opens the given file, converts the data if necessary, and sends the fax. If the parameter *Format* is set to *DivaFaxFmtAutodetect*, the format is automatically detected from the file extension and the header.

The file name may include drive / network share and path information. If only the file name is given, only the current directory is searched.

*SendFax* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the sending is completed or the call is disconnected. This may take several minutes depending on the document. A return code of *DivaSuccess* indicates that the streaming has finished. If asynchronous mode is enabled, the method returns right away and the application may process the call by events or by using *WaitAsyncComplete*. The events that are signaled are *OnFaxPageProcessed* and *OnFaxProcessed*.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ("08154711", DivaCallType_Fax )
If ( retVal= DivaResultSuccess ) Then
    retVal = SendFax ( "MyFax.tif" )
    CallObj.Disconnect ( )
End If
```

**See also**

SendFaxes, ReceiveFax

## SendFaxes

Sends the given multiple fax using the current call.

retVal = object.SendFaxes ( Filename, Format )

**Parameter**

*Filename*

The parameter *Filename* is a string value that specifies the fax documents to be sent. The single document names are separated by commas. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaFaxFmtAutodetect*.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SendFaxes* sends the fax documents from the files. Between the different documents the receiver is notified that a new document has started. The parameter *Filename* contains the files to be streamed separated by commas.

If the parameter *Format* is set to *DivaAudioAutodetect*, the format is automatically detected from the file extension and the file header. If the format is not set to *DivaFaxFmtAutodetect*, the files need to have the same fax format.

The file name may include drive / network share and path information. If only the file name is given, only the current directory is searched.

**Note:** The way files are interpreted depends on the property *FaxMultipleDocument*. If this property is set, each file is sent as a separate document. If the property is not set, the files are combined to one document.

*SendFaxes* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the sending is completed or the call is disconnected. This may take several minutes depending on the document. A return code of *DivaSuccess* indicates that the streaming has finished. If asynchronous mode is enabled, the method returns right away and the application may process the call by events or by using *WaitAsyncComplete*. The signaled events are *OnFaxPageProcessed* and *OnFaxProcessed*.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
CallObj.FaxMultipleDocument = True
retVal = CallObj.Connect ("08154711", DivaCallType_Fax )
If ( retVal= DivaResultSuccess ) Then
    retVal = SendFaxs ( "MyFax.tif,Order.tif,Greeting.Tif" )
    CallObj.Disconnect ( )
End If
```

**See also**

SendFax, ReceiveFax

## ReceiveFax

Receives the fax and stores it as the given document.

retVal = object.ReceiveFax ( Filename, Format )

**Parameter**

*Filename*

The parameter *Filename* is a string value that specifies the name of the received fax documents. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaFaxFmtDefault*.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*ReceiveFax* receives the fax document and stores the data in the file specified by the file name. If the parameter *Format* is set to *DivaFaxFmtDefault*, the format is set to *DivaFaxFmtTiffClassF*.

The file name may include drive / network share and path information. If only the file name is given, only the current directory is searched.

*ReceiveFax* is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the receiving is completed or the call is disconnected. It may take several minutes depending on the document. A return code of *DivaSuccess* indicates that the streaming has finished. If asynchronous mode is enabled, the method returns right away and the application may process the call by event processing or by using *WaitAsyncComplete*. The signaled events are *OnFaxPageProcessed* and *OnFaxProcessed*.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.WaitForCall ( DivaCallType_Fax )
If ( retVal= DivaResultSuccess ) Then
    retVal = CallObj.SendFax ( "MyFax.tif" )
    CallObj.Disconnect ( )
End If
```

**See also**

SendFaxes, SendFax

## SendData

Sends plain data.

retVal = object.SendData ( DataLength, pData, Handle )

**Parameter**

*DataLength*

The parameter *DataLength* is an integer value that specifies the amount of data to be sent.

*pData*

The parameter *pData* specifies the location of the data to be sent. The application passes the data as array of bytes.

*Handle* (optional)

The parameter *Handle* is a long value given to identify the buffer by the caller. This parameter is optional and currently not used.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method sends the given data to the remote side. The data is not interpreted.

This is a synchronous method. The method blocks until the data is sent on the line. A return code of *DivaSuccess* indicates that the data has been sent. Depending on the underlying protocol, the data may not yet be on the line.

**Example**

```
Dim CallObj
Dim Mydata(10) As Byte
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.WaitForCall ( DivaCallType_Modem )
If ( retVal= DivaResultSuccess ) Then
   retVal = CallObj.SendData ( 10, Mydata )
   CallObj.Disconnect ( )
End If
```

**See also**

ReceiveData

## ReceiveData

Receives plain data.

retVal = object.ReceiveData ( DataLength, pData, pLengthReceived, MinReceived, MaxSeconds )

**Parameter**

*DataLength*

The parameter *DataLength* is an integer value that specifies the maximum amount of data to be stored in the receive buffer.

*pData*

The parameter *pData* specifies the location where to place the data. The application passes an array of bytes to be filled with the received data.

*pLengthReceived*

The parameter *pLengthReceived* is the location of a long value where the amount of bytes copied to the buffer is inserted.

*MinReceived* (optional)

The parameter *MinReceive* is a long value that specifies the minimum amount of data to wait for. The parameter is optional.

*MaxSeconds* (optional)

The parameter *MaxSeconds* is a long value that specifies the maximum amount of time to wait for the received data. The parameter is optional. By default, there is no timeout.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method retrieves received data to the given buffer. The data is not interpreted.

The availability of data is signaled by the event *OnDataAvailable*, the amount of available data is given with the event. The event needs to be enabled using the *SignalEvent* property.

This is a synchronous method. The application may control the amount of data and the time to wait using the parameter *MinReceived* and *MaxSeconds*. By default, the method waits without any timeout and returns with the first received data. The application may specify a minimum amount of data to be received before return and / or a maximum time to wait.

**Example**

```
Dim CallObj
Dim Mydata(10) As Bytes
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.WaitForCall ( DivaCallType_Modem )
If ( retVal= DivaResultSuccess ) Then
    retVal = CallObj.ReceiveData ( 10, Mydata, DataReceived )
    CallObj.Disconnect ( )
End If
```

**See also**

SendData, OnDataAvailable

## Hold

Moves a connected call to the hold state.

retVal = object.Hold ( )

**Parameter**

None

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

If the call is connected, the transition to the hold state is initiated.

This is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the transition to the hold state is completed or the transition failed. The return code and the call state upon return show the result.

If asynchronous mode is enabled, the method returns right away and the application may process the transition by events or by using *WaitAsyncComplete*. The signaled event is *OnSuppServeCompleted*.

If the call is not in the connected state, the method returns *DivaResultInvalidState*. If the call is already in the hold state, *DivaResultSuccess* is returned for both modes. If the event *OnSuppServeCompleted is* enabled, it is triggered as well. The event may occur before the hold method returns.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ( "08154711" )
If ( retVal= DivaResultSuccess ) Then
    retVal = CallObj.Hold ( )
End If
```

**See also**

Retrieve, BlindCallTransfer, OnSuppServeCompleted, OnDisconnected

## Retrieve

Retrieves a call that is in the hold state.

retVal = object.Retrieve ( )

**Parameter**

None

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method retrieves a call, which is on hold, back to the connected state.

This is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the transition to the connected state is completed or the transition failed. The return code and the call state upon return show the result.

If asynchronous mode is enabled, the method returns right away and the application may process the transition by events or by using *WaitAsyncComplete*. The signaled event is *OnSuppServeCompleted*.

If the call is not in the hold state, the method returns *DivaResultInvalidState*. If the call is already in the connected state, *DivaResultSuccess* is returned for both modes. If the event *OnSuppServeCompleted is* enabled, it is also triggered. The event may occur before the hold method returns.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ( "08154711" )
If ( retVal= DivaResultSuccess ) Then
    retVal = CallObj.Hold ( )
    ' Do Something else, e. g. other call
    retVal = CallObj.Retrieve ( )
End If
```

**See also**

Hold, BlindCallTransfer, OnSuppServeCompleted, OnDisconnected

## BlindCallTransfer

Transfers the active call to the given number.

retVal = object.BlindCallTransfer (Destination, MaxSecondsToWait)

**Parameter**

*Destination*

The parameter *Destination* specifies the number to dial for the second call. See Remarks.

*MaxSecondsToWait* (optional)

The parameter *MaxSecondsToWait* is a long value that specifies the maximum amount of time the transfer should take.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method transfers the active call to the given destination. The original call defined by the call object needs to be in the connected or hold state. If the call is not in the hold state, the call is placed on hold before dialing to the given destination is initiated. Once the call is connected, the transfer is complete.

The application may limit the time for the transfer, especially for the establishment of the second call. If the optional parameter *MaxSecondsToWait* is set to non-zero, the transfer will be aborted when the timeout is reached.

This is an asynchronous method. If asynchronous mode is disabled (default), the method blocks until the transfer is completed or the transfer failed. If the transfer is finished, the call is already connected when the function returns.

If asynchronous mode is enabled, the method returns right away and the application may process the transfer by events or by using *WaitAsyncComplete*. On successful completion of the transfer, the event *OnSuppServeCompleted* with the parameter *bSuccess* is signaled. The original call is already disconnected at this time and no mode events are signaled. *OnDisconnected* is signaled in case the original call is disconnected with an unsuccessful transfer.

**Example**

```
Dim CallObj
Set CallObj = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj.Connect ( "08154711" )
If ( retVal= DivaResultSuccess ) Then
      retVal = CallObj.Hold ( )
    ' Do Something else, e. g. other call
   retVal = CallObj.BlindCallTransfer ( "01234567")
End If
```

**See also**

Hold, Retrieve, OnSuppServeCompleted, OnDisconnected

## CallDeflection

Deflects the active call to the given number.

retVal = object.CallDeflection (Destination)

**Parameter**

*Destination*

The parameter *Destination* specifies the number or address to which the call is deflected.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method deflects the active call to the given destination. The active call defined by the call object needs to be in the offering or connected state.

This is an asynchronous method. If asynchronous mode is disabled (default), the method is blocked until the deflection is completed or the deflection failed.

If asynchronous mode is enabled, the method returns right away and the application may process the deflection by events or by using *WaitAsyncComplete*. On successful completion of the deflection, the event *OnSuppServeCompleted* with the parameter *bSuccess* is signaled. The original call is already disconnected at this time and no mode events are signaled. *OnDisconnected* is signaled in case the original call is disconnected with an unsuccessful deflection.

**See also**

Hold, Retrieve, OnSuppServeCompleted, OnDisconnected

## InterConnect

Interconnects two connected calls and routes the audio signal between them.

retVal = object1.InterConnect ( object2 )

**Parameter**

*object2*

DivaCall object of the call that should be interconnected with the call specified by object1.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

Both calls identified by object1 and object2 need to have an assigned data channel. This requires that the calls are at least on the proceeding state. The calls are interconnected and the audio signal from object1 is routed to object2 and vice versa.

This is a synchronous method. When the method returns with the result code *DivaResultSuccess*, the calls are interconnected. In case one of the objects is in an invalid state or the interconnect fails due to hardware limitation, an error code is returned.

**Example**

```
Dim CallObj1
Dim CallObj2

Set CallObj1 = CreateObject ( "DivaSDK.DivaCall" )
Set CallObj2 = CreateObject ( "DivaSDK.DivaCall" )
retVal = CallObj1.Connect ( "08154711" )
If ( retVal= DivaResultSuccess ) Then
    retVal = CallObj2.Connect ( "08154722 )
    If ( retVal= DivaResultSuccess ) Then
      retVal = CallObj1.InterConnect ( CallObj2 )
    End If
End If
```

**See also**

InterDisconnect

## InterDisconnect

Removes the interconnect between two calls.

retVal = object1.InterDisconnect ( *bDisconnect* )

**Parameter**

*bDisconnect*

BOOL value indicating if the calls should be disconnected.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The call identified by the DivaCall object needs to be interconnected to another call. The interconnect in condition is removed. If the parameter *bDisconnect* is not set, the calls remain connected to the remote peer. Otherwise both calls are also disconnected.

This is a synchronous method. When the method returns with the result code *DivaResultSuccess*, the interconnect condition is cleared. In case the object is in an invalid state, an error code is returned.

**See also**

InterConnect

## SetupSupervisedCallTransfer

*SetupSupervisedCallTransfer* initializes a supervised call transfer.

CallObject.SetupSupervisedCallTransfer ( ConsultationCallObject )

**Parameter**

*ConsultationCallObject*

This parameter specifies the call object of type *DivaCall* that will be used for the consultation call.

**Returns**

If the supervised transfer is initialized successful, the method returns *DivaResultSuccess* (0).

**Remarks**

For a supervised call transfer, the consultation call needs to be established in a way that allows the switching equipment to complete the transfer. In general, the application is free to initialize a call object manually and establish the call. The method *SetupSupervisedCallTransfer* provides a more convenient way to handle this.

The method is called with a call object that has been created on the same DivaInstance. The new call object is filled with the necessary parameters to initiate the call when the method returns. The actions done by this method depend on the property *TransferNoHold*. By default, this property is set to false and the existing call is put on hold. Depending on the *TransferUseSameChannel* property, the channel information is set in the consultation call object.

In synchronous mode, the method returns when the consultation call has been initialized and the existing call is on hold.

In asynchronous mode, the method returns right away after the consultation call is initialized. If the existing call is put on hold, which is the default, the result is communicated via the event *OnSuppServeCompleted*

The application is responsible for start dialing on the consultation call.

**See also**

CompleteSupervisedCallTransfer, BlindCallTransfer, TransferNoHold, TransferUseSameChannel

## CompleteSupervisedCallTransfer

*CompleteSupervisedCallTransfer* completes a supervised call transfer.

CallObject.CompleteSupervisedCallTransfer ( ConsultationCallObject, Timeout )

**Parameter**

*ConsultationCallObject*

This parameter specifies the call object of type *DivaCall*. The object has a connection to the destination of the call transfer.

**Returns**

If the supervised transfer is completed, successful the method returns *DivaResultSuccess* (0).

**Remarks**

The call transfer of the call, which is identified by the call object, to the destination, which is identified by the consultation call object, is initiated. The consultation call needs to be in a state that allows the switching equipment to complete a call transfer.

In synchronous mode, the method returns when the transfer is either completed of failed. In asynchronous mode, the method returns right away and the result is reported by the event *OnSuppServeCompleted*.

**See also**

SetupSupervisedCallTransfer, BlindCallTransfer, TransferNoHold, TransferUseSameChannel

## SetCallProperties

Sets a call property for either signaling or media.

retVal = object.SetCallProperty ( Type, Value )

**Parameter**

*Type*

The parameter *Type* specifies the property. For valid types and the data formats, refer to DivaCPT.

**Value**

The parameter *Value* contains the information to be set. The type of the parameter depends on the property. See Remarks.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The most common properties for a call are passed as object properties to a DivaCall object. However, the Dialogic® Diva® communication platform supports a wide range of properties for call setup and media handling. These extended properties are covered via the extensible properties handled by *SetCallProperties* and *GetCallProperties*. The type *DivaCPT* lists the extended call properties and the corresponding types. The type is checked for the given VARANT type. If a mismatch is detected, the method returns an error.

The function is always synchronous and returns right away.

**Note:** The modification of an extended call property does only store the value. The real action behind is handled when the corresponding method that uses the property is called.

**Example**

```
Dim bFlag As Boolean
Dim Result As DIVASDKLib.DivaResultCodes
bFlag = True
Result = MyCall.SetCallProperty( DIVASDKLib.DivaCPT. CPT_EchoCancellerEnableNLP,bFlag )
```

**See also**

GetCallProperties

## GetCallProperties

Retrieves a call property for either signaling or media.

retVal = object.GetCallProperty ( Type, Value )

**Parameter**

*Type*

The parameter *Type* specifies the property. For valid types and the data formats, refer to DivaCPT.

*Value*

The parameter *Value* specifies where to place the requested information. The type of the parameter depends on the property.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The most common properties for a call are passed as object properties to a DivaCall object. However, the Dialogic® Diva® communication platform supports a wide range of properties for call setup and media handling. These extended properties are covered via the extensible properties handled by *SetCallProperties* and *GetCallProperties*. The type *DivaCPT* lists the extended call properties and the corresponding types.

**Example**

```
Dim Name As String
Dim Result As DIVASDKLib.DivaResultCodes
Result = MyCall.GetCallProperty( DIVASDKLib.DivaCPT.CPT_CalledName, Name)
```

**See also**

SetCallProperties

## ConnectAudioProvider

Attaches an audio provider to the data channels of an existing call.

CalObject.ConnectAudioProvider ( ProviderName, ProviderChannelID, Mode )

**Parameter**

*ProviderName*

*ProviderName* is a string parameter containing the name of the audio provider. This needs to be the same name the audio provider used to register with the Dialogic® Diva® API.

*ProviderChannelID*

If the audio provider supports multiple channels, this parameter defines the logical channel instance to be used. The format of this is depending on the audio provider.

*Mode*

The parameter *Mode* defines in which directions the streaming should be done. For valid options, see DivaAudioProviderMode.

**Returns**

If the audio provider exists and the instances could be connected, the method returns *DivaResultSuccess*. If the audio provider name is not recognized, the method returns *DivaResultInvalidHandle*. A failure to attach the instance is reported as *DivaResultNoChannel*.

**Remarks**

Audio providers are used to process the audio signal in a separate instance, e.g., in a TTS processing instance. For more information on the architecture of audio providers, refer to the Dialogic® Diva® API documentation.

**See also**

DisconnectAudioProvider

## DisconnectAudioProvider

Removes an audio provider from the data channels of an existing call.

CalObject.ConnectAudioProvider ( ProviderName, Mode )

**Parameter**

*ProviderName*

*ProviderName* is a string parameter containing the name of the audio provider. This needs to be the same name the audio provider used to register with the Dialogic® Diva® API.

*Mode*

The parameter mode defines for which directions the streaming should be removed. For valid options, see DivaAudioProviderMode.

**Returns**

If the audio provider exists and the instances could be connected, the method returns *DivaResultSuccess*. If the audio provider name is not recognized, the method returns *DivaResultInvalidHandle*.

**Remarks**

Audio providers are used to process the audio signal in a separate instance, e.g., in a TTS processing instance. For more information on the architecture of audio providers, refer to the Dialogic® Diva® API documentation.

**See also**

ConnectAudioProvider

## ReceiveAudio

Receives audio data in the given format.

```
retVal = object.ReceiveAudio ( Format, DataLength, pData, pLengthReceived,
                                MaxSeconds )
```

**Parameter**

*Format*

The parameter *Format* specifies the audio format to be used. The raw formats of DivaAudioFmt except the ADPCM formats are supported.

*DataLength*

The parameter *DataLength* is an integer value that specifies the maximum amount of data to be stored in the receive buffer.

*pData*

The parameter *pData* specifies the location where to place the audio data. The application passes an array of bytes to be filled with the received data.

*pLengthReceived*

The parameter *pLengthReceived* is the location of a long value where the amount of bytes copied to the buffer is inserted.

*MaxSeconds* (optional)

The parameter *MaxSeconds* is a long value that specifies the maximum amount of time to wait for the received data. The parameter is optional. By default, there is no timeout.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method converts the received audio into the given format and copies it into the given data buffer.

The availability of data is signaled by the event *OnDataAvailable*, the amount of available data is given with the event. The event needs to be enabled using the *SignalEvent* property.

In asynchronous mode, the method returns right away, if no data is available the received length is set to zero. In synchronous mode, the application may control the time to wait using the parameter *MaxSeconds*. By default, the method waits without any timeout.

**Note:** This method is not available for Microsoft® Visual Basic® 6 applications.

**See also**

SendAudio, ReceiveData

## SendAudio

Sends audio signal in the given format.

retVal = object.SendAudio ( Format, DataLength, pData )

**Parameter**

*Format*

The parameter *Format* specifies the audio format to be used. The raw formats of DivaAudioFmt except the ADPCM formats are supported.

*DataLength*

The parameter *DataLength* is an integer value that specifies the amount of data to be sent.

*pData*

The parameter *pData* specifies the location of the audio data to be sent. The application passes the data as array of bytes.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method converts the given data according to the specified format and sends it to the remote side.

In asynchronous mode, the method returns right away. The data buffer will be locked until the data is sent. If events are enabled every time a data buffer is sent, the event *OnVoiceStreamed* is called.

In synchronous mode, the method blocks until the data has been sent. Note that calling *SendAudio* in synchronous mode may create small gaps on the line when the system cannot ensure that the next audio is available before the preceding buffer has been sent.

**Note:** This method is not available for Microsoft® Visual Basic® 6 applications.

**See also**

ReceiveAudio, SendData

## ClearDetectedTones

Removes any detected tones from the internal buffer.

object.ClearDetectedTones ( )

**Parameter**

None

**Returns**

None

**Remarks**

*ClearDetectedTones* deletes any detected tones from the internal buffer. Note that the property *EnableDTMFToneSplitting* needs to be set to separate processing of digits and tones. By default, tones and digits are stored in the digit buffer. This is a purely synchronous method.

**Example**

```
CallObj.ClearDetectedTones ()
```

**See also**

DetectedTones, EnableDTMFToneSplitting, ClearDetectedDigits, DetectedDigits

## EnableSingleToneDetector

Enables the tone detector for single tones.

VoiceCall.EnableSingleToneDetector ( MinDuration, MinSNR, MinLevel, MaxAM, MaxFM )

**Parameter**

*MinDuration*

The *MinDuration* parameter specifies the minimum duration of a tone before the detection is reported. The time is given in milliseconds.

*MinSNR*

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 parameter is optional, the default is 20 dB.

*MinLevel*

The *MinLevel* parameter specifies the minimum level of the detected signal. The value is specified in dB in the range of 128 dB to -128 dB. The parameter is optional, the default is -30 dB.

*MaxAM*

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 255 dB. The parameter is optional, the default is 1.

*MaxFM*

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. The parameter is optional, the default is 1.

**Returns**

If the function succeeds, the return value is *DivaResultSuccess* (0). Otherwise a corresponding *DivaResultCode* is returned.

**Remarks**

The method 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 implicitly stopped.

When a tone within the specified range is detected, the event handler *OnSingleToneDetected* is signaled. The frequency of the detected tone is passed to the event handler.

If the application is running in synchronous mode, the current operation is terminated with the return code *DivaResultToneDetected*. The information about the detected tone can be retrieved via the method *GetToneDetectorResults*.

**See also**

EnableDualToneDetector, DisableToneDetector, OnSingleToneDetected, OnDualToneDetected, GetToneDetectorResult

## EnableDualToneDetector

Enables the tone detector for dual tones.

VoiceCall.EnableDualToneDetector ( MinDuration, MinSNR, MinLevel, MaxDiffHighToLow, MaxDiffLowToHigh )

**Parameters**

*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 parameter is optional, the default is 20 dB.

*MinLevel*

[in] The *MinLevel* parameter specifies the minimum level of the detected signal. The value is specified in dB in the range of 128 to -128. The parameter is optional, the default is -30 dB.

*MaxDiffHighToLow*

[in] The *MaxDiffHighToLow* parameter specifies the maximum allowed difference in levels between the higher and the lower frequency tone. The value is specified in dB in the range of 127 to -127. The parameter is optional, the default is 10 dB.

*MaxDiffLowToHigh*

[in] The *MaxDiffLowToHigh* parameter specifies the maximum allowed difference in levels between the lower and higher frequency tone. The value is specified in dB in the range of 127 to -127. The parameter is optional, the default is 10 dB.

**Returns**

If the function succeeds, the return value is *DivaResultSuccess* (0). Otherwise a corresponding *DivaResultCode* is returned.

**Remarks**

The method validates that the requested tone can be detected. If successful, the tone detection is started. Any pending previously enabled generic tone detection is implicitly stopped.

When a matching dual tone is detected the event handler *OnDualToneDetected* is signaled. The frequencies of the detected tones are passed to the signal handler.

If the application is running in synchronous mode, the current operation is terminated with the return code *DivaResultToneDetected*. The information about the detected tone can be retrieved via the method *GetToneDetectorResults*.

**See also**

EnableSingleToneDetector, DisableToneDetector, OnSingleToneDetected, OnDualToneDetected, GetToneDetectorResult

## DisableToneDetector

Disables the tone detector for single or dual tones.

VoiceCall.DisableToneDetector ( )

**Parameters**

none

**Returns**

The return value is always *DivaResultSuccess* (0).

**Remarks**

The method stops any pending detector and clears the internal resources related to tone detection.

**See also**

EnableSingleToneDetector, OnSingleToneDetected, OnDualToneDetected, GetToneDetectorResult

## GetToneDetectorResult

The method retrieves an object containing the detailed tone detector results.

Dim DetectedTone as DivaSDKLib.DivaToneResult

DetectedTone = VoiceCall.GetToneDetectorResult ( )

**Parameters**

none

**Returns**

The method returns a reference to an object containing the properties of the detected tone. If no information are available, zero is returned.

**Remarks**

If information about a detected tone is available, an object of type *DivaSDKLib.DivaToneResult* is created and returned to the caller. The object contains the information about the tone, e.g., energy and signal to noise ratio. The caller is responsible to clean the object if no longer needed.

**See also**

EnableSingleToneDetector, DisableToneDetector, OnSingleToneDetected, OnDualToneDetected, GetToneDetectorResult

## EnableAMD

The method enables the answering machine detector.

Result = VoiceCall.EnableAMD ( MaxInitialSilence, MaxHumanTalkerTime, MaxInterSpeakerTimeout )

**Parameters**

*MaxInitialSilence*

[in] The *MaxInitialSilence* parameter specifies the maximum time in seconds until the remote side is expected to start speaking. When this timeout is reached without detecting a speaker, the answering machine detector terminates.

*MaxHumanTalkerTime*

[in] The *MaxHumanTalkerTime* parameter specifies the time in seconds 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, it will be interpreted as an answering machine message.

*MaxInterSpeakerTimeout*

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

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

With a call to *EnabelAMD* the application enables the analysis of the inbound audio stream for answering machine detection based on the length of the speech. The method is only valid for outgoing calls. This method is a purely synchronous method and stores the values for the answering machine detection, the processing starts when the call is connected.

The Diva Dialogic® Diva® SDK compares the length of the received speech with the given parameter. If the length of the announcements is below *MaxHumanSpeakerTime*, a human has answered the phone. If the length is above *MaxHumanSpeakerTime*, an automated system has answered. If no signal is received, the detector terminates when the *MaxInitialSilence* is reached.

In asynchronous mode, the result of the answering machine detector is reported via the event *OnAMDFinished*. The result is passed as a parameter to the event handler, for possible results refer to DivaAMDResult. This requires that the event reporting is enabled.

In synchronous mode, the currently blocked method will continue if a result of the answering machine detector is available. The result is signaled by the return code of the method. Typically this is the *Connect* method. However, applications may also enable the answering machine detector after the *Connect* method reports a connection and starts recording. In this case the record method will return the result of the answering machine detector. For possible detector results signaled as return code, see the DivaResultAMD... codes of *DivaResultcodes*.

**See also**

DisableAMD, OnAMDFinished

## DisableAMD

The method disables the answering machine detector.

Result = VoiceCall.EnableAMD ( MaxInitialSilence, MaxHumanTalkerTime, MaxInterSpeakerTimeout )

**Parameters**

none

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method disables the answering machine detector and clears the command for the answering machine detector. In asynchronous mode, the event *OnAMDFinished* is called. In synchronous mode, no blocked method is influenced by this call.

**See also**

EnableAMD, OnAMDFinished

## AMDClearRecordingTones

The method removes all tones specified via *AMDRecordingTone* from the internal list.

Call CallObj.AMDClearRecordingTones ()

**Parameters**

none

**Returns**

None

**Remarks**

The method removes all tone definitions for the answering machine beep from the internal list.

**See also**

AMDRecordingTone, EnableAMD, DisableAMD

### ResetCodecList

Each device object has a default codec list that is used for incoming and outgoing calls processed on this device if not overwritten on a per call base. The method *ResetCodecList* removes all codecs from this list.

**Parameter**

None

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns a corresponding error code.

**Remarks**

The method removes all default codecs for the call. Note if no codec is added before a call is initiated, accepted, or answered, the default codec list of the device object is used.

**See also**

AddCodec

### AddCodec

Each device object has a default codec list that is used for incoming and outgoing calls, if not overwritten on a per call base. Via *AddCodec* the application adds a codec to the default list.

Result = Device.AddCodec ( Codec, FrameSize, Direction )

**Parameter**

*Codec*

The *Codec* parameter specifies the codec to be added.  For valid codecs refer to DivaCodecs.

*FrameSize*

The *FrameSize* parameter is an integer value that specifies the frame size for the codec.

*Direction*

The *Direction* parameter specifies if the codec is used for transmit, receive, or for both. The type of the parameter is DivaDirections.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns a corresponding error code.

**Remarks**

This is a synchronous method that returns immediately. The codec is added to the internal codec list of the device object.

**See also**

ResetCodecList

## DivaCall Properties

The DivaCall component provides several properties to set certain parameters for a call or retrieve properties of a call. All properties need to be set to the *Connect* or *Answer* methods prior to the call in order to provide that they are valid for the call. Parameters set for an existing call are only used by the method *SetCallType*.

Properties of a call can be read at any time. For incoming calls, the first time the properties may be read is from the *OnIncomingCall* event.

This section contains the following DivaCall Properties:

- VoiceEnableEchoCanceller
- VoiceEchoCancellerActive
- EnableExtendedToneDetection
- EnableDigitDetection
- X25CalledAddress
- X25CallingAddress
- X25NCPI
- X25NCPIAsText
- TransferUseSameChannel
- TransferNoHold
- TransferCompleteOnAlerting
- TransferCompleteOnProceeding
- TransferUseCallingNumber
- TransferUseCallingName
- InputVolume
- OutputVolume
- RedirectNumber
- RedirectReason
- CalledNumberType
- CalledNumberId
- CallingNumberType
- CallingNumberId
- CallingNumberPresentation
- CallingNumberScreening
- EnableDTMFToneSplitting
- DetectedTones
- SingleToneDetectorMinFrequency
- SingleToneDetectorMaxFrequency
- AMDRecordingTone
- EnableTransactionRecording
- InputSamplingRate
- OutputSamplingRate

### SignalEvents

Enables or disables the signaling of events.

CallObj.SignalEvents = True

**Type**

Boolean

**Default value**

False

**Availability**

Read and write

**Remarks**

The *SignalEvents* property is available for reading and writing. On write, it enables or disables the signaling of events. On read, it returns the current state of the event signaling.

**Note:** This property does not change between synchronous and asynchronous operation mode.

**See also**

AsyncMode, Operation Modes

### AsyncMode

Sets the operation mode to synchronous or asynchronous.

CallObj.AsyncMode = True

**Type**

Boolean

**Default value**

False

**Availability**

Read and write

**Remarks**

The *AsyncMode* property is available for reading and writing. On write, it enables or disables the asynchronous mode. On read, it returns the current state of the operation mode. A value of false for the *AsyncMode* sets the operation mode to synchronous, which is the default. See the description of the Operation Modes for more information on these modes.

**Note:** This property does not change the signaling of events. This is selected by the *SignalEvents* property.

**See also**

SignalEvents, Operation Modes

### Device

Sets the line device that needs to be used for communication.

CallObj.Device = DivaListenDevicesAll

**Type**

Integer

**Default value**

DivaListenDevicesAll

**Availability**

Read and write

**Remarks**

The *Device* property is available for reading and writing. On write it selects the device to be used for an outgoing call. The devices available in the system are numbered by an index starting with 1. A value of zero means usage of any device.

On read the property provides the device selected for this call object.

**See also**

DivaDevice

### SignaledService

Retrieves the information about the service that is signaled for an incoming call.

Service = CallObj.SignaledService

**Type**

Integer

**Availability**

Read only

**Remarks**

The *SignaledService* property provides the service signaled for an incoming call. Possible options are:

- DivaSignaledServiceUnknown
- DivaSignaledServiceAnalog
- DivaSignaledServiceDigital
- DivaSignaledServiceGSM

**See also**

SignalService

### SignalService

The *SignalService* property sets the service to signal for an outgoing call.

CallObj.SignalService=DivaServiceSpeech

**Type**

Integer

**Default value**

Depending on the call type given by CallObj.Connect (…)

**Availability**

Write only

**Remarks**

The *SignalService* property sets the service to signal to the remote side. Options are:

- DivaSigServiceDigital
- DivaSigServiceGSM
- DivaSigServiceAnalog
- DivaSigServiceAudio3
- DivaSigServiceAudio7
- DivaSigServiceSpeech
- DivaSigServiceTelephony
- DivaSigServiceFaxG3

**See also**

SignaledService

### LocalNumber

The *LocalNumber* property sets the local number to be signaled for outgoing calls.

CallObj.LocalNumber = "08154711"

**Type**

String

**Default value**

""

**Availability**

Write only

**See also**

LocalSubAddress, CallingNumber, CalledNumber

### LocalSubAddress

The *LocalSubAddress* property sets the local sub address to be signaled for outgoing calls.

CallObj.LocalSubAdress = "1111"

**Type**

String

**Default value**

""

**Availability**

Write only

**See also**

LocalNumber, CallingNumber, CalledNumber

### CalledNumber

The *CalledNumber* property provides the dialed number for an incoming call.

Number = CallObj.CalledNumber

**Type**

String

**Default value**

“”

**Availability**

Read only

**Remarks**

The availability of the called number depends on the underlying telecommunication network.

**See also**

CallingNumber

### CallingNumber

The *CallingNumber* property provides the originator's number of an incoming call.

Number = CallObj.CallingNumber

**Type**

String

**Default value**

""

**Availability**

Read only

**Remarks**

The availability of the calling number depends on the underlying telecommunication network.

**See also**

CalledNumber

### Channel

The *Channel* property provides the used physical channel number for the call.

CurrentChannel = CallObj.Channel

**Type**

Integer

**Availability**

Read only

**See also**

No references

### DialingComplete

The *DialingComplete* property sets or retrieves the dialing complete state.

CallObj.DialingComplete = true

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

If the property is set before calling *CallObj.Connect*, the remote side is notified that no more dial information will follow. When read, the property provides the information for an incoming call if more dial information may be signaled. Note that the transfer of the dialing complete information depends on the underlying protocol.

**See also**

No references

### ServiceIndicator

This property provides the service indicator from the line information element.

```
Dim  Service as long

Service = CallObj.ServiceIndicator
```

**Type**

Boolean

**Default value**

False

**Availability**

Read only

**Remarks**

The service indicator is extracted from the line information element and provided via this property. This information is only available for the US protocol NI2.

### RxSpeed, TxSpeed

*RxSpeed* and *TxSpeed* provide the negotiated speed information.

Speed = CallObj.RxSpeed

**Type**

Integer

**Availability**

Read only

**Remarks**

The speed value may be different in transmit and receive direction. *TxSpeed* and *RxSpeed* are identical for protocols with symmetric speed values.

**See also**

No references

### DisconnectReason

Retrieves the information about the disconnect reason.

Reason = CallObj.DisconnectReason

**Type**

DivaDiscReason

**Availability**

Read only

**Remarks**

The *DisconnectReason* property provides information why a call is disconnected or could not be connected. Available reasons are:

- DivaDiscReasonUnspecified
- DivaDiscReasonUserInitiated
- DivaDiscReasonBusy
- DivaDiscReasonReject
- DivaDiscReasonNoAnswer
- DivaDiscReasonCableError
- DivaDiscReasonUnknownNumber
- DivaDiscReasonInvalidNumber
- DivaDiscReasonNoResource
- DivaDiscReasonNoDataChannel
- DivaDiscReasonNoFaxDevice
- DivaDiscReasonFaxTrainingFailed
- DivaDiscReasonFaxLocalAbort
- DivaDiscReasonFaxRemoteAbort

**See also**

No references

### DetectedDigits

The *DetectedDigits* property provides the detected DTMF digits.

Digits = CallObj.DetectedDigits

**Type**

String

**Default value**

""

**Availability**

Read only

**Remarks**

The property retrieves the detected digits from the internal buffer. The digits remain in the internal buffer until the application clears the buffer using the *ClearDetectedDigits* method. Digit and tone detection must be enabled.

**See also**

ClearDetectedDigits, EnableExtendedToneDetection, EnableDigitDetection

### FaxLocalId

The *FaxLocalId* property is used to set the local fax identification that is used for incoming and outgoing fax transfers.

CallObj.FaxLocalId = "+49 MyId"

**Type**

String

**Default value**

""

**Availability**

Write only

**See also**

FaxRemoteId

### FaxHeadLine

The *FaxHeadLine* property is used to set the head line printed on every page of an outgoing fax document.

CallObj.FaxHeadLine = "Sent by Diva SDK"

**Type**

String

**Default value**

""

**Availability**

Write only

**Remarks**

The fax headline printed on each page depends on the given information. If enough space is available, the date and time information as well as the current page is added automatically.

**See also**

No references

### FaxMaxSpeed

The *FaxMaxSpeed* property sets the maximum speed for a fax transfer.

CallObj.FaxMaxSpeed = DivaFaxSpeed14400

**Type**

DivaFaxSpeed

**Default value**

DivaFaxSpeedAutomatic

**Availability**

Write only

**Remarks**

The property limits the maximum speed for negotiating a fax transmission or reception. By default, the speed negotiation is automatic and will negotiate the highest possible value depending on the line quality. Applications may limit this for various reasons.

**See also**

FaxDisableHighResolution, FaxDisableECM, FaxDisableMR, FaxDisableMMR

### FaxEnablePolling

The *FaxEnablePolling* property enables the active or passive polling.

CallObj.FaxEnablePolling = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The property enables active or passive polling.

**See also**

SendFaxes

### FaxDisableHighResolution

The *FaxDisableHighResolution* property disables high resolution.

CallObj.FaxDisableHighResolution = False

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The property disables the high resolution format. Fax documents can be sent in various resolutions, by default the resolution is negotiated automatically. Applications may reduce the resolution to get shorter transmission times.

**Note:** If the document contains high resolution pages, they will be converted automatically to the negotiated resolution.

**See also**

FaxMaxSpeed, FaxDisableECM, FaxDisableMR, FaxDisableMMR

### FaxDisableECM

The *FaxDisableECM* property disables the error correction mode.

CallObj.FaxDisableECM = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The property disables the error correction mode. In general, the usage is negotiated automatically. Applications may restrict the use of the error correction mode.

**See also**

FaxMaxSpeed, FaxDisableHighResolution, FaxDisableMR, FaxDisableMMR

### FaxDisableMR

The *FaxDisableMR* property disables the fax compression.

CallObj.FaxDisableMR = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The property disables the modified read compression. If the property is enabled, only the standard fax compression is used. Any higher compression is disabled, even MMR. In general, the usage of the compression is negotiated automatically. Applications may restrict the usage. The data format of the received or transmitted files does not depend on this option.

**See also**

FaxMaxSpeed, FaxDisableHighResolution, FaxDisableECM, FaxDisableMMR

### FaxDisableMMR

The *FaxDisableMMR* property disables the error high level fax compression.

CallObj.FaxDisableMMR = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The property disables the Modified Modified Read compression (MMR). If this is disabled, only the standard fax or Modified Read (MR) compression is used. In general, the usage of the compression is automatically negotiated. Applications may restrict the use. The data format of the received or transmitted files does not depend on this option.

**See also**

FaxMaxSpeed, FaxDisableHighResolution, FaxDisableECM, FaxDisableMR

### FaxMultipleDocument

The *FaxMultipleDocument* property enables sending of multiple documents.

CallObj.FaxMultipleDocuments = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The property enables the sending of multiple fax documents over one single connection. If the application uses the method *SendFaxes*, this option needs to be set to the *Connect* method prior to the call.

**See also**

SendFaxes

### FaxReverseSession

The *FaxReverseSession* property changes the direction.

CallObj.FaxReverseSession = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The property changes the direction of the call. Fax connections are direction oriented, the initiator of a call sends the fax and the answerer of the call receives the fax. This property is used to change the direction. If a connection initiated as voice call, is changed to fax, the call direction has to change as well. A sample for this is a system where users dial in to receive a fax. When switching to fax mode, the direction changes as well.

**See also**

SendFax

### FaxRemoteId

*FaxRemoteId* provides the fax station identification of the remote fax.

RemoteId = CallObj.FaxRemoteId

**Type**

String

**Default value**

""

**Availability**

Read only

**Remarks**

The remote station is negotiated during the call progress and available when the connection is established. The information is not yet available with the event *OnIncomingCall*.

**See also**

FaxHighResolution, FaxMRActive, FaxMMRActive, FaxECMActive, FaxPollingActive, FaxPages

### FaxHighResolution

*FaxHighResolution* provides the information if high resolution is negotiated.

Resolution = CallObj.FaxHighResolution

**Type**

Boolean

**Default value**

False

**Availability**

Read only

**Remarks**

The vertical resolution is negotiated during the call progress and available when the connection is established. The information is not yet available with the event *OnIncomingCall*.

**See also**

FaxMRActive, FaxMMRActive, FaxECMActive, FaxPollingActive, FaxPages

### FaxMRActive

The *FaxMRActive* property provides the information if the Modified Read compression is used.

Compression = CallObj.FaxMRActive

**Type**

Boolean

**Default value**

False

**Availability**

Read only

**Remarks**

The compression is negotiated during the call progress and available when the connection is established. The information is not yet available with the event *OnIncomingCall*.

If both *FaxMRActive* and *FaxMMRActive* are false, the standard compression is used.

**See also**

FaxHighResolution, FaxMMRActive, FaxECMActive, FaxPollingActive, FaxPages

## FaxMMRActive

The *FaxMMRActive* property provides the information if Modified Modified Read compression is used.

Resolution = CallObj.FaxMMRActive

**Type**

Boolean

**Default value**

False

**Availability**

Read only

**Remarks**

The compression is negotiated during the call progress and available when the connection is established. The information is not yet available with the event *OnIncomingCall*.

If *FaxMMRActive* is false and the following *FaxMRActive* is false, the standard compression is used; see example.

```
Dim UsedCompression As String
If ( CallObject.FaxMMRActive = True ) Then
   UsedCompression = "Highest Compression"
Else If ( CallObject.FaxMRCompression = True ) Then
   UsedCompression = "Medium Compression"
Else
   UsedCompression = "Standard Compression"
End If
```

**See also**

FaxHighResolution, FaxMRActive, FaxECMActive, FaxPollingActive, FaxPages

## FaxECMActive

The *FaxECMActive* property provides the information if the error correction mode is active.

ECM = CallObj.FaxECMActive

**Type**

Boolean

**Default value**

False

**Availability**

Read only

**Remarks**

The error correction mode is negotiated during the call progress and available when the connection is established. The information is not yet available with the event *OnIncomingCall*.

**See also**

FaxHighResolution, FaxMRActive, FaxMMRActive, FaxPollingActive, FaxPages

## FaxPollingActive

The *FaxPollingActive* property provides the information if polling was negotiated.

Active = CallObj.FaxPollingActive

**Type**

Boolean

**Default value**

False

**Availability**

Read only

**Remarks**

Polling has to be enabled by the application using the *FaxEnablePolling* property. The polling mode is negotiated during the call progress and available when the connection is established. Based on the result, the application may send or receive a document.

**See also**

FaxHighResolution, FaxMRActive, FaxMMRActive, FaxEnablePolling, FaxPages

## FaxPages

*FaxPages* provides the information about the currently processed pages.

Pages = CallObj.FaxPages

**Type**

Integer

**Default value**

0

**Availability**

Read only

**Remarks**

The property provides the information about the currently sent or received amount of pages. After disconnection, the total number of pages processed is returned.

**See also**

FaxHighResolution, FaxMRActive, FaxMMRActive, FaxEnablePolling

### FaxEnableColor

The *FaxEnableColor* property enables the negotiation of the JPEG color fax option.

CallObj.FaxEnableColor = true

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The result of the negotiation needs to be checked when the call is connected. The property *FaxColorSelected* is true if the color fax option has been selected.

**See also**

FaxColorSelected

### FaxColorSelected

The *FaxColorSelected* property is true if the color option has been selected.

bColor = CallObj.FaxColorSelected

**Type**

Boolean

**Default value**

False

**Availability**

Read only

**Remarks**

If the color fax option is enabled by *FaxEnableColor*, this property provides the result of the negotiation. If this property returns true, the application needs to pass a color (JPEG) document to *SendFax*.

**See also**

FaxEnableColor

### FaxStoreMode

The property *FaxStoreMode* specifies how received fax pages are stored in files.

CallObj.FaxStoreMode = DivaFaxRxStorePerSession

**Type**

DivaFaxRxStoreModes

**Default**

DivaFaxRxStorePerSession

**Remarks**

By default, the received fax pages are stored in a single file. Via the property *FaxStoreMode*, JPEG color fax pages can be stored in separate files. For more information, refer to DivaFaxRxStoreModes.

**See also**

DivaFaxRxStoreModes, FaxEnableColor

### VoiceEnableEchoCanceller

The *VoiceEnableEchoCanceller* property enables the echo cancellation for a voice call.

CallObj.VoiceEnableEchoCanceller = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

This property enables the echo cancellation for a voice call.

**Note:** Echo cancellation needs DSP resources and may not run on the boards.

**See also**

VoiceEchoCancellerActive

### VoiceEchoCancellerActive

The *VoiceEchoCancellerActive* property shows if the echo canceller is active.

EchoActive = CallObj.VoiceEchoCancellerActive

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The echo cancellation needs to be enabled via *VoiceEnableEchoCanceller* before calling *Connect* or *Answer*.

**See also**

VoiceEnableEchoCanceller

### EnableExtendedToneDetection

Enables or disables the detection of extended tones.

CallObj.EnableExtendedToneDetection = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The *EnableExtendedToneDetection* property enables or disables the detection of extended tones. If enabled, detected tones are written to the internal digit buffer. If the application wants to receive events for detected tones, the *SignalEvents* property must be set as well.

For available Extended Tones, see DivaTones. Note that only some of the tones can be generated.

**See also**

SignalEvents

### EnableDigitDetection

Enables or disables the detection of DTMF digits.

CallObj.EnableDigitDetection = True

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The *EnableDigitDetection* property enables or disables the detection of DTMF digits and fax caller and calling tones. If enabled, detected digits are written to the internal digit buffer. If the application wants to receive events for detected tones, the *SignalEvents* property must be set as well.

**See also**

SignalEvents

### X25CalledAddress

Gets or sets the X.25 called address.

CallObj.X25CalledAddress = "1234"

**Type**

String

**Default value**

""

**Availability**

Read and write

**Remarks**

The property is only valid if the call type is set to *DivaCallType_X25*. When set, the information is used for the data channel connect of an outgoing X.25 call. When read, it returns the called address of an inbound X.25 call. The length of the X.25 called address is limited to 16 digits.

**See also**

X25CallingAddress, X25NCPI, X25NCPIAsText

### X25CallingAddress

Gets or sets the X.25 calling address.

CallObj.X25CallingAddress = "998"

**Type**

String

**Default value**

""

**Availability**

Read and write

**Remarks**

The property is only valid if the call type is set to *DivaCallType_X25*. When set, the information is used for the data channel connect of an outgoing X.25 call. When read, it returns the calling address of an inbound X.25 call. The length of the X.25 calling address is limited to 16 digits.

**See also**

X25CalledAddress, X25NCPI, X25NCPIAsText

### X25NCPI

Gets or sets the X.25 NCPI.

Dim MyNCPI as Byte(255)

MyNCPI = CallObj.X25NCPI

**Type**

Binary data, first byte length

**Default value**

Empty, length byte set to zero.

**Availability**

Read and write

**Remarks**

The property is only valid if the call type is set to *DivaCallType_X25*. The NCPI contains called and calling address, facilities, and user-user information.

When set, the information is used to specify the X.25 parameter for the data channel connect of an outgoing X.25 call. When read, it returns the plain NCPI containing the information from the remote peer.

Note that the exchange of binary data arrays is not supported by all platforms. If the platform does not support the binary exchange, the property *X25NCPIAsText* can be used.

**See also**

X25CalledAddress, X25CallingAddress, X25NCPIAsText

### X25NCPIAsText

Gets or sets the X.25 NCPI using a string to exchange the data.

CallObj.X25NCPIAsText = “”

**Type**

String

**Default Value**

“”

**Availability**

Read and write

**Remarks**

The property is only valid if the call type is set to *DivaCallType_X25*. The NCPI contains called and calling address, facilities, and user-user information.

The data is coded as hexadecimal using an ASCII string. The length of the data is given by the length of the string. When set, the information is used to specify the X.25 parameter for the data channel connect of an outgoing X.25 call. When read, it returns the plain NCPI containing the information from the remote peer.

**See also**

X25CalledAddress, X25CallingAddress, X25NCPI

### TransferUseSameChannel

The *TransferUseSameChannel* property specifies if the implicit consultation call of a blind call transfer is done on the same channel.

CallObj.TransferUseSameChannel = true

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

Depending on the used communication platform and the network behind, the call transfer properties are different. The application can overwrite the default for the parameter of a blind call transfer.

**See also**

TransferNoHold, TransferCompleteOnAlerting, TransferCompleteOnProceeding, BlindCallTransfer

### TransferNoHold

The *TransferNoHold* property specifies if the primary call is not put on hold before the consultation call is initiated.

CallObj.TransferNoHold = true

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

Depending on the used communication platform and the network behind the call transfer properties are different. The application can overwrite the default for the parameter for a blind call transfer.

**See also**

TransferUseSameChannel, TransferCompleteOnAlerting, TransferCompleteOnProceeding, BlindCallTransfer

### TransferCompleteOnAlerting

The *TransferCompleteOnAlerting* property specifies the condition for a completion of a call transfer initiated by the blind call transfer method.

CallObj.TransferCompleteOnAlerting = true

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

By default, the blind call transfer is completed when the implicit consultation call is connected. This condition can be changed to alerting or proceeding.

**See also**

TransferUseSameChannel, TransferCompleteOnProceeding, BlindCallTransfer

### TransferCompleteOnProceeding

The *TransferCompleteOnProceeding* property specifies the condition for a completion of a call transfer initiated by the blind call transfer method.

CallObj.TransferCompleteOnProceeding = true

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

By default, the blind call transfer is completed when the implicit consultation call is connected. This condition can be changed to alerting or proceeding.

**See also**

TransferUseSameChannel, TransferCompleteOnAlerting, BlindCallTransfer

### TransferUseCallingNumber

The *TransferUseCallingNumber* property specifies the usage of the calling number when transferring the call.

CallObj.TransferUseCallingNumber = true

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

By default, the Diva SDK selects the calling number for a blind call transfer based on information of the primary call. If the primary call is an incoming call, the called number is used as calling number of the implicit consultation call. If the calling number of the incoming call should be used, this property must be set. If using the method *CallDeflection* the property specifies that the number specified in the property LocalNumber should be used as calling number.

**See also**

BlindCallTransfer, CallDeflection

### TransferUseCallingName

The *TransferUseCallingName* property specifies that the calling name should be signaled to the transfer destination.

CallObj.TransferUseCallingName = true

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The property is only valid when using the method *CallDeflection*. If set the name set via the extended call property *CPT_CallingName* is signaled to the transfer destination.

**See also**

CallDeflection

### InputVolume

The *InputVolume* property specifies the volume for an outbound streaming.

CallObj.InputVolume = 10

CallObj.InputVolume = -4

**Type**

int

**Default value**

0

**Availability**

Write only

**Remarks**

The volume for streaming can be controlled by the application in the range of -18 to +18 db to reduce or increase the amplitude of the audio signal. A value of zero leaves the amplitude unchanged.

**See also**

OutputVolume

### OutputVolume

The *OutputVolume* property specifies the volume for an outbound streaming.

CallObj.OutputVolume = 10

CallObj.OutputVolume = -4

**Type**

int

**Default value**

0

**Availability**

Write only

**Remarks**

The volume for streaming can be controlled by the application in the range of -18 to +18 db to reduce or increase the amplitude of the audio signal. A value of zero leaves the amplitude unchanged.

**See also**

InputVolume

### RedirectNumber

The *RedirectNumber* property provides redirecting or redirected number.

Number = CallObj.RedirectNumber

**Type**

String

**Default value**

""

**Availability**

Read only

**Remarks**

The availability of the calling number depends on the underlying telecommunication network. Outgoing calls that are redirected, are redirected to this number. Incoming calls that are redirected contain the number of the redirecting extension. Note that this is not always the originally called extension.

**See also**

RedirectReason

### RedirectReason

The *RedirectReason* property provides reason for the redirection.

Reason = CallObj.RedirectReason

**Type**

DivaRedirectReasons

**Default value**

DivaRedirectUnknown

**Availability**

Read only

**Remarks**

The availability of the redirect reason depends on the underlying telecommunication network. For valid redirect reasons, refer to DivaRedirectReasons.

**See also**

RedirectNumber

### CalledNumberType

The *CalledNumberType* property sets or gets the called number type.

CallObj.CalledNumberType = DivaSDKLib.DivaNumberTypes.NumberTypeInternational

**Type**

DivaNumberTypes

**Default value**

DivaSDKLib.DivaNumberTypes.NumberTypeUnknown

**Availability**

Read and write

**Remarks**

The number type and identification is handled differently in certain networks. The SS7 network especially requires setting of this parameter. For valid number types, see DivaNumberTypes.

**See also**

CalledNumberId, CallingNumberType, CallingNumberId, CallingNumberPresentation, CallingNumberScreening

### CalledNumberId

The *CalledNumberId* property sets or gets the called number identification.

CallObj.CalledNumberId = DivaSDKLib.DivaNumberIdentifications.NumberIdISDNTelephony

**Type**

DivaNumberIdentification

**Default value**

DivaSDKLib.DivaNumberIdentification.NumberIsUnknown

**Availability**

Read and write

**Remarks**

The number type and identification is handled differently in certain networks. The SS7 network especially requires setting of this parameter. For valid number types, see DivaNumberIdentifications.

**See also**

CalledNumberType, CallingNumberType, CallingNumberId, CallingNumberPresentation, CallingNumberScreening

### CallingNumberType

The *CallingNumberType* property sets or gets the calling number type.

CallObj.CallingNumberType = DivaSDKLib.DivaNumberTypes.NumberTypeInternational

**Type**

DivaNumberTypes

**Default value**

DivaSDKLib.DivaNumberTypes.NumberTypeUnknown

**Availability**

Read and write

**Remarks**

The number type and identification is handled differently in certain networks. The SS7 network especially requires setting of this parameter. For valid number types, see DivaNumberTypes.

**See also**

CalledNumberType, CalledNumberId, CallingNumberId, CallingNumberPresentation, CallingNumberScreening

### CallingNumberId

The *CallingNumberId* property sets or gets the calling number type.

CallObj.CallingNumberId = DivaSDKLib.DivaNumberIdentifications.NumberIdISDNTelephony

**Type**

DivaNumberIdentifications

**Default value**

DivaSDKLib.DivaNumberIdentifications.NumberIsUnknown

**Availability**

Read and write

**Remarks**

The number type and identification is handled differently in certain networks. The SS7 network especially requires setting of this parameter. For valid number types, see DivaNumberIdentifications.

**See also**

CalledNumberType, CalledNumberId, CallingNumberType, CallingNumberPresentation, CallingNumberScreening

### CallingNumberPresentation

The *CallingNumberPresentation* property sets or gets the calling number presentation information.

CallObj.CallingNumberPresentation= DivaSDKLib.DivaNumberPresentations.NumberPresentationAllowed

**Type**

DivaNumberPresentations

**Default value**

DivaSDKLib.DivaNumberPresentations.NumberPresentationAllowed

**Availability**

Read and write

**Remarks**

The underlying media or network must support presentation settings. It is not guaranteed that a restricted presentation will avoid that the calling number is signaled to the remote end.

**See also**

CalledNumberType, CalledNumberId, CallingNumberId, CallingNumberId, CallingNumberScreening

### CallingNumberScreening

The *CallingNumberScreening* property sets or gets the calling number screening information.

CallObj.CallingNumberScreening= DivaSDKLib.DivaNumberScreenings.NumberScreeningUser

**Type**

DivaNumberScreenings

**Default value**

DivaSDKLib.DivaNumberScreenings.NumberScreeningUser

**Availability**

Read and write

**Remarks**

None.

**See also**

CalledNumberType, CalledNumberId, CallingNumberType, CallingNumberId, CallingNumberPresentation

### EnableDTMFToneSplitting

The *EnableDTMFToneSplitting* property enables the separate processing of DTMF digits and detected tones

```
CallObj. EnableDTMFToneSplitting = true
```

**Type**

Bool

**Availability**

Write only

**Remarks**

By default, detected DTMF digits and detected tones are processed within the same buffer. If an application requires an independent processing, the property *EnableDTMFToneSplitting* can be set. In this case, the DTMF digits are available via the property *DetectedDigits* and the tones via *DetectedTones*.

**See also**

DetectedTones, ClearDetectedTones, EnableExtendedToneDetection

### DetectedTones

The *DetectedTones* property provides the detected tones.

Tones = CallObj.DetectedTones

**Type**

String

**Default**

""

**Availability**

Read only

**Remarks**

The property retrieves the detected tones from the internal buffer. The tones remain in the internal buffer until the application clears the buffer using the *ClearDetectedTones* method. Tone detection must be enabled via *EnableExtendedToneDetection*.

By default, detected DTMF digits and detected tones are processed within the same buffer. The property *EnableDTMFToneSplitting* must be set to retrieve tones via this property.

**See also**

EnableDTMFToneSplitting, ClearDetectedTones, EnableExtendedToneDetection

### SingleToneDetectorMinFrequency

The *SingleToneDetectorMinFrequency* property specifies the minimum frequency a single tone needs to be signaled to the application.

```
CallObj. SingleToneDetectorMinFrequency = 1000
```

**Type**

long

**Default**

0 (no limitation)

**Remarks**

By default, the detected tones are signaled to the application. Via *SingleToneDetectorMinFrequency* the lower range of tones to be signaled can be specified.

**See also**

SingleToneDetectorMaxFrequency, EnableSingleToneDetector, OnSingleToneDetected

### SingleToneDetectorMaxFrequency

The *SingleToneDetectorMaxFrequency* property specifies the maximum frequency a single is allowed to have in order to be signaled to the application.

```
CallObj. SingleToneDetectorMaxFrequency = 1600
```

**Type**

long

**Default**

0 (no limitation)

**Remarks**

By default, the detected tones are signaled to the application. Via *SingleToneDetectorMaxFrequency* the upper range of tones to be signaled can be specified.

**See also**

SingleToneDetectorMinFrequency, EnableSingleToneDetector, OnSingleToneDetected

### AMDRecordingTone

Specifies a recording tone for the answering machine detector.

```
Dim Tone as DivaSDKLib.DivaCustomTone
Tone.Frequency = 1000
Tone.FrequencyVariation = 100
Tone.Duration = 500
Tone.DurationVariation = 100
CallObj.AMDRecordingTone = Tone
```

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

The answering machine detector stops once the detector has identified a human talker or an answering machine. At this time, the announcement of the answering machine may still be in progress. Applications that want to leave a message require that the result is reported when the answering machine enters the recording mode, which is the case after the beep tone. Via *AMDRecordingTone*, the application can specify one or more tones. The answering machine detector will indicate an event once the tone has been detected. In synchronous mode, the Connect method will finish when the beep has been detected. Multiple beep tones can be set by calling *AMDRecordingTone* multiple times. The application must enable the generic tone detector to allow beep tone detection.

**See also**

EnableAMD, EnableDualToneDetector, EnableSingleToneDetector, Connect

### EnableTransactionRecording

Enables the mixed recording of received and transmitted signal.

```
CallObj.EnableTransactionRecording = true
```

**Type**

Boolean

**Default value**

False

**Availability**

Write only

**Remarks**

By default, the recording functions of the Dialogic® Diva® Component API record only the received audio signal. If *EnableTransactionRecording* is set to true, the received and transmitted signals are mixed into one stream and recorded.

**See also**

RecordVoiceFile

### InputSamplingRate

CallObj.InputSamplingRate = 8000

**Default**

8000

**Remarks**

The property is write only and can be set at any time. The default sampling rate is 8000. The minimum supported sampling rate is 1250 and the maximum sampling rate is 51200.

### OutputSamplingRate

CallObj.OutputSamplingRate = 8000

**Default**

8000

**Remarks**

The property is write only and can be set at any time. The default sampling rate is 8000. The minimum supported sampling rate is 1250 and the maximum sampling rate is 51200.

## DivaCall Events

This section contains the following DivaCall Events:



### OnIncomingCall

The event *OnIncomingCall* is triggered when an incoming call is ringing.

CallObject_OnIncomingCall ( )

**Parameter**

None

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*.

**See also**

SignalEvents

### OnCallProgress

The event *OnCallProgress* is signaled when the call state proceeding or alerting is reached.

CallObject_ OnCallProgress ( State As Integer )

**Parameter**

*State*

**Remarks**

The parameter contains the new call state. Note that only the call states *DivaCallState_Ringing* and *DivaCallState_Proceeding* are signaled. All other call states, e.g., connected, are signaled via different events.

**See also**

SignalEvents, DivaCallState

### OnConnected

The event *OnConnected* is triggered when the call is established and data could be transferred.

CallObject_OnConnected ( )

**Parameter**

None

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*.

**See also**

SignalEvents

### OnDisconnected

The event *OnDisconnected* is triggered when the call is disconnected or cannot be connected.

CallObject_OnDisconnected ( )

**Parameter**

None

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*.

**See also**

SignalEvents

### OnToneReceived

The event *OnToneReceived* is triggered when a tone or digit is detected.

CallObject_OnToneReceived ( cTone as Char )

**Parameter**

*cTone*

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The tone detection must be enabled using the property *EnableExtendedToneDetection*. The DTMF detection must be enabled using the property *EnableDigitDetection*.

The received tone is signaled with this event. The tone also remains in the internal digit buffer. The ASCII chars '0' to '9',‘A’ to ‘D’, '*' and '#' are available for digits.

**See also**

SignalEvents, EnableDigitDetection, EnableExtendedToneDetection

### OnVoiceStreamed

The event *OnVoiceStreamed* is triggered when the audio streaming has finished.

CallObject_OnVoiceStreamed ( bWrapped as Boolean )

**Parameter**

*bWrapped*

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when the streaming, initiated by one of the streaming functions, has finished.

**Note:** The event is triggered when the audio is sent to the line.

If the parameter *bWrapped* is true, continuous play has been selected and the sending will restart automatically.

**See also**

SignalEvents, SendVoiceFile, SendVoiceFiles, SendVoiceFilesEx

### OnRecordEnded

The event *OnRecordEnded* is triggered when the audio recording has finished.

CallObject_OnRecordEnded ( Reason as DivaRecordEndReason )

**Parameter**

*Reason*

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when the recording, initiated by *RecordVoiceFile*, has finished.

The parameter *Reason* specifies the reason of the end, either the maximum duration has been reached or the maximum silence has been detected.

**See also**

SignalEvents, RecordVoiceFile

### OnFaxPageProcessed

The event *OnFaxPageProcessed* is triggered when the fax page has been sent or received.

CallObject_OnFaxPageProcessed ( Pages as Integer )

**Parameter**

*Pages*

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when a page has been sent or received.

The parameter *Pages* specifies the amount of pages actually sent or received.

**Note:** No page event is triggered for the last page. The last page and therefore the completion of the document is signaled by *OnFaxProcessed*.

**See also**

SignalEvents, OnFaxProcessed

### OnFaxProcessed

The event *OnPageProcessed* is triggered when the fax is successfully completed.

CallObject_OnFaxProcessed ( )

**Parameter**

None

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when a fax is successfully received or sent. If this event is not received before *OnDisconnected* is triggered, the fax transmission or reception failed.

**See also**

SignalEvents, OnFaxPageProcessed

### OnSuppServeCompleted

The event *OnSuppServeCompleted* is triggered when a supplementary service request is completed.

CallObject_OnSuppServeCompleted ( bSuccess As Boolean)

**Parameter**

*bSuccess*

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when the supplementary service request, e.g., call transfer is finished. The parameter *bSuccess* contains the result.

**See also**

SignalEvents, BlindCallTransfer, Hold, Retrieve

### OnDataAvailable

The event *OnDataAvailable* is triggered when the received data is ready for processing.

CallObject_OnDataAvailable ( nCount As Integer)

**Parameter**

*nCount*

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when the received data is available for a data connection.

**See also**

SignalEvents, ReceiveData

## OnSingleToneDetected

The event *OnSingleToneDetected* is signaled when a single tone is detected that matches the detector parameter.

VoiceCall_OnSingleToneDetected ( Frequency as Long )

**Parameter**

*Frequency*

Specifies the frequency of the detected tone.

**Remarks**

The signaling of events must be enabled using the property *SignalEvents* and the single tone detector must be enabled via *EnableSingleToneDetector*. The event is signaled when a single tone that matches the parameter specified with *EnableSingleToneDetector* is detected or ended. If the frequency is none zero, the tone has started. A frequency of zero signals that the tone ended.

The application may retrieve more information on the detect parameter via the object *DivaToneResult* that can be retrieved via the method *GetToneDetectorResult*. If events are enabled, the detector results are internally cleared when the next tone is received.

**See also**

EnableSingleToneDetector, EnableDualToneDetector, OnDualToneDetected, GetToneDetectorResult

## OnDualToneDetected

The event *OnDualToneDetected* is signaled when a dual tone is detected that matches the detector parameter.

VoiceCall_OnDualToneDetected ( FrequencyLow as Long, FrequencyHigh as Long )

**Parameter**

*FrequencyLow*
*FrequencyHigh*

**Remarks**

The signaling of events must be enabled using the property *SignalEvents* and the dual tone detector must be enabled via *EnableDualToneDetector*. The event is signaled when a dual tone that matches the parameter specified with *EnableDualToneDetector* is detected or ended. If the *FrequencyLow* and *FrequencyHigh* are none zero, the tone has started. A value of zero for both parameter signals that the tone ended.

The application may retrieve more information on the detected parameter via the object *DivaToneResult* that can be retrieved via the method *GetToneDetectorResult*. If events are enabled, the detector results are internally cleared when the next tone is received.

**See also**

EnableSingleToneDetector, EnableDualToneDetector, OnSingleToneDetected, GetToneDetectorResult

## OnAMDFinished

The event *OnAMDFinished* is signaled when the answering machine detector has finished the processing and the result is available.

VoiceCall_OnAMDFinished ( Result as DivaAMDResult )

**Parameter**

*Result*

**Remarks**

The signaling of events must be enabled using the property *SignalEvents* and the answering machine detector must be enabled via the method *EnableAMD*. The event is signaled when the answering machine detector has finished the processing and a result is available.

For more information and possible results for the answering machine detector, refer to *EnableAMD* and *DivaAMDResult*.

**See also**

EnableAMD, DivaAMDResult

## DivaCall Defines

This section contains the following DivaCall Defines:



### DivaAudioFmt

The Dialogic® Diva® Component API 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.

| Options | Value |
|---|---|
| DivaAudioDefault | 0 |
| DivaAudioAutoDetect | 1 |
| DivaAudioWav_aLaw8K8BitMono | 10 |
| DivaAudioWav_uLaw8K8BitMono | 11 |
| DivaAudioWav_PCM_8K8BitMono | 12 |
| DivaAudioWav_PCM_8K16BitMono | 13 |
| DivaAudioRaw_aLaw8K8BitMono | 100 |
| DivaAudioRaw_uLaw8K8BitMono | 101 |
| DivaAudioRaw_PCM_8K8BitMono | 102 |
| DivaAudioRaw_PCM_8K16BitMono | 103 |

| Options | Value |
|---|---|
| DivaAudioRaw_ADPCM_8K4BitMono | 104 |
| DivaAudioRaw_ADPCM_6K4BitMono | 105 |

*DivaAudioDefault*

This options is only available for recording. The default format is DivaAudioWav_PCM_8K8BitMono.

*DivaAudioAutoDetect*

This option is only available for sending wave files. The format is detected from the header of the file.

*DivaAudioWav_aLaw8K8BitMono*

The data is coded as 8 Bit a-law with an 8 KHz sampling rate. The storage format contains a wave file header.

*DivaAudioWav_uLaw8K8BitMono*

The data is coded as 8 Bit µ-law with an 8 KHz sampling rate. The storage format contains a wave file header.

*DivaAudioWav_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.

*DivaAudioWav_PCM_8K16BitMono*

The data is coded as 16 Bit PCM with an 8 KHz sampling rate. The storage format contains a wave file header.

*DivaAudioRaw_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.

*DivaAudioRaw_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.

*DivaAudioRaw_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.

*DivaAudioRaw_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.

*DivaAudioRaw_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.

*DivaAudioRaw_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® Diva® communication platform that supports 'extended voice'. This format is an adaptive format and can only be processed based on an audio file.

### DivaFaxFmt

The TIFF formats containing the Symmetric keyword are aligned to a symmetric resolution regardless with which horizontal resolution the document has been received.

```
typedef enum
{
        DivaFaxFmtAutodetect,
        DivaFaxFmtDefault,
        DivaFaxFmtTiffClassF,
        DivaFaxFmtTiffClassFSymetric,
        DivaFaxFmtSff,
        DivaFaxFmtTiffG3,
        DivaFaxFmtTiffG3Symetric,
        DivaFaxFmtTiffG4,
        DivaFaxFmtTiffG4Symetric,
        DivaFaxFmtColorJPEG,
} DivaFaxFmt;
```

*DivaFaxFmtAutoDetect*

The format is auto detected from the file. The format is only valid for outgoing faxes.

*DivaFaxFmtDefault*

The format is set to the default. The format is only for incoming faxes.

*DivaFaxFmtSff*

The data is coded according to the SFF format which is used as the internal format of the Dialogic® Diva® API and the CAPI interface.

*DivaFaxFmtTiffClassF*

The data is coded according to the TIFF Class F specification using RLE coding.

*DivaFaxFmtTiffG3*

The data is coded according to the TIFF Class F specification using G3 coding.

The TIFF formats containing the Symmetric keyword are aligned to a symmetric resolution regardless with which horizontal resolution the document has been received.

*DivaFaxFmtTiffG4*

The data is coded according to the TIFF Class F specification using G4 coding. This format takes less disk space.

*DivaFaxFmtColorJPEG*

The data is coded as JPEG according to the color fax specification.

### DivaCallState

The *DivaCallState* constants define the current state of a call.

```
typedef enum
{
        DivaCallState_Idle = 0
        DivaCallState_Listening,
        DivaCallState_Connecting,
        DivaCallState_Ringing,
        DivaCallState_Offering,
        DivaCallState_Alerting,
        DivaCallState_Connected,
        DivaCallState_OnHold,
        DivaCallState_Disconnecting,
        DivaCallState_Disconnected,
        DivaCallState_Proceeding,
} DivaCallState;
```

*DivaCallState_Idle*

The call is idle. An outgoing connection may be established.

*DivaCallState_Listening*

The call is in the listen state. Incoming calls will be reported to the application. Outgoing calls can also be serviced.

*DivaCallState_Connecting*

The call has been initiated or accepted and the call establishment is on progress. This may take some time depending on the call type.

*DivaCallState_Ringing*

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.

*DivaCallState_Offering*

The call has been signaled to one or more applications and not yet been answered. This call state is only available for incoming calls.

*DivaCallState_Alerting*

The incoming call is in the alerting state.

*DivaCallState_Connected*

The data channel is connected and data can be streamed in both directions.

*DivaCallState_OnHold*

The call is in hold state and no data channel is currently available.

*DivaCallState_Disconnecting*

Disconnect of the call is in progress.

*DivaCallState_Disconnected*

The call is disconnected. The application may get the disconnect reasons and other parameter via the DivaCall Properties.

*DivaCallState_Proceeding*

The call is proceeding.

## DivaCallTypes

```
typedef enum
{
        DivaCallType_Voice,
        DivaCallType_FaxG3,
        DivaCallType_Modem,
        DivaCallType_DigitalData,
        DivaCallType_X75,
        DivaCallType_V120,
        DivaCallType_GSM,
        DivaCallType_X25,
        DivaCallType_AutoDetect,
} DivaCallType;
```

*DivaCallType_Voice*

The call is processed as a voice call. The data channel is set to plain audio streaming according to G.711. The Dialogic® Diva® API handles a-law and µ-law coding for voice streaming functions. Outgoing calls with this call type are signaled to the switch as speech.

*DivaCallType_FaxG3*

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.

*DivaCallType_Modem*

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.

*DivaCallType_DigitalData*

The call is processed as a digital data call. The data channel is set to handle digital data. Plain HDLC is done. This provides 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.

*DivaCallType_X75*

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.

*DivaCallType_V120*

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.

*DivaCallType_GSM*

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.

*DivaCallType_X25*

The call is processed as a X.25 call. The data channel is set to handle digital data using the X.75 protocol in layer 2 and X.25 in layer 3. The X.25 parameter can be set by the properties *X25CalledAddress*, *X.25CallingAddress*, and *X25NCPI*. Outgoing calls with this call type are signaled to the switch as unrestricted digital information.

*DivaCallType_AutoDetect*

The call is processed as a digital call. The protocol is detected depending on the first received frame. Options to detect are V.120, plain digital data, and X.25.

### DivaListenServices

```
typedef enum
{
        DivaListenServiceNone,
        DivaListenServiceDigital,
        DivaListenServiceAnalogAll,
        DivaListenServiceAll,
} DivaCallType;
```

*DivaListenServiceNone,*

The listen is disabled, no incoming calls are signaled.

*DivaListenServiceDigital,*

Only calls that are signaled as digital calls are forwarded to the application.

*DivaListenServiceAnalogAll,*

Only calls that are signaled as analog calls are forwarded to the application.

*DivaListenServiceAll*

All incoming calls of any service type are signaled to the application.

### DivaRejectCode

```
typedef enum
{
        DivaAllowOthers = 1,
        DivaRejectNormalCallClearing = 2,
        DivaRejectUserBusy = 3,
        DivaRejectChannelNotAvailable,
        DivaRejectFacilityRejected,
        DivaRejectChannelNotAccepted,
        DivaRejectIncompatibleDestination,
        DivaRejectDestinationOutOfOrder,
        DivaRejectUserDefined = 0x3400,
} DivaRejectCode;
```

*DivaAllowOthers*

The call will be rejected by the application but other applications on the same line may accept this call.

*DivaRejectNormalCallClearing*

The call will be rejected. The reason signaled to the network is normal call clearing.

*DivaRejectUserBusy*

The call will be rejected. The reason signaled to the network is that the user is busy.

*DivaRejectChannelNotAvailable*

The call will be rejected. The reason signaled to the network is that the requested data channel is currently not available.

*DivaRejectFacilityRejected*

The call will be rejected. The reason signaled to the network is that one of the signal facilities could not be processed.

*DivaRejectChannelNotAccepted*

The call will be rejected. The reason signaled to the network is that the requested data channel is not accepted by the application.

*DivaRejectIncompatibleDestination*

The call will be rejected. The reason signaled to the network is that the destination is not compatible to the signaled call, e.g., a digital call to a phone that processes only analog and telephony services.

*DivaRejectDestinationOutOfOrder*

The call will be rejected. The reason signaled to the network is that the destination is currently out of order.

*DivaRejectUserDefined*

The call will be rejected. The reason signaled to the network is application-specific. The application sets the reason in the low byte, the upper byte must be 0x34. The coding of the reason must be according to the used network.

### DivaDiscReason

```
typedef enum
{
        DivaDiscReasonUnspecified = 1,
        DivaDiscReasonNormalClearing,
        DivaDiscReasonUserInitiated,
        DivaDiscReasonBusy,
        DivaDiscReasonReject,
        DivaDiscReasonNoAnswer,
        DivaDiscReasonCableError,
        DivaDiscReasonUnknownNumber,
        DivaDiscReasonInvalidNumber,
        DivaDiscReasonNumberChanged,
        DivaDiscReasonIncompatibleDest,
        DivaDiscReasonNoResource,
        DivaDiscReasonNoDataChannel,
        DivaDiscReasonNoFaxDevice,
        DivaDiscReasonFaxTrainingFailed,
        DivaDiscReasonFaxLocalAbort,
        DivaDiscReasonFaxRemoteAbort,
        DivaDiscReasonModemNegFailed,
        DivaDiscReasonModemNoAnswer,
        DivaDiscReasonModemCarrierLost,
        DivaDiscReasonIllegalData,
        DivaDiscReasonFileAccess,
} DivaDiscReason;
```

*DivaDiscReasonUnspecified*

There is no specific information why the call failed.

*DivaDiscReasonNormalCallClearing*

The call ended with the default cause.

*DivaDiscReasonUserInitiated*

The application initiated the disconnect of the call.

*DivaDiscReasonBusy*

The remote end is busy and could not take the call. This disconnect reason is only signaled for outgoing calls.

*DivaDiscReasonReject*

The call was rejected by the remote peer.

*DivaDiscReasonNoAnswer*

The remote end did not answer the call and the call timed out. This disconnect reason is only signaled for outgoing calls.

*DivaDiscReasonCableError*

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.

*DivaDiscReasonNumberUnknown*

The switch responds that the dialed number is not known.

*DivaDiscReasonInvalidNumber*

The switch responds that the dialed number is not in a valid format or not complete.

*DivaDiscReasonNumberChanged*

The switch responds that the dialed number is not known because it has changed.

*DivaDiscReasonIncompatibleDest*

The call is rejected because the destination is not compatible to the requested service (call type).

*DivaDiscReasonNoResource*

The system is out of resources and could not service the call. This is a locally generated reason.

*DivaDiscReasonNoDataChannel*

All channels on the line device are already in use. This reason is locally generated.

*DivaDiscReasonNoFaxDevice*

The call failed due to establishment of the data channel. The remote device is not responding as a fax device.

*DivaDiscReasonFaxTrainingFailed*

The call failed due to establishment of the data channel. The remote device is not responding as a fax device.

*DivaDiscReasonFaxLocalAbort*

The call failed due to establishment of the data channel. The fax protocol has been disconnected due to a local error.

*DivaDiscReasonFaxRemoteAbort*

The call failed due to establishment of the data channel. The fax protocol has been disconnected due to a remote error.

*DivaDiscReasonModemNegFailed*

The call failed due to establishment of the data channel. The modem negotiation failed. Reasons could be a bad line or different settings.

*DivaDiscReasonModemNoAnswer*

The call failed due to establishment of the data channel. The remote device does not respond like a modem.

*DivaDiscReasonModemCarrierLost*

The call is disconnected because the modem lost the carrier signal.

*DivaDiscReasonIllegalData*

The call failed due to data transmission reasons. The data to be transmitted has the wrong format.

*DivaDiscReasonFileAccess*

The file to be used for sending or receiving the data could not be accessed.

### DivaSignaledService

Defines the service signaled by the network for an incoming call.

```
typedef enum
{
        DivaSignaledServiceUnknown = 1,
        DivaSignaledServiceAnalog,
        DivaSignaledServiceDigital,
        DivaSignaledServiceGSM,
} DivaSignaledService;
```

*DivaSignaledServiceUnknown*

The service for the incoming call could not be determined.

*DivaSignaledServiceAnalog*

The call is signaled as an analog call. It might be a voice, fax, or modem call.

*DivaSignaledServiceDigital*

The call is signaled as a digital call.

*DivaSignaledServiceGSM*

The call is signaled as a digital call using GSM services.

### DivaSignalService

The enum *DivaSignalService* defines the service to be signaled to the network for an outgoing call.

```
typedef enum
{
        DivaSigServiceDigital = 1,
        DivaSigServiceGSM,
        DivaSigServiceAnalog,
        DivaSigServiceAudio3,
        DivaSigServiceAudio7,
        DivaSigServiceSpeech,
        DivaSigServiceTelephony,
        DivaSigServiceFaxG3,
} DivaSignalService;
```

*DivaSigServiceDigital*

The call is signaled as a purely digital call.

*DivaSigServiceGSM*

The call is signaled as a digital call using GSM services.

*DivaSigServiceAnalog*

The call is signaled as an analog call.

*DivaSigServiceAudio3*

The call is signaled as an audio call with 3.14 KHz.

*DivaSigServiceAudio7*

The call is signaled as an audio call with 7 KHz.

*DivaSigServiceSpeech*

The call is signaled with the capabilities set to speech.

*DivaSigServiceTelephony*

The call is signaled as ISDN telephony call.

*DivaSigServiceFaxG3*

The call is signaled as analog call carrying fax G3 data.

### DivaTones

```
typedef enum
{
        Diva_EndOfTone = 0x80,
        Diva_UnknownTone,
        Diva_DialTone,
        Diva_PBXInternalDialTone,
        Diva_SpecialDialTone,
        Diva_SecondDialTone,
        Diva_RingingTone,
        Diva_SpecialRingingTone,
        Diva_BusyTone,
        Diva_CongestionTone,
        Diva_SpecialInformationTone,
        Diva_ComfortNoise,
        Diva_HoldTone,
        Diva_RecordTone,
        Diva_CallerWaitingTone,
        Diva_CallWaitingTone,
        Diva_PayTone,
        Diva_PositiveIndicationTone,
        Diva_NegativeIndicationTone,
        Diva_WarningTone,
        Diva_IntrusionTone,
        Diva_CallingCardServiceTone,
        Diva_PayphoneRecognitionTone,
        Diva_CPEAlertingSignal,
        Diva_OffHookWarningTone,
        Diva_InterceptTone,
        Diva_ModemCallingTone,
        Diva_FaxCallingTone,
        Diva_AnswerTone,
        Diva_AnswerTonePhaseReversal,
        Diva_ANSam,
        Diva_ANSamPhaseReversal,
        Diva_Bell103AnswerTone,
        Diva_FaxFlags,
        Diva_FaxG2GroupId,
        Diva_HumanSpeech,
        Diva_ToneMF1 = 0xF1,
        Diva_ToneMF2,
        Diva_ToneMF3,
        Diva_ToneMF4,
        Diva_ToneMF5,
        Diva_ToneMF6,
        Diva_ToneMF7,
        Diva_ToneMF8,
        Diva_ToneMF9,
        Diva_ToneMF0,
        Diva_ToneMFStart = 0xFD,
        Diva_ToneMFStop = 0xFF,
} DivaTones;
```

## DivaNumberTypes

| Options | Value |
|---|---|
| NumberTypeUnkown | 0 |
| NumberTypeInternational | 1 |
| NumberTypeNational | 2 |
| NumberTypeNetwork | 3 |
| NumberTypeSubscriber | 4 |
| NumberTypeAbbreviated | 6 |

*NumberTypeUnkown*

The number type is not known. This is the default.

*NumberTypeInternational*

The number is coded as an international format.

*NumberTypeNational*

The number is called as national format.

*NumberTypeNetwork*

The type of number "network-specific number" is used to indicate administration/service number specific to the serving network, e.g., used to access an operator.

*NumberTypeSubscriber*

Subscriber provided number. Prefix or escape digits shall not be included.

*NumberTypeAbbreviated*

The number is abbreviated.

## DivaNumberIdentifications

| Options | Value |
|---|---|
| NumberIdUnknown | 0 |
| NumberIdISDNTelephony | 1 |
| NumberIdData | 3 |
| NumberIdNational | 8 |
| NumberIdPrivate | 9 |

*NumberIdUnknown*

The number identification is not known. This is the default.

*NumberIdISDNTelephony*

The number identification is ISDN telephony.

*NumberIdData*

The number identification is data.

*NumberIdNational*

The number is a national identification.

*NumberIdPrivate*

The number has a private identification.

### DivaNumberPresentations

| Options | Value |
|---|---|
| NumberPresentationAllowed | 0 |
| NumberPresentationRestricted | 1 |
| NumberPresentationNotAvailable | 2 |

*NumberPresentationAllowed*

The number presentation of the calling number is allowed.

*NumberPresentationRestricted*

The number presentation of the calling number is restricted and will be suppressed if supported by the switch.

*NumberPresentationNotAvailable*

The number presentation information is not available.

### DivaNumberScreenings

| Options | Value |
|---|---|
| NumberScreeningUser | 0 |
| NumberScreeningUserPassed | 1 |
| NumberScreeningUserFailed | 3 |
| NumberScreeningNetwork | 4 |

*NumberScreeningUser*

The calling party number has been provided by the caller and forwarded by the network without any validation.

*NumberScreeningUserPassed*

The calling party number has been provided by the caller and the verification by the network was successful.

*NumberScreeningUserFailed*

The calling party number has been provided by the caller but the verification by the network failed.

*NumberScreeningNetwork*

The calling party number has been provided by the network. Either the caller has not provided a number or the screening failed.

## DivaAudioProviderMode

| Options | Value |
|---|---|
| DivaAPModeReceive | 1 |
| DivaAPModeStream | 2 |
| DivaAPModeBoth | 3 |

*DivaAPModeReceive*

The audio received from the Dialogic® Diva® communication platform is given to the audio provider.

*DivaAPModeStream*

Data provided from the audio provider is streamed by the Diva communication platform.

*DivaAPModeBoth*

Audio signals are exchanged in both directions.

## DivaRedirectReasons

| Options | Value |
|---|---|
| DivaRedirectUnknown | 0 |
| DivaRedirectBusy | 1 |
| DivaRedirectNoReply | 2 |
| DivaRedirectCallDeflection | 4 |
| DivaRedirectDTEOutOfOrder | 9 |
| DivaRedirectByCalledDTE | 10 |
| DivaRedirectUnconditional | 15 |

*DivaRedirectUnknown*

The reason for the redirection is unknown.

*DivaRedirectBusy*

The call is redirected because the extension was busy.

*DivaRedirectNoReply*

The call is redirected because the original called destination did not reply.

*DivaRedirectCallDeflection*

The call is redirected by a call deflection.

*DivaRedirectDTEOutOfOrder*

The call is redirected due to an out of service condition of the called destination.

*DivaRedirectByCalledDTE*

The call is redirected by the called endpoint.

*DivaRedirectUnconditional*

The call is redirected unconditionally.

## DivaRecordEndReason

| Options | Value |
|---|---|
| DivaRecordEndReason_TimeReached | 1 |
| DivaRecordEndReason_Silence | 2 |
| DivaRecordEndReason_Unspecified | 3 |
| DivaRecordEndReason_ToneDetected | 4 |

*DivaRecordEndReason_TimeReached*

The recording is terminated because the maximum time for recording to a file is reached.

*DivaRecordEndReason_Silence*

The recording is terminated because the maximum silence time during recording to a file is reached.

*DivaRecordEndReason_Unspecified*

There is no specific reason for the termination of the recording. The application has terminated the recording or the call is disconnected.

*DivaRecordEndReason_ToneDetected*

The recording is terminated due to a DTMF tone. The application has specified this DTMF tone as stop tone.

## DivaSendVoiceEndReason

| Options | Value |
|---|---|
| DivaSendVoiceEndReason_Undefined | 0 |
| DivaSendVoiceEndReason_Cancelled | 1 |
| DivaSendVoiceEndReason_Disconnected | 2 |
| DivaSendVoiceEndReason_Restarted | 3 |

*DivaSendVoiceEndReason_Undefined*

The reason for stopping the streaming is unknown.

*DivaSendVoiceEndReason_Cancelled*

The sending has been terminated by the application via StopSending.

*DivaSendVoiceEndReason_Disconnected*

The call is disconnected and the sending has been stopped implicitly.

*DivaSendVoiceEndReason_Restarted*

The sending has been restarted. This is typically done due to a continuous sending condition.

## DivaCPT

| Property Name | Value | Mode | Type |
|---|---|---|---|
| CPT_BearerCapabilities | 4 | Set / Get | Byte array |
| CPT_SignaledLineDiscReason | 14 | Get | long |
| CPT_VoiceEarlyDataChannel | 17 | Set | bool |
| CPT_SecondCallingNumber | 19 | Get | long |
| CPT_CallingName | 21 | Set / Get | BSTR |
| CPT_ConnectedName | 22 | Get | BSTR |
| CPT_CallingSubAddress | 23 | Set / Get | BSTR |
| CPT_CalledSubAddress | 24 | Set / Get | BSTR |
| CPT_OriginalCalledNumber | 25 | Get | BSTR |
| CPT_ConnectedNumber | 26 | Get | BSTR |
| CPT_CalledName | 27 | Get | BSTR |
| CPT_DisconnectReason | 29 | Set | long |
| CPT_DisconnectCause | 30 | Set | long |
| CPT_RedirectionNumber | 31 | Get | BSTR |
| CPT_VoiceRecordStartTones | 100 | Set | Byte array |
| CPT_VoiceDTMF_SendDuration | 101 | Set | long |
| CPT_VoiceDTMF_SendPause | 102 | Set | long |
| CPT_VoiceDTMF_DetectDuration | 103 | Set | long |
| CPT_VoiceDTMF_DetectPause | 104 | Set | long |
| CPT_VoiceEarlyDataDiscOnInfo | 106 | Set | bool |
| CPT_EchoCancellerEnableNLP | 107 | Set | bool |
| CPT_EchoCancellerAutoDisable1 | 108 | Set | bool |
| CPT_EchoCancellerAutoDisable2 | 109 | Set | bool |
| CPT_EchoCancellerTailLength | 110 | Set | long |
| CPT_EchoCancellerPreDelay | 111 | Set | long |
| CPT_EnableDTMFTrailingEdge | 112 | Set | bool |
| CPT_EnableTransparentLI | 122 | Set | bool |
| CPT_HumanTalkerThreshold | 123 | Set | long |
| CPT_VoiceActivityThreshold | 124 | Set | long |
| CPT_VoiceDTMF_TxLevelGroup | 125 | Set | long |
| CPT_SingleToneOffDuration | 126 | Set | long |
| CPT_DualToneOffDuration | 127 | Set | long |
| CPT_FaxPageQuality | 212 | Get | long |
| CPT_FaxPageEndInfo | 213 | Get | long |
| CPT_FaxRemoteFeatures | 214 | Get | Byte array |
| CPT_FaxRemoteMaxHorzRes | 215 | Get | long |
| CPT_FaxRemoteMaxVertRes | 216 | Get | long |
| CPT_FaxRemoteMaxSpeed | 217 | Get | long |

| Property Name | Value | Mode | Type |
|---|---|---|---|
| CPT_FaxRemoteNSF | 218 | Get | Byte array |
| CPT_FaxLocalNSF | 219 | Set | Byte array |
| CPT_EnableInterrupt | 222 | Set | bool |
| CPT_RequestInterrupt | 223 | Set | bool |
| CPT_FaxProcedureInterrupt | 224 | Get | bool |
| CPT_FaxEnableSecurity | 225 | Set | bool |
| CPT_FaxRemoteSupportsSubaddr | 226 | Get | bool |
| CPT_FaxRemoteSupportsPassword | 227 | Get | bool |
| CPT_FaxSignalSubAddress | 228 | Set | BSTR |
| CPT_FaxSignalPassword | 229 | Set | BSTR |
| CPT_FaxRemoteSubAddress | 230 | Get | BSTR |
| CPT_FaxRemotePassword | 231 | Get | BSTR |
| CPT_FaxAllowDocumentStretching | 234 | Set | bool |
| CPT_FaxRemoteScanLineLength | 235 | Get | long |
| CPT_MaximumSpeed | 400 | Set | long |
| CPT_DataBits | 401 | Set / Get | long |
| CPT_StopBits | 402 | Set / Get | long |
| CPT_Parity | 403 | Set / Get | long |
| CPT_DisableCompression | 800 | Set | bool |
| CPT_DisableV42 | 801 | Set | bool |
| CPT_DisableMNP | 802 | Set | bool |
| CPT_ForceReliable | 803 | Set | bool |
| CPT_DisableRetrain | 804 | Set | bool |
| CPT_ModulationClass | 805 | Set | long |
| CPT_NegotiatedV42V42bis | 806 | Get | bool |
| CPT_NegotiatedMNP4MNP5 | 807 | Get | bool |
| CPT_NegotiatedTransparent | 808 | Get | bool |
| CPT_NegotiatedCompression | 809 | Get | bool |
| CPT_DCD | 810 | Get | bool |
| CPT_CTS | 811 | Get | bool |
| CPT_ConnectedNorm | 812 | Get | long |
| CPT_RoundTripDelay | 813 | Get | long |
| CPT_GuardTone | 814 | Set | bool |
| CPT_ModemLeasedLine | 815 | Set | bool |
| CPT_Modem4WireOption | 816 | Set | bool |
| CPT_DisableDiscOnBusyTone | 817 | Set | bool |
| CPT_DisableCallingTone | 818 | Set | bool |
| CPT_DisableAnswerTone | 819 | Set | bool |
| CPT_EnableDialToneDetect | 820 | Set | bool |
| CPT_DisableStepUp | 821 | Set | bool |

| Property Name | Value | Mode | Type |
|---|---|---|---|
| CPT_DisableStepDown | 822 | Set | bool |
| CPT_DisableSplitSpeed | 823 | Set | bool |
| CPT_EnableShortAnswerTone | 824 | Set | bool |
| CPT_DisableFlushTimer | 825 | Set | bool |
| CPT_EnableEmptyFrames | 826 | Set | bool |
| CPT_EnableMultimoding | 827 | Set | bool |
| CPT_BypassControl | 828 | Set | bool |
| CPT_DisableModulationV21 | 829 | Set | bool |
| CPT_DisableModulationV22 | 830 | Set | bool |
| CPT_DisableModulationV22bis | 831 | Set | bool |
| CPT_DisableModulationV23 | 832 | Set | bool |
| CPT_DisableModulationV32 | 833 | Set | bool |
| CPT_DisableModulationV32bis | 834 | Set | bool |
| CPT_DisableModulationV34 | 835 | Set | bool |
| CPT_DisableModulationV90DPCM | 836 | Set | bool |
| CPT_DisableModulationBell103 | 837 | Set | bool |
| CPT_DisableModulationBell212A | 838 | Set | bool |
| CPT_DisableModulationAllFS | 839 | Set | bool |
| CPT_DisableModulationK56Flex | 840 | Set | bool |
| CPT_DisableModulationX2 | 841 | Set | bool |
| CPT_DisableV42Detection | 842 | Set | bool |
| CPT_EnableModulationV29FDX | 843 | Set | bool |
| CPT_EnableModulationV33 | 844 | Set | bool |
| CPT_EnableModulationV90APCM | 845 | Set | bool |
| CPT_EnableModulationV22FS | 846 | Set | bool |
| CPT_EnableModulationV29FS | 847 | Set | bool |
| CPT_EnableModulationV23_1 | 848 | Set | bool |
| CPT_EnableModulationV23_2 | 849 | Set | bool |
| CPT_MinimumTxSpeed | 850 | Set | long |
| CPT_MaximumTxSpeed | 851 | Set | long |
| CPT_MinimumRxSpeed | 852 | Set | long |
| CPT_MaximumRxSpeed | 853 | Set | long |
| CPT_TxLevelAdjust | 854 | Set | bool |
| CPT_DisableV34TxLevelReduction | 855 | Set | bool |
| CPT_DisableV34PreCoding | 856 | Set | bool |
| CPT_DisableV34PreEmphasis | 857 | Set | bool |
| CPT_DisableV34Shaping | 858 | Set | bool |
| CPT_DisableV34NonLinearEncoding | 859 | Set | bool |
| CPT_DisableV34ManualReduction | 860 | Set | bool |
| CPT_DisableV34Training16Point | 861 | Set | bool |

| Property Name | Value | Mode | Type |
|---|---|---|---|
| CPT_DisableV34SymbolRate2400 | 862 | Set | bool |
| CPT_DisableV34SymbolRate2743 | 863 | Set | bool |
| CPT_DisableV34SymbolRate2800 | 864 | Set | bool |
| CPT_DisableV34SymbolRate3000 | 865 | Set | bool |
| CPT_DisableV34SymbolRate3200 | 866 | Set | bool |
| CPT_DisableV34SymbolRate3429 | 867 | Set | bool |
| CPT_ForceReliableIfV34 | 868 | Set | bool |
| CPT_DisableSDLC | 869 | Set | bool |
| CPT_DisableReliableIf1200 | 870 | Set | bool |
| CPT_BufferDuringV42Detection | 871 | Set | bool |
| CPT_DisableV42SelectivReject | 872 | Set | bool |
| CPT_DisableMNP3 | 873 | Set | bool |
| CPT_DisableMNP4 | 874 | Set | bool |
| CPT_DisableMNP10 | 875 | Set | bool |
| CPT_TransparentModeIfV22bis | 876 | Set | bool |
| CPT_TransparentModeIfV32bis | 877 | Set | bool |
| CPT_BreakMode | 878 | Set | bool |
| CPT_ModemEarlyConnect | 879 | Set | bool |
| CPT_ModemPassIndication | 880 | Set | bool |
| CPT_SDLCLinkAddress | 881 | Set | bool |
| CPT_SDLCModuloMode | 882 | Set | bool |
| CPT_SDLCWindowSize | 883 | Set | bool |
| CPT_SDLCXID | 884 | Set | bool |
| CPT_SDLCReverseEstablishment | 885 | Set | bool |
| CPT_V18Selected | 886 | Set | bool |
| CPT_V18ProbingSequence | 887 | Set | bool |
| CPT_V18CountryProbingSequence | 888 | Set | bool |
| CPT_V18ProbingMessage | 889 | Set | bool |
| CPT_V18ReinitializeOnSilence | 890 | Set | bool |
| CPT_V18RevertToAnswerMode | 891 | Set | bool |
| CPT_V18DisconnectOnBusy | 892 | Set | bool |
| CPT_V18AutomodingMonitorMode | 893 | Set | bool |
| CPT_V18TextProbingForCarrierMode | 894 | Set | bool |
| CPT_V18TXPSpaceParityInOrigMode | 895 | Set | bool |
| CPT_V18EnableV18OriginationMode | 896 | Set | bool |
| CPT_V18EnableV18AnswerMode | 897 | Set | bool |
| CPT_V18EnableV21OriginationMode | 898 | Set | bool |
| CPT_V18EnableV21AnswerMode | 899 | Set | bool |
| CPT_V18EnableBell103OrigMode | 900 | Set | bool |
| CPT_V18EnableBell103AnswerMode | 901 | Set | bool |

| Property Name | Value | Mode | Type |
|---|---|---|---|
| CPT_V18EnableV23OriginationMode | 902 | Set | bool |
| CPT_V18EnableV23AnswerMode | 903 | Set | bool |
| CPT_V18EnableEDTMode | 904 | Set | bool |
| CPT_V18EnableBAUDOT45Mode | 905 | Set | bool |
| CPT_V18EnableBAUDOT47Mode | 906 | Set | bool |
| CPT_V18EnableBAUDOT50Mode | 907 | Set | bool |
| CPT_V18EnableDTMFMode | 908 | Set | bool |
| CPT_V18TransmitLevel | 909 | Set | long |
| CPT_V18AsyncFormatV21 | 910 | Set | long |
| CPT_V18AsyncFormatV23 | 911 | Set | long |
| CPT_V18AsyncFormatBell103 | 912 | Set | long |
| CPT_V18AsyncFormatEDT | 913 | Set | long |
| CPT_V18AsyncFormatBAUDOT | 914 | Set | long |
| CPT_V18TimerTcTimeout | 915 | Set | long |
| CPT_V18TimerTmTimeout | 916 | Set | long |
| CPT_V18CleanCarrierTime | 917 | Set | long |
| CPT_V18EchoSupressTime | 918 | Set | long |
| CPT_EnableModulationV22bisFS | 919 | Set | bool |
| CPT_EnableModulationBell202CID | 937 | Set | bool |
| CPT_EnableModulationBell202POS | 938 | Set | bool |
| CPT_EnableModulationBell103SIA | 939 | Set | bool |
| CPT_EnableModulationV21Bits10 | 940 | Set | bool |
| CPT_EnableModulationV23reverse | 941 | Set | bool |
| CPT_B1Protocol | 1200 | Set | long |
| CPT_B2Protocol | 1201 | Set | long |
| CPT_B3Protocol | 1202 | Set | long |
| CPT_B1Configuration | 1203 | Set | Byte array |
| CPT_B2Configuration | 1204 | Set | Byte array |
| CPT_B3Configuration | 1205 | Set | Byte array |
| CPT_LLC | 1206 | Set / Get | Byte array |
| CPT_HLC | 1207 | Set / Get | Byte array |
| CPT_B_ChannelInfo | 1208 | Set | Byte array |
| CPT_KeypadFacility | 1209 | Set / Get | Byte array |
| CPT_UserUserInfo | 1210 | Set / Get | Byte array |
| CPT_FacilityDataArray | 1211 | Set / Get | Byte array |
| CPT_DisplayInfo | 1212 | Get | Byte array |
| CPT_TotalChargeUnits | 1213 | Get | long |
| CPT_SpecialInfoElement | 1214 | Set / Get | Byte array |
| CPT_ChannelInfoElement | 1215 | Get | Byte array |
| CPT_ProgressIndElement | 1216 | Get | Byte array |

| Property Name | Value | Mode | Type |
|---|---|---|---|
| CPT_GlobalConfiguration | 1217 | Set | long |
| CPT_ReverseDataChannelConnect | 1218 | Set | bool |
| CPT_CauseInfoElement | 1219 | Get | Byte array |
| CPT_LineInfoElement | 1222 | Set | byte array |
| CPT_X25_ReverseRestart | 2003 | Set | bool |
| CPT_AutoDetectMode | 2400 | Set | long |
| CPT_AutoDetectX75ForceX25 | 2401 | Set | bool |
| CPT_AutoDetectMaxFrames | 2402 | Set | long |
| CPT_AutoDetectMaxSeconds | 2401 | Set | long |
| CPT_NoAnswerTimeout | 4000 | Set | long |
| CPT_ConnectTimeout | 4001 | Set | long |

*CPT_BearerCapabilities*

*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.

*CPT_SignaledLineDiscReason*

The property is read only and returns the disconnect reason in the format signaled by the line.

*CPT_VoiceEarlyDataChannel*

The property 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*.

*CPT_SecondCallingNumber*

The properties are read only and provide the information about a second calling party number. A second calling party number may be signaled by SMS gateways.

*CPT_CallingName*

The parameter is read only. On read, it provides the calling name for an incoming call. On write, it allows to set the name for an outgoing call. The availability of the name depends on the underlying network.

*CPT_ConnectedName*

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.

*CPT_CallingSubAddress*

The property provides the calling party address signaled on an incoming call or sets the calling party address for an outgoing call.

*CPT_CalledSubAddress*

The property provides the called party address signaled on an incoming call or sets the called party address for an outgoing call.

*CPT_OriginalCalledNumber*

*CPT_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.

*CPT_ConnectedNumber*

*CPT_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.

*CPT_CalledName*

*CPT_CalledName* is a read only parameter and specifies the name of the endpoint that answered the call.

*CPT_DisconnectReason*

*CPT_DisconnectReason* is a write only property to set the disconnect reason. For valid disconnect reasons, see *DivaActiveDiscReasons* in the Dialogic® Diva® API documentation. 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.

*CPT_DisconnectCause*

*CPT_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.

*CPT_RedirectionNumber*

This is a read only parameter that provides the redirecting number for an incoming call.

*CPT_VoiceRecordStartTones*

The property 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*.

*CPT_VoiceDTMF_SendDuration*

The property is write only and specifies the duration and pause of generated DTMF tones.

*CPT_VoiceDTMF_SendPause*

The property is write only and specifies the duration and pause of generated DTMF tones.

*CPT_VoiceDTMF_DetectDuration*

The property is write only and specifies the duration and pause for DTMF tone detection. The property must be set prior to the call to *DivaReportDTMF*.

*CPT_VoiceDTMF_DetectPause*

The properties are write only and specify the duration and pause for DTMF tone detection. The properties must be set prior to the call to *DivaReportDTMF.*

*CPT_VoiceEarlyDataDiscOnInfo*

*CPT_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.

*CPT_EchoCancellerEnableNLP*

The property is write only and enables the non-linear processing for the echo canceller.

*CPT_EchoCancellerAutoDisable1*

The property is write only and bypasses the echo canceller upon detection of phase reversed 2100 Hz (operation according to G.165)

*CPT_EchoCancellerAutoDisable2*

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).

*CPT_EchoCancellerTailLength*

The property is write only. Echo canceller time span in ms, default is implementation-specific.

*CPT_EchoCancellerPreDelay*

The property is write only. Echo canceller pre-delay before starting.

*CPT_EnableDTMFTrailingEdge*

The property is a write only and enables the reporting of the training edge of a DTMF tone. The default is disabled.

*CPT_EnableTransparentLI*

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).

*CPT_HumanTalkerThreshold*

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.

*CPT_VoiceActivityThreshold*

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.

*CPT_VoiceDTMF_TxLevelGroup*

The property is write only and sets the transmit level for DTMF tones. The level can be set in the range of -124 to +127.

*CPT_SingleToneOffDuration*

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 of 32 to 8000 milliseconds.

*CPT_DualToneOffDuration*

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 of 32 to 8000 milliseconds.

*CPT_FaxPageQuality*

*PT_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* in the Dialogic® Diva® API documentation.

*CPT_FaxPageEndInfo*

*CPT_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* in the Dialogic® Diva® API documentation.

*CPT_FaxRemoteFeatures*

*CPT_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.

*CPT_FaxRemoteMaxHorzRes*

*CPT_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.

*CPT_FaxRemoteMaxVertRes*

*CPT_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.

*CPT_FaxRemoteMaxSpeed*

*DivaCPT_FaxRemoteMaxSpeed* is a read only property and provides the maximum speed the receiving fax station can support. Note that this is not the finally negotiated speed because this depends on the line quality.

*CPT_FaxRemoteNSF*

*CPT_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.

*CPT_FaxLocalNSF*

*CPT_FaxLocalNSF* is a write only property and specifies the non standard facilities to be sent to the remote fax station. The data is expected as binary data, first byte length field.

*CPT_EnableInterrupt*

The property is write only and enables the fax procedure interrupt. The usage is depending on the remote peer. The property *CPT_FaxProcedureInterrupt* returns the result.

*CPT_RequestInterrupt*

The property is write only and requests the fax procedure interrupt. The usage is depending on the remote peer. The property *CPT_FaxProcedureInterrupt* returns the result.

*CPT_FaxProcedureInterrupt*

The property is read only and returns the state of the procedure interrupt negotiation. The property can only be negotiated if the property *CPT_RequestInterrupt* or *CPT_FaxProcedureInterrupt* are enabled.

*CPT_FaxEnableSecurity*

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.

*CPT_FaxRemoteSupportsSubaddr*

The properties are read only and provide the information if the remote party is able to handle secure fax protocols.

*CPT_FaxRemoteSupportsPassword*

The properties are read only and provide the information if the remote party is able to handle secure fax protocols.

*CPT_FaxSignalSubAddress*

The properties are write only and specify the sub-address and password to be sent to the remote side within the Fax T.30 negotiation.

*CPT_FaxSignalPassword*

The properties are write only and specify the sub address and password to be sent to the remote side within the Fax T.30 negotiation.

*CPT_FaxRemoteSubAddress*

The properties are read only and provide the sub address and password or the remote party negotiated during fax T.30 negotiation.

*CPT_FaxRemotePassword*

The properties are read only and provide the sub address and password or the remote party negotiated during fax T.30 negotiation.

*CPT_FaxAllowDocumentStretching*

If this option is selected, a TIFF document provided with a resolution that is half of the next matching fax format will be stretched. For example, a document with a resolution of 800 pixels per line will be stretched to 1600 pixel per line and centered on the next matching resolution of 1728 pixel per line.

*CPT_FaxRemoteScanLineLength*

*DivaCPT_FaxRemoteScanLineLength* is a read only property and provides the maximum scan line length the receiving fax station can support. The value is given as DivaScanLineMax.

*CPT_MaximumSpeed*

*CPT_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 *CPT_TxSpeed* and *CPT_RxSpeed* properties.

*CPT_DataBits*

The properties are read and write and set / get the framing for an asynchronous connection.

*CPT_StopBits*

The properties are read and write and set / get the framing for an asynchronous connection.

*CPT_Parity*

The properties are read and write and set / get the framing for an asynchronous connection.

*CPT_DisableCompression*

The property is write only and disables any compression for an analog modem connection.

*CPT_DisableV42*

The properties are write only and disable any V.42 or MNP negotiation for an analog modem connection.

*CPT_DisableMNP*

The properties are write only and disable any V.42 or MNP negotiation for an analog modem connection.

*CPT_ForceReliable*

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.

*CPT_DisableRetrain*

The property is write only and disables the retrain for an analog modem connection.

*CPT_ModulationClass*

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*.

*CPT_NegotiatedV42V42bis*

The properties are read only and valid only for analog modem connections. If the property is set, the negotiation succeeds in the specified reliable protocol. If *CPT_NegotiatedCompression* is also set, the corresponding compression, V.42bis or MNP5, is also negotiated.

*CPT_NegotiatedMNP4MNP5*

The properties are read only and valid only for analog modem connections. If the property is set, the negotiation succeeds in the specified reliable protocol. If *CPT_NegotiatedCompression* is also set, the corresponding compression, V.42bis or MNP5, is also negotiated.

*CPT_NegotiatedTransparent*

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.

*CPT_NegotiatedCompression*

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.

*CPT_DCD*

The properties are read only and valid only for the call type modem. *CPT_DCD* reflects the state of the DCD modem signal and *CPT_CTS* reflects the state of the CTS signal.

**Note:** The CTS signal is only provided if the call type is modem and any of the extended modem settings have been enabled.

*CPT_CTS*

The properties are read only and valid only for the call type modem. *CPT_DCD* reflects the state of the DCD modem signal and *CPT_CTS* reflects the state of the CTS signal.

**Note:** The CTS signal is only provided if the call type is modem and any of the extended modem settings have been enabled.

*CPT_ConnectedNorm*

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.

*CPT_RoundTripDelay*

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.

*CPT_GuardTone*

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.

*CPT_ModemLeasedLine*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_Modem4WireOption*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableDiscOnBusyTone*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableCallingTone*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableAnswerTone*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableDialToneDetect*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableStepUp*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableStepDown*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableSplitSpeed*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableShortAnswerTone*

The property is write only. Enabling the short answer tone for low bit rate connections reduces the time until the connection is detected.

*CPT_DisableFlushTimer*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableEmptyFrames*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableMultimoding*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_BypassControl*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationV21*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationV22*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationV22bis*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationV23*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationV32*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationV32bis*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationV34*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationV90DPCM*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationBell103*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationBell212A*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationAllFS*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationK56Flex*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableModulationX2*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV42Detection*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableModulationV29FDX*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableModulationV33*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableModulationV90APCM*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableModulationV22FS*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableModulationV29FS*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableModulationV23_1*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableModulationV23_2*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_MinimumTxSpeed*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_MaximumTxSpeed*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_MinimumRxSpeed*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_MaximumRxSpeed*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_TxLevelAdjust*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34TxLevelReduction*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34PreCoding*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34PreEmphasis*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34Shaping*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34NonLinearEncoding*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34ManualReduction*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34Training16Point*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34SymbolRate2400*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34SymbolRate2743*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34SymbolRate2800*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34SymbolRate3000*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34SymbolRate3200*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV34SymbolRate3429*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_ForceReliableIfV34*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableSDLC*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableReliableIf1200*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_BufferDuringV42Detection*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableV42SelectivReject*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableMNP3*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableMNP4*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_DisableMNP10*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_TransparentModeIfV22bis*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_TransparentModeIfV32bis*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_BreakMode*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_ModemEarlyConnect*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_ModemPassIndication*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_SDLCLinkAddress*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_SDLCModuloMode*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_SDLCWindowSize*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_SDLCXID*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_SDLCReverseEstablishment*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_V18Selected*

The property is write only and enables the V.18 mode.

*CPT_V18ProbingSequence*

Text of the message string used for probing. The property is write only.

*CPT_V18CountryProbingSequence*

Pre-defined country probing sequences. For available countries, refer to DivaV18DefProbings. The property is write only.

*CPT_V18ProbingMessage*

Array of bytes containing the sequence of modulation norm identifiers that specifies the order used in answer mode probing. The property is write only.

*CPT_V18ReinitializeOnSilence*

The property is write only and enables or disables the reinitialization on silence.

*CPT_V18RevertToAnswerMode*

The property is write only and enables or disables the revert to answer mode on timeout.

*CPT_V18DisconnectOnBusy*

The property is write only and enables or disables disconnect on busy detection.

*CPT_V18AutomodingMonitorMode*

The property is write only and enables or disables automoding monitor.

*CPT_V18TextProbingForCarrierMode*

The property is write only and enables or disables continuous carrier probing with the message.

*CPT_V18TXPSpaceParityInOrigMode*

The property is write only and enables or disables the sending of TXP with space parity in origination mode.

*CPT_V18EnableV18OriginationMode*

The property is write only and enables the V.18 originate mode (CI/TXP procedure, V.21 data state, TX: 980/1180 Hz 300 bps, RX: 1650/1850 Hz 300 bps).

*CPT_V18EnableV18AnswerMode*

The property is write only and enables V.18 answer mode (CI/TXP procedure, V.21 data state, TX: 1650/1850 Hz 300 bps, RX: 980/1180 Hz 300 bps).

*CPT_V18EnableV21OriginationMode*

The property is write only and enables V.21 originate mode (TX: 980/1180 Hz 300 bps, RX: 1650/1850 Hz 300 bps).

*CPT_V18EnableV21AnswerMode*

The property is write only and enables V.21 answer mode (TX: 1650/1850 Hz 300 bps, RX: 980/1180 Hz 300 bps).

*CPT_V18EnableBell103OrigMode*

The property is write only and enables Bell 103 originate mode (TX: 1270/1070 Hz 300 bps, RX: 2225/2025 Hz 300 bps).

*CPT_V18EnableBell103AnswerMode*

The property is write only and enables Bell 103 answer mode (TX: 2225/2025 Hz 300 bps, RX: 1270/1070 Hz 300 bps).

*CPT_V18EnableV23OriginationMode*

The property is write only and enables V.23 originate mode (TX: 390/450 Hz 75 bps, RX: 1300/1700 Hz 1200 bps).

*CPT_V18EnableV23AnswerMode*

The property is write only and enables V.23 originate mode (TX: 390/450 Hz 75 bps, RX: 1300/1700 Hz 1200 bps).

*CPT_V18EnableEDTMode*

The property is write only and enables EDT mode (980/1180 Hz 110 bps).

*CPT_V18EnableBAUDOT45Mode*

The property is write only and enables BAUDOT 45 mode (1800/1400 Hz 22 ms/bit).

*CPT_V18EnableBAUDOT47Mode*

The property is write only and enables BAUDOT 47 mode (1800/1400 Hz 21 ms/bit).

*CPT_V18EnableBAUDOT50Mode*

The property is write only and enables BAUDOT 50 mode (1800/1400 Hz 20 ms/bit).

*CPT_V18EnableDTMFMode*

The property is write only and enables DTMF mode (DTMF 50ms on / 50ms off).

*CPT_V18TransmitLevel*

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.

*CPT_V18AsyncFormatV21*

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

*CPT_V18AsyncFormatV23*

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.

*CPT_V18AsyncFormatBell103*

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

*CPT_V18AsyncFormatEDT*

Asynchronous data format used in EDT mode. The coding is described in DivaV18Framing. Value 0 will set the default. The property is write only.

*CPT_V18AsyncFormatBAUDOT*

Asynchronous data format used in BAUDOT modes. The coding is described in DivaV18Framing. Value 0 will set the default. The property is write only.

*CPT_V18TimerTcTimeout*

Timeout time for ITU-T V.18 timer Tc in ms. (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.

*CPT_V18TimerTmTimeout*

Timeout time for ITU-T V.18 timer Tm in ms. (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.

*CPT_V18CleanCarrierTime*

Time span in ms 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.

*CPT_V18EchoSupressTime*

Time span in ms 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.

*CPT_EnableModulationV22bisFS*

The property is write only. For more information, refer to the CAPI extensions in the document CxModem.pdf.

*CPT_EnableModulationBell202CID*

The property is write only and enables the modem modulation Bell 202 CID.

*CPT_EnableModulationBell202POS*

The property is write only and enables the modem modulation Bell 202 POS.

*CPT_EnableModulationBell103SIA*

The property is write only and enables the modem modulation Bell 103 SIA.

*CPT_EnableModulationV21Bits10*

The property is write only and enables the modem modulation V.21 Bits 10.

*CPT_EnableModulationV23reverse*

The property is write only and enables the modem modulation V.23 reverse.

*CPT_B1Protocol*

The property is write only. The B1 protocol according to the CAPI 2.0 specification.

*CPT_B2Protocol*

The property is write only. The B2 protocol according to the CAPI 2.0 specification.

*CPT_B3Protocol*

The property is write only. The B3 protocol according to the CAPI 2.0 specification.

*CPT_B1Configuration*

The property is write only. The B1 configuration options according to the CAPI 2.0 specification.

*CPT_B2Configuration*

The property is write only. The B2 configuration options according to the CAPI 2.0 specification.

*CPT_B3Configuration*

The property is write only. The B3 configuration options according to the CAPI 2.0 specification.

*CPT_LLC*

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.

*CPT_HLC*

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.

*CPT_B_ChannelInfo*

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.

*CPT_KeypadFacility*

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.

*CPT_UserUserInfo*

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 is passed in the alert message.

*CPT_FacilityDataArray*

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.

*CPT_DisplayInfo*

The property is read only and reads the display information received from an incoming call.

*CPT_TotalChargeUnits*

The property is read only and provides the amount of charge units reported by the network.

*CPT_SpecialInfoElement*

The property is used for setting specific elements.

*CPT_ChannelInfoElement*

The property is read only and provides the channel information element as received from the line.

*CPT_ProgressIndElement*

The property is read only and provides the progress information element as received from the line.

*CPT_GlobalConfiguration*

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.

*CPT_ReverseDataChannelConnect*

The parameter is write only and specifies that the data channel connection is not initiated by the side that has initiated the physical connection.

*CPT_CauseInfoElement*

*DivaCPT_CauseInfoElement* is a read only property and provides the cause information element as signaled on the underlying network.

*CPT_LineInfoElement*

The property is read only and returns the line info element. The line info element is supported by US protocols such as NI2 and includes the service indicator. Applications that require only the service indicator may use the property ServiceIndicator of DivaCall.

*CPT_X25_ReverseRestart*

*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.

*CPT_AutoDetectMode*

The property is reserved for future use.

*CPT_AutoDetectX75ForceX25*

The property is write only. If specified, the autodetect mode for digital protocols interprets X.25 frames in layer 2 as X.25 connections.

*CPT_AutoDetectMaxFrames*

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.

*CPT_AutoDetectMaxSeconds*

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*.

*CPT_NoAnswerTimeout*

The property is write only and specifies the amount of time in seconds to wait until the remote side picks up the call.

*CPT_ConnectTimeout*

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.

## DivaFaxRxStoreModes

| Options | Value |
|---|---|
| DivaFaxRxStorePerSession | 0 |
| DivaFaxRxStorePerDocument | 1 |
| DivaFaxRxStorePerPage | 2 |

*DivaFaxRxStorePerSession*

All pages of the complete fax session are stored in one file. This is the default.

*DivaFaxRxStorePerDocument*

This option is reserved for future use.

*DivaFaxRxStorePerPage*

Each page of a fax reception is stored in 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.

## DivaScanLineMax

| Options | Value |
|---|---|
| DivaScanLineUnknown | 0 |
| DivaScanLineMax215 | 1 |
| DivaScanLineMax255 | 2 |
| DivaScanLineMax303 | 3 |

*DivaScanLineUnknown*

The remote side did not provide the scan line capabilities.

*DivaScanLineMax215*

The remote fax is able to handle scan lines of 215 millimeters. This corresponds to the ISO A4 format.

*DivaScanLineMax255*

The remote fax is able to handle scan lines of 255 millimeters. This corresponds to the ISO B4 format.

*DivaScanLineMax303*

The remote fax is able to handle scan lines of 303 millimeters. This corresponds to the ISO A3 format.

## DivaAMDResult

| Options | Value |
|---|---|
| DivaAMDUserTerminated | 0 |
| DivaAMDHumanTalker | 1 |
| DivaAMDAnsweringMachine | 2 |
| DivaAMDAnsweringMachineTone | 3 |
| DivaAMDSilence | 4 |
| DivaAMDFaxOrModem | 5 |
| DivaAMDAnsweringMachineEnd | 6 |

*DivaAMDUserTerminated*

The application has cancelled the answering machine detector.

*DivaAMDHumanTalker*

A human talker has been detected by the answering machine detector.

*DivaAMDAnsweringMachine*

An answering machine has answered the call.

*DivaAMDAnsweringMachineTone*

An answering machine has been detected by the special answering machine tone.

*DivaAMDSilence*

The detection timed out due to silence from the called side.

*DivaAMDFaxORModem*

A fax or modem has answered the call.

*DivaAMDAnsweringMachineEnd*

The answering machine detector has detected one of the defined tones that mark the start of the recording.

## DivaCall Result Codes

The following table lists the result codes and their description.

**Note:** A result code other than *DivaSuccess* does not automatically point to a failure. Several functions return states and reasons with the result code, e.g., the voice recording returns the reason why recording ended.

| DivaCall Result Codes | |
|---|---|
| **Name** | **Description** |
| DivaResultSuccess | The method succeeded. |
| DivaResultInvalidParameter | The parameters passed to the method are not valid. |
| DivaResultInvalidState | The call to this method is not valid in this state, e.g., call to a streaming function while not connected. |
| DivaResultInvalidFunction | The requested function is invalid. The reason may be a write operation on a read only property. |
| DivaResultOutOfMemory | The required memory could not be allocated from the system. |
| DivaResultSystemError | The underlying system environment is not ready for processing. |
| | |
| DivaResultDestBusy | The remote side is busy. |
| DivaResultNoAnswer | The remote side did not answer the call. |
| DivaResultUnspecific | An outgoing call failed for an unspecified reason. |
| DivaResultUnallocatedNumber | The number used for an outgoing call is no longer valid. |
| DivaResultAnotherAppGotThatCall | An incoming call could not be answered because another application was also listening and got the call. |
| DivaResultNoChannel | There is no channel available. All channels of the selected line device are in use. |
| DivaResultOpenFile | The file required for the operation could not be opened. This happens, when data is sent from a file and this file is not available. If receiving data to a file, the file could not be created. |
| DivaResultUnsupportedFormat | The requested format is not supported for the audio or fax processing. |
| DivaResultReadFile | The read operation for the file failed. |
| DivaResultTimeout | The reason for the termination was a timeout. Refer to the methods that use this result code for more information. |
| DivaResultNotSupported | The requested operation is not supported. |
| DivaResultTimeReached | The operation has finished because the maximum time limit has been reached. |
| DivaResultToneDetected | The operation has finished because a tone has been detected. |
| DivaResultSilenceDetected | The operation has finished because silence has been detected. |
| DivaResultDisconnected | The operation has finished because the call has been disconnected. The reason for disconnecting can be obtained with the property *DisconnectReason*. |
| DivaResultNoData | There is no data available to be retrieved by the application. |
| DivaResultAMDHumanTalker | A human talker has been detected by the answering machine detector. |
| DivaResultAMDAnsweringMachine | The answering machine detector has detected that an answering machine has answered the call. |
| DivaResultAMDAnsweringMachineTone | The special answering machine tone signaled the answering machine detector that an answering machine has been detected. |
| DivaResultAMDSilence | The answering machine detection timed out due to silence from the called side. |
| DivaResultAMDFaxOrModem | The answering machine detector has detected that a fax or modem has answered the call. |
| DivaResultAMDRecordStarted | The answering machine detector detected one of the tones that mark the start to record a message. |

# CHAPTER 7

# DivaSystem References

DivaSystem provides information on the installed hardware and creates instances of DivaInstance. The usage of DivaSystem is optional. Applications that do not need system information for call processing may use DivaCall directly.

The DivaSystem Methods are described below. For DivaSystem properties, see DivaSystem Properties on page 134.

## DivaSystem Methods

This section contains the following DivaSystem Methods:



### GetDevice

Gets a DivaDevice object.

DeviceObject = SystemObject.GetDevice ( nDevice )

**Parameter**

*nDevice*

The parameter *nDevice* specifies the device to get the object for.

**Returns**

The method returns a pointer to the requested object or zero if an error occurs.

**Remarks**

The method returns a pointer to an object of type DivaDevice. The device is defined by the parameter *nDevice*. The value must be in the range from 1 to the maximum amount of devices returned by the property *NumDevices*.

**See also**

DivaDevice, NumDevices

### CreateInstance

Creates a DivaInstance object with specific parameter.

InstanceObject = SystemObject.CreateInstance (bBlocking, MaxConnections, BuffersPerConnect, MaxDataSize )

**Parameter**

*bBlocking*

The parameter *bBlocking* specifies the default value for the operation mode of call objects created on the instance.

*MaxConnections*

The parameter *MaxConnections* specifies maximum amount of calls to be created on this object.

*BuffersPerConnect*

The parameter *BuffersPerConnect* specifies the number of used buffers. The value should be between 2 and 10, both inclusive. The default is 4.

*MaxDataSize*

The parameter *MaxDataSize* specifies the maximum buffer size.

**Returns**

The method returns a pointer to the requested object or zero if an error occurs.

**Remarks**

The method returns a pointer to an object of type DivaInstance. Objects of DivaInstance and DivaCall could be created directly. If the application needs specific parameters for buffer and resource management, the *CreateInstance* method provides them.

The buffer size may be optimized for performance using a large buffer of up to 2Kbytes or for delay on audio calls using a smaller buffer size.

**See also**

DivaInstance

### LoadAudioProvider

Loads an audio provider DLL.

Result = SystemObject.LoadAudioProvider ( Filename )

**Parameter**

*Filename*

The parameter *Filename* specifies the audio provider DLL to be loaded.

**Returns**

The method returns *DivaResultSuccess* if the DLL is successfully loaded. If the DLL could not be found, the result *DivaResultOpenFile* is returned.

**Remarks**

The method loads the given DLL into the current process space. If the DLL requires initialization, it must export a function called *InitializeAudioProvider*. If this function is available, it will be called once the DLL is loaded.

**See also**

FreeAudioProvider, ConnectAudioProvider, DisconnectAudioProvider

### FreeAudioProvider

Unloads a previously loaded audio provider DLL.

Result = SystemObject.FreeAudioProvider ( Filename )

**Parameter**

*Filename*

The parameter Filename specifies the audio provider DLL to be unloaded.

**Returns**

The method returns *DivaResultSuccess* if the DLL is successfully unloaded. If the DLL is not loaded, the result *DivaResultInvalidParameter* is returned.

**Remarks**

The methods checks if the given DLL is loaded. If the DLL is loaded and a function named *TerminateAudioProvider* is exported, this function is called. Then, the DLL is released.

**See also**

LoadAudioProvider, ConnectAudioProvider, DisconnectAudioProvider

## SetTraceFilename

Changes the trace file name for tracing.

SystemObject.SetTraceFilename ( Filename, MaxSize )

**Parameter**

*Filename*

The parameter Filename specifies the new name of the trace file.

*MaxSize (optional)*

The parameter MaxSize specifies the maximum file size in Kbytes. A value of zero indicates no limitation. The default is zero.

**Returns**

none

**Remarks**

If a trace file exists, the file is closed. The new file is opened, if the file exists the data is appended.

**See also**

TraceLevel

## CreateDevice

Creates an object of type DivaDevice to create and configure a virtual IP-based device.

DeviceObject = SystemObject.CreateDevice ()

**Parameter**

None

**Returns**

The method returns a new object of type DivaDevice or zero if an error occurs.

**Remarks**

To configure IP-based line devices via the application the object DivaDevice is used. The method *CreateDevice* creates a new object that allows setting the device-specific IP parameter, e. g., signaling protocol. For details on DivaDevice, refer to the description of the object.

**See also**

DivaDevice References

## Initialize

Explicitly starts the initialization process.

**Parameter**

None

**Returns**

The method returns *DivaResultSuccess* if the initialization was successfully finished. Other potential results are *DivaResultNoDevice* or *DivaResultOutOfMemory*.

**Remarks**

The method starts the initialization process explicitly. This allows the application to retrieve the result of the initialization. Note that for compatibility reasons, the initialization process is implicitly done if any method or property of DivaSystem is called that requires the initialization.

**See also**

None

## DivaSystem Properties

This section contains the following DivaSystem properties:

- NumDevices
- TotalChannels
- TraceLevel

### NumDevices

The *NumDevices* property returns the number of installed devices.

nDevices = SystemObj.NumDevices

**Type**

Integer

**Default value**

None

**Availability**

Read only

**Remarks**

The property returns the number of installed devices. Please note that each line of a physical hardware represents one device.

**See also**

TotalChannels

### TotalChannels

The *TotalChannels* property returns the number of data channels of all installed devices.

nChannels = SystemObj.TotalChannels

**Type**

Integer

**Default value**

None

**Availability**

Read only

**Remarks**

For information on channels per device refer to *DivaDevice*.

**See also**

NumDevices

### TraceLevel

Sets the trace level of SDK tracing.

CallObj.TraceLevel = 6

**Type**

Long

**Default Value**

1

**Availability**

Write only

**Remarks**

The property is used to set the amount of trace information written to the trace file. A value of 0 disables the writing of trace information. A value of 1 writes error messages.

**See also**

SetTraceFilename

# CHAPTER 8

## DivaInstance References

DivaInstance can be used by applications to create call objects with preset properties. In addition call objects created on basis of DivaInstance are optimized for resource usage. The usage of DivaInstance is optional. Applications may directly use DivaCall. DivaInstance Methods are described below. DivaInstance properties are described in DivaInstance Properties on page 140.

## DivaInstance Methods

This section contains the following DivaInstance Methods:

- CreateCall
- CreateConference
- MWIActivate
- MWIDeactivate
- MWIReport
- GetMWIIndication

### CreateCall

Creates a DivaCall object with properties of the instance.

CallObject = InstanceObject.CreateCall ( )

**Parameter**

none

**Returns**

The method returns a pointer to the requested object or zero if an error occurs.

**Remarks**

The method returns a pointer to an object of type DivaCall. The new object has the properties set for DivaInstance. These are the properties, set when the instance was created. The properties can be modified on the call object.

**See also**

DivaCall

### CreateConference

Creates a DivaConference object based on the instance properties.

ConferenceObject = object1.CreateConference ( )

**Parameter**

none

**Returns**

The method returns a pointer to the requested object or zero if an error occurs.

**Remarks**

The method creates an object of type DivaConference and returns a reference to the object. The object is based on the properties of the instance. The object can be used to manage conferences. The members of the conference are of type DivaCall.

**See also**

CreateCall

## MWIActivate

Activates a message waiting indication.

Set bValue = object.MWIActivate ( Device, Service, NumMessages, Status, Reference, Mode, ReceiveingUser, ControllingUserNumber, ControllingUserProvidedNumber, Time )

**Parameter**

*Device*

Specifies the line device on which the activation should be done. The device is an index starting from 1 to the maximum amount of devices. Refer to DivaSystem References and DivaDevice References for more information.

*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*

The parameter specifies the amount of messages that should be signaled.

*Status*

Specifies the status, for options see DivaMWIMessageStatus.

*Reference*

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

*Mode*

The parameter specifies the invocation mode. For valid options see DivaMWIInvokeMode.

*ReceivingUserNumber*

The parameter specifies 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.

*Time*

The parameter is optional. If the time is given, the format must be YYYY.MM.DD.HH.MM

**Returns**

If the method succeeds, the result code is *DivaResultSuccess*.

**Remarks**

The method activates the message waiting as defined by the given parameter. The method is synchronous and returns when the message waiting activation has been successfully initiated or an error occurred. The internal timeout for a successful operation is set to 2 seconds.

**See also**

MWIDeactivate

## MWIDeactivate

Deactivates a message waiting indication.

Set bValue = object.MWDeactivate ( Device, Service, Mode, ReceiveingUser, ControllingUserNumber )

**Parameter**

*Device*

Specifies the line device on which the deactivation should be done. The device is an index starting from 1 to the maximum amount of devices. Refer to DivaSystem and DivaDevice for more information.

*Service*

The parameter 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.

*Mode*

The parameter specifies the invocation mode. For valid options see DivaMWIInvokeMode.

*ReceivingUserNumber*

The parameter specifies 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.

**Returns**

If the method succeeds, the result code is *DivaResultSuccess*.

**Remarks**

The method deactivates the message waiting as defined by the given parameter. The method is synchronous and returns when the message waiting deactivation has been successfully initiated or an error occurred. The internal timeout for a successful operation is set to 2 seconds.

**See also**

MWIActivate

## MWIReport

Enables or disables the reporting of a message waiting indication on the specified device.

**Parameter**

*Device*

The *Device* parameter specifies the line device. The line device is an index, starting with one. The parameter is of type Long.

*Enable*

The *Enable* parameter specifies if the reporting should be enabled or disabled. The type of the parameter is Boolean.

**Returns**

If the method succeeds, the return code is *DivaSuccess*. Other result codes may be *DivaResultLineDevice* or *DivaResultNotSupported*.

**Remarks**

The method enables or disables the reporting of events for message waiting indications. The application must implement the event handler to receive information about waiting messages.

**See also**

OnMWIIndication, GetMWIIndication, DivaMWIIndication

## GetMWIIndication

The method retrieves an object containing details of the received message waiting indication.

Dim MWIIndication as DivaSDKLib.DivaMWIIndication

DivaMWIIndication = Inst.GetMWIInidication()

**Parameters**

None

**Returns**

The method returns a reference to an object containing the information about the message waiting indication. If no information is available, zero is returned.

**Remarks**

When information about the message waiting indication is available, an object of type DivaSDKLib.DivaMWIIndication is created and returned to the caller. The object contains the information about the waiting messages, e.g., the amount of messages. The caller is responsible for clearing the object if no longer needed.

**See also**

OnMWIIndication, MWIReport, DivaMWIIndication

## DivaInstance Properties

This section contains the following DivaInstance properties:



### LocalNumber

The *LocalNumber* property sets the default local number for call objects.

InstanceObj.LocalNumber = "012345678"

**Type**

String

**Default value**

""

**Availability**

Read and write

**See also**

LocalSubAddress

### LocalSubAddress

The *LocalSubAddress* property sets the default local sub address for call objects.

InstanceObj.LocalSubAddress = "012345678"

**Type**

String

**Default value**

""

**Availability**

Read and write

**See also**

LocalNumber

### VoiceEchoCanceller

The *VoiceEchoCanceller* property sets the default for enabled or disabled echo cancellation for voice calls on call objects created on this instance.

InstanceObj.VoiceEchoCanceller = True

**Type**

Boolean

**Default value**

False

**Availability**

Read and write

**See also**

No references

### FaxLocalId

The *FaxLocalId* property sets the default local ID for fax calls on call objects created on this instance.

InstanceObj.FaxLocalId = "+49 xxxxxxxx"

**Type**

String

**Default value**

""

**Availability**

Read and write

**See also**

FaxHeadLine

### FaxHeadLine

The *FaxHeadLine* property sets the default head line for fax calls on call objects created on this instance.

InstanceObj.FaxHeadLine = "Sent by Diva SDK"

**Type**

String

**Default value**

""

**Availability**

Read and write

**See also**

FaxLocalId

### DivaMWIIndication

The object *DivaMWIIndication* is used to retrieve detailed results for a message waiting indication. The usage of the object is optional. Applications that only need the information if messages are waiting or not do not require this object.

The object DivaMWIIndication can be retrieved via a call to *GetMWIIndication* on the DivaInstance object. The object provides read only properties for the received information.

The following table lists the properties, their type and the description.

| Property Name | Type | Description |
|---|---|---|
| Device | Long | Specifies the device on which the MWI was received. |
| Line | Long | The Line parameter specifies the analog line on which the message waiting indication was received. The parameter is only valid for devices based on Diva Analog Media Boards; for all other Diva Media Boards, the parameter is set to zero. |
| Status | Long | Reserved for future use. |
| NumMessages | Long | Amount of messages waiting; zero if no messages are waiting. |
| Service | Long | The service signaled with the MWI, e.g., voice. |
| Called Party | String | Number for which the message was delivered (Not on analog lines) |
| Controlling user | String | Number of the user that initiated the MWI. |

## DivaInstance Defines

This section contains the folloing DivaInstance defines:

- DivaMWIMessageStatus
- DivaMWIInvokeMode

### DivaMWIMessageStatus

| Option | Value |
|---|---|
| DivaMWIMessageAdded | 0 |
| DivaMWIMessageRemoved | 1 |
| DivaMWIMessageUnknown | 0xffff |

*DivaMWIMessageAdded*

The message should be added

*DivaMWIMessageRemoved*

The message should be removed

*DivaMWIMessageUnknown*

The status of the message is unknown

### DivaMWIInvokeMode

| Option | Value |
|---|---|
| DivaMWIInvokeDeferred | 0 |
| DivaMWIInvokeImmediate | 1 |
| DivaMWIInvokeCombined | 2 |
| DivaMWIInvokeSuppress | 3 |

*DivaMWIInvokeDeferred*

Deferred invocation mode

*DivaMWIInvokeImmediate*

Immediate invocation mode

*DivaMWIInvokeCombined*

Combined invocation mode

*DivaMWIInvokeSuppress*

Suppress the invocation mode

## DivaInstance Events

This section contains the following DivaInstance Event:

### OnMWIIndication

The event *OnMWIIndication* is signaled when a message waiting indication is received.

Inst_OnMWIIndication ( Device as Long, Line as Long, MessageWaiting as Boolean )

**Parameter**

*Device*

The *Device* parameter specifies the line device on which the message waiting indication was received. The device is an index, starting from one.

*Line*

The *Line* parameter specifies the analog line on which the message waiting indication was received. The parameter is only valid for devices based on Dialogic® Diva® Analog Media Boards; for all other Diva Media Boards, the parameter is set to zero. The line is an index, starting from one.

*MessageWaiting*

The *MessageWaiting* parameter specifies if a message is waiting. If true, one or more messages are waiting.

**Remarks**

The reporting of message waiting events must be enabled by the method MWIReport. The event is signaled when a MWI indication is received. The status, e.g., if a message is waiting or not, is reported with the event parameters.

The application may retrieve more information about the indication via the object DivaMWIIndication that can be retrieved via the method *GetMWIIndication* on the DivaInstance object.

**See also**

MWIReport, GetMWIIndication, DivaMWIIndication

# CHAPTER 9

# DivaDevice References

The object DivaDevice is used to provide information about installed communication devices. Default settings to be used on all calls made on this device can be set. The description of the properties states this. DivaDevice objects are created by DivaSystem.

## DivaDevice Methods

This section contains the following DivaDevice Methods:

- ResetCodecList
- AddCodec
- SetServiceState

### ResetCodecList

Each device object has a default codec list that is used for incoming and outgoing calls processed on this device, if not overwritten on a per call base. The method *ResetCodecList* removes all codecs from this list.

retVal = DeviceObj.RestCodecList ()

**Parameter**

None

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns a corresponding error code.

**Remarks**

The method removes all default codecs. The application must either use *AddCodec* on the device object or the call object to add one or more codec before calls can be initiated, accepted, or answered.

**See also**

AddCodec

### AddCodec

Each device object has a default codec list that is used for incoming and outgoing calls, if not overwritten on a per call base. Via *AddCodec* the application adds a codec to the default list.

Result = Device.AddCodec ( Codec, Framesize, Direction )

**Parameter**

*Codec*

The *Codec* parameter specifies the codec to be added. For valid codecs, refer to DivaCodecs.

*FrameSize*

The *FrameSize* parameter is an integer value that specifies the frame size for the codec.

*Direction*

The *Direction* parameter specifies if the codec is used for transmit, receive, or for both. The type of the parameter is DivaDirections.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns a corresponding error code.

**Remarks**

This is a synchronous method that returns immediately. The codec is added to the internal codec list of the device object.

**See also**

ResetCodecList

## SetServiceState

Sets the service state of a line device.

retVal = DeviceObj.SetServiceState ( Enable )

**Parameter**

*Enable*

*Enable* is a boolean value specifying if the device should be placed in service or out of service.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The function changes the service state of a line device. The function is only supported on line devices based on Dialogic® 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.

## DivaDevice Properties

This section contains the following DivaDevice properties:

- Channels
- SerialNumber
- FaxSupported
- ModemSupported
- CodecALaw
- VoIPSupported
- ExtendedVoiceSupported
- HoldRetrieveSupported
- TransferSupported
- ForwardSupported
- CallDeflectionSupported
- Line
- IPBased
- PSTNBased
- AnalogBased
- Layer1Status
- Layer2Status
- AnalogLineStatus
- RedAlarm
- BlueAlarm
- YellowAlarm

### Channels

The *Channels* property returns the number of data channels of the devices.

nChannels = DeviceObj.Channels

**Type**

Integer

**Availability**

Read only

**See also**

No references

### SerialNumber

The *SerialNumber* property returns the serial number of the device.

nSerialNumber = DeviceObj.SerialNumber

**Type**

Integer

**Availability**

Read only

**See also**

No references

### FaxSupported

The *FaxSupported* property returns true if the device supports fax transmission and reception.

bFax = DeviceObj.FaxSupported

**Type**

Boolean

**Availability**

Read only

**See also**

No references

### ModemSupported

The *ModemSupported* property returns true if the device supports modem protocols.

bModem = DeviceObj.ModemSupported

**Type**

Boolean

**Availability**

Read only

**See also**

No references

### CodecALaw

The *CodecALaw* property returns true if the line is configured for a-law voice coding. If false is returned, the line uses µ-law coding.

bCodexAlaw= DeviceObj.CodecALaw

**Type**

Boolean

**Availability**

Read only

**See also**

No references

### VoIPSupported

The *VoIPSupported* property returns true if the device supports RTP streaming. This property is only valid for devices that are PSTN-based. The information if a device is using IP-based communication the property *IPBased* must be used.

bFax = DeviceObj.VoIPSupported

**Type**

Boolean

**Availability**

Read only

**See also**

IPBased

### ExtendedVoiceSupported

The *ExtendedVoiceSupported* property returns true if the device supports tone detection, automatic gain control, extended conference support and other extended voice functions. In general, this is supported by the Dialogic® Diva® Media Boards having DSP support.

bFax = DeviceObj.ExtendedVoiceSupported

**Type**

Boolean

**Availability**

Read only

**See also**

No references

### HoldRetrieveSupported

The *HoldRetrieveSupported* property returns true if the device supports the supplementary services hold and retrieve.

bFax = DeviceObj.HoldRetrieveSupported

**Type**

Boolean

**Availability**

Read only

**See also**

No references

### TransferSupported

The *TransferSupported* property returns true if the device supports the supplementary services call transfer.

bFax = DeviceObj.TransferSupported

**Type**

Boolean

**Availability**

Read only

**See also**

No references

### ForwardSupported

The *ForwardSupported* property returns true if the device supports the supplementary services call forwarding.

bFax = DeviceObj.ForwardSupported

**Type**

Boolean

**Availability**

Read only

**See also**

No references

### CallDeflectionSupported

The *CallDeflectionSupported* property returns true if the device supports the supplementary services call deflection.

bFax = DeviceObj.CallForwardingSupported

**Type**

Boolean

**Availability**

Read only

**See also**

No references

### Line

The *Line* property returns for multi-line boards to which line the object corresponds. The returned value is an index starting with one. For boards that have only one line, the value one is returned.

Line = DeviceObj.Line

**Type**

long

**Availability**

Read only

**See also**

No references

### IPBased

The *IPBased* property returns true if the device is based on IP signaling and media streaming, e.g., H.323 or SIP.

bMedia = DeviceObj.IPBased

**Type**

Boolean

**Availability**

Read only

**See also**

PSTNBased, AnalogBased

### PSTNBased

The *PSTNBased* property returns true if the device is based on PSTN. This includes ISDN PRI, inband signaling protocols like RBS and E1 R2 as well as analog lines.

bMedia = DeviceObj.PSTNBased

**Type**

Boolean

**Availability**

Read only

**See also**

IPBased, AnalogBased

### AnalogBased

The *AnalogBased* property returns true if the device is based on a POTS line using an Dialogic® Diva® Analog Media Board.

bAnalog = DeviceObj.AnalogBased

**Type**

Boolean

**Availability**

Read only

**See also**

IPBased, PSTNBased

### Layer1Status

The *Layer1Status* property returns the status of the layer 1 if supported by the device type. This is only valid for devices that are PSTN-based and not analog-based. For valid layer 1 status values, refer to DivaDeviceLayer1Status.

bL1State = DeviceObj.Layer1Status

**Type**

DivaDeviceLayer1Status

**Availability**

Read only

**See also**

PSTNBased, AnalogBased, Layer2Status, RedAlarm, BlueAlarm, YellowAlarm, DivaDeviceLayer1Status, AnalogLineStatus

### Layer2Status

The *Layer2Status* property returns the status of the layer 2 if supported by the device type. This is only valid for devices that are PSTN-based and not analog-based. For valid layer 2 status values, refer to DivaDeviceLayer2Status.

bL1State = DeviceObj.Layer2Status

**Type**

DivaDeviceLayer2Status

**Availability**

Read only

**See also**

PSTNBased, AnalogBased, Layer1Status, RedAlarm, BlueAlarm, YellowAlarm, DivaDeviceLayer2Status, AnalogLineStatus

### AnalogLineStatus

The *AnalogLineStatus* property returns the status of the specified analog line if supported by the device type. This is only valid for devices that are analog-based. For valid analog line status values, refer to DivaDeviceAnalogStatus.

bL1State = DeviceObj.AnalogLineStatus ( 1 )

**Type**

DivaDeviceAnalogStatus

**Availability**

Read only

**See also**

PSTNBased, AnalogBased, Layer2Status, RedAlarm, BlueAlarm, YellowAlarm, DivaDeviceLayer1Status, DivaDeviceAnalogStatus

### RedAlarm

The *RedAlarm* property returns the state of the red alarm. This is only valid for T1/E1-based hardware. For valid alarm status values please refer to DivaAlarmStatus.

bL1State = DeviceObj.RedAlarm

**Type**

DivaAlarmStatus

**Availability**

Read only

**See also**

PSTNBased, AnalogBased, Layer1Status, Layer2Status, BlueAlarm, YellowAlarm, DivaAlarmStatus

### BlueAlarm

The *BlueAlarm* property returns the state of the blue alarm. This is only valid for T1/E1-based hardware. For valid alarm status values, refer to DivaAlarmStatus.

bL1State = DeviceObj.BlueAlarm

**Type**

DivaAlarmStatus

**Availability**

Read only

**See also**

PSTNBased, AnalogBased, Layer1Status, Layer2Status, RedAlarm, YellowAlarm, DivaAlarmStatus

### YellowAlarm

The *YellowAlarm* property returns the state of the yellow alarm. This is only valid for T1/E1-based hardware. For valid alarm status values, refer to DivaAlarmStatus.

bL1State = DeviceObj.YellowAlarm

**Type**

DivaAlarmStatus

**Availability**

Read only

**See also**

PSTNBased, AnalogBased, Layer1Status, Layer2Status, BlueAlarm, RedAlarm, DivaAlarmStatus

## DivaDevice Defines

This section contains the following DivaDevice Defines:

- DivaDeviceLayer1Status
- DivaDeviceLayer2Status
- DivaDeviceAnalogStatus
- DivaAlarmStatus

### DivaDeviceLayer1Status

| Option | Value |
|---|---|
| L1NotSupported | 0x80000000 |
| L1Down | 0 |
| L1Up | 1 |
| L1SyncLost | 2 |
| L1Synchronized | 3 |

*L1NotSupported*

The device does not support a layer 1 status reporting.

*L1Down*

The layer 1 is down, typically a cable disconnect.

*L1Up*

The layer 1 is up and ready to exchange data.

*L1SyncLost*

The layer 1 has lost synchronization.

*L1Synchronized*

The layer 1 is synchronized but not yet fully up.

### DivaDeviceLayer2Status

| Option | Value |
|---|---|
| L2NotSupported | 0x80000000 |
| L2Down | 0 |
| L2Up | 1 |
| L2Closing | 2 |
| L2Activating | 3 |
| L2Initializing | 4 |

*L2NotSupported*

The device does not support a layer 2 status reporting.

*L2Down*

The layer 2 failed to negotiate.

*L2Up*

The layer 2 is up and signaling messages can be exchanged.

*L2Closing*

The layer 2 is shutting down.

*L2Activating*

The layer 2 is trying to establish a connection.

*L2Initializing*

The layer 2 is initializing to establish a link.

### DivaDeviceAnalogStatus

| Option | Value |
|---|---|
| AnalogNotSupported | 0x80000000 |
| AnalogLineDown | 0 |
| AnalogLineOffHook | 1 |
| AnalogLineIdle | 2 |
| AnalogLineRing | 3 |
| AnalogLinePolarityReverse | 4 |

*AnalogNotSupported*

The device is not an analog-based device.

*AnalogLineDown*

The analog line is down, e.g., due to a cable disconnect.

*AnalogLineOffHook*

The analog line is off hook.

*AnalogLineIdle*

The analog line is idle. An active connection to the switch is available.

*AnalogLineRing*

The switch is signaling an incoming call by a ring.

*AnalogLinePolarityReverse*

The line has reverse polarity.

### DivaAlarmStatus

| Option | Value |
|---|---|
| AlarmNotSupported | 0x80000000 |
| AlarmInactive | 0 |
| AlarmActive | 1 |

*AlarmNotSupported*

The device does not support alarm states. Only PRI-based devices support.

*AlarmInactive*

The alarm is inactive, which indicates a well running system.

*AlarmActive*

The alarm condition is active.

# CHAPTER 10

## DivaConference References

The Diva Component API handles conferences on a DivaConference object. The object allows for managing a conference with unlimited members. The conference - including mixing and automatic gain control - is handled by the underlying Diva communication platform. Members of the conference are of the type DivaCall.

The conference object is created based on a DivaInstance. The method CreateConference returns an object of type DivaConference. On this object, calls can be added to the conference and therefore become conference members. A call may be removed at any time. All calls to add or remove a call are synchronous calls.

Applications may want to stream a conference object or may want to record the conference. The conference object provides methods to stream and record audio that are similar to the streaming and recording on a DivaCall object. The streaming can be synchronous or asynchronous. The conference object provides an event interface for notifications when the streaming or recording ends.

The streaming is done on a so called "virtual master call". The conference object provides that the virtual master is automatically switched if needed, e.g., the master is removed from the conference or is disconnected. However, applications may want to control this in order to stream an announcement to a single member. The conference object provides methods to detect if a member is currently the virtual master and also to select a different member as master.

## DivaConference Methods

All methods of DivaConference are synchronous methods. When the methods return, the operation has been done and the result code contains the success or failure reason.

This section contains the following DivaConference Methods:



### Add

Adds a new member to the conference.

retVal = ConfObject.Add ( CallObject, Rights )

**Parameter**

*CallObject*

The parameter *CallObject* is of type DivaCall and specifies the member to be added.

*Rights*

The parameter *Rights* specifies the rights of the new member.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method adds the DivaCall object identified as parameter *CallObject* to the conference. The new member gets the rights specified by the parameter *Rights*. For valid rights, see DivaConfMemberRights. By default, the rights speak and listen are granted.

**See also**

Remove, SetRights, Clear, MemberCount

## Remove

Removes a member from the conference.

retVal = ConfObject.Remove ( CallObject )

**Parameter**

*CallObject*

The parameter *CallObject* is of type DivaCall and specifies the member to be removed.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method removes the DivaCall object identified as parameter *CallObject* from the conference.

**See also**

Add, SetRights, Clear, MemberCount

## SetRights

Modifies the rights of the conference member.

retVal = ConfObject.SetRights ( CallObject, Rights )

**Parameter**

*CallObject*

The parameter *CallObject* is of type DivaCall and specifies the member.

*Rights*

The parameter *Rights* specifies the rights to be granted to the member.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method modifies the right of the conference member identified by the parameter CallObject. The member must have been added to the conference previously by the *Add* method. The member gets the rights specified by the parameter *Rights*. For valid rights see DivaConfMemberRights. By default, the rights speak and listen are granted.

**See also**

Add, Remove, Clear, MemberCount

### Clear

Clears the conference and optionally disconnects the members.

retVal = object1.Clear ( bDisconnect )

**Parameter**

*bDisconnect*

BOOL value indicating if the calls should be disconnected.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

The method removes the members from the conference and releases the conference. The conference is no longer available after a call to this method and must be deleted. If the parameter *bDisconnect* is set, the calls that were members of the conference are disconnected.

**See also**

Add, Remove, SetRights, MemberCount

### IsMaster

Retrieves the information if the specified call object is the master call of the conference.

Set bValue = object.IsMaster ( CallObject )

**Parameter**

*CallObject*

The parameter *CallObject* is of type DivaCall and specifies the call for which the information about virtual master is returned.

**Returns**

True if the call object is the virtual master. If the call object is not a member or the call is not the virtual master, the method returns false.

**Remarks**

The method verifies if the specified call object is part of the conference and if it is the current virtual master call. This is a synchronous method.

**See also**

IsMember, SetMaster

### IsMember

Retrieves the information if the specified call object is the member of the conference.

Set bValue = object.IsMember ( CallObject )

**Parameter**

*CallObject*

The parameter *CallObject* is of type DivaCall and specifies the call for which the information about membership is returned.

**Returns**

True if the call object is part of the conference. If the call object is not a member, the method returns false.

**Remarks**

The method verifies if the specified call object is part of the conference. This is a synchronous method.

**See also**

IsMaster, SetMaster

## SetMaster

Changes the current virtual master call.

Set Result = ConfObject.SetMaster ( CallObject )

**Parameter**

*CallObject*

The parameter *CallObject* is of type DivaCall and specifies the call that should be the new virtual master call.

**Returns**

If the function succeeds, the result code is *DivaResultSuccess* (0).

**Remarks**

The method verifies if the specified call object is part of the conference and if it is not the current master call. Before switching the master call, the streaming on this call is terminated. Any pending streaming on the conference object is not interrupted by switching the master call.

Applications may switch the virtual master call because an announcement to a single conference member should be done. If this member is currently the virtual master call, the streaming would be heard by the members. This is a synchronous method.

**See also**

IsMember, IsMaster

## SendVoiceFile

Streams audio information from a file.

retVal = object.SendVoiceFile ( Filename, Format )

**Parameter**

*Filename*

The parameter *Filename* is a string value that specifies the file to be streamed. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaAudioAutodetect*.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SendVoiceFile* opens the given file and streams the data. If the parameter *Format* is set to *DivaAudioAutodetect*, the format is automatically detected from the file header.

The file name may include drive / network share and path information. If only the file name is given, only the current directory is searched.

*SendVoiceFile* is an asynchronous method. If asynchronous mode is disabled, the method blocks the caller until the streaming is completed. A return code of *DivaSuccess* indicates that the streaming has finished successfully. If events are enabled, the termination of the streaming is signaled by a call to the event method *OnVoiceStreamed*.

**See also**

SendVoiceFiles, StopSending, RecordVoiceFile, StopRecording

## SendVoiceFiles

Streams audio information from one or more files.

retVal = object.SendVoiceFiles ( Filename, Format, Continuous, MaxSeconds)

**Parameter**

*Filename*

The parameter *Filename* is a string value that specifies the files to be streamed. The single files to be streamed are separated by commas. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaAudioAutodetect*.

*Continuous* (optional)

The parameter *Continuous* is a Boolean value. If set, the audio streaming is repeated until the maximum time is reached or the call is disconnected. The parameter is optional and the default is false.

*MaxSeconds* (optional)

The parameter *MaxSeconds* is a long value that specifies the maximum time the audio should be streamed. A value of zero specifies no limitation. The parameter is optional and the default is no timeout.

**Returns**

If the method succeeds, the return value is *DivaResultSuccess* (0). In case of an error, the method returns an error code.

**Remarks**

*SendVoiceFiles* streams the audio information from the files. No gap will occur between the different files. The parameter *Filename* contains the files to be streamed. The files are separated by commas.

If the parameter *Format* is set to *DivaAudioAutodetect*, the format is automatically detected from the file header. If the format is not set to *DivaAudioAutodetect*, the files must have the same audio format.

*SendVoiceFiles* is an asynchronous method. If asynchronous mode is disabled, the method blocks the caller until the streaming is completed. A return code of *DivaSuccess* indicates that the streaming has finished successfully. If events are enabled, the termination of the streaming is signaled by a call to the event method *OnVoiceStreamed*.

**See also**

SendVoiceFile, StopSending, RecordVoiceFile, StopRecording

### StopSending

Terminates the streaming of the voice data.

object.StopSending ( )

**Parameter**

None

**Returns**

None

**Remarks**

This is a synchronous method. If events are enabled, a confirmation via a call to the event method *OnVoiceStreamed* is signaled.

The method terminates pending data that is streaming. If audio streaming is active, pending audio buffers are discarded, and the streaming stops right away.

**See also**

SendVoiceFile, SendVoiceFiles, RecordVoiceFile, StopRecording

### RecordVoiceFile

Records audio to a file. Optionally, it stops recording after a specified time.

retVal = object.RecordVoiceFile ( Filename, Format, MaxSeconds )

**Parameter**

*Filename*

The parameter *Filename* is a string value that specifies the files to be streamed. The individual files to be streamed are separated by commas. See Remarks.

*Format* (optional)

The parameter *Format* is a long value that specifies the file format. The parameter is optional. The default value is *DivaAudioDefault*.

*MaxSeconds* (optional)

The parameter *MaxSeconds* is a long value and specifies the maximum recording length in seconds. A value of zero indicates no timeout. The parameter is optional and the default is no timeout.

**Returns**

In asynchronous mode, the return value is *DivaResultSuccess* (0) if successful. In case of an error, the method returns an error code. If the function is called with disabled asynchronous mode, the return value depends on the operation mode and the parameters. See Remarks.

**Remarks**

The method records the audio information to the specified file. The file name may include drive / network share and path information.

If only the file name is given, the file is placed in the current directory. If the parameter *Format* is set to *DivaAudioDefault*, the format is set to PCM 8 KHz Mono.

*RecordVoiceFile* is an asynchronous method. If asynchronous mode is disabled, the method blocks until one of the following occurs:

- The last member is removed from the conference. The return value is *DivaResultDisconnected*.
- *MaxSeconds* is set to non-zero and the timeout has been reached. The return value is *DivaResultTimeReached*.

- If events are enabled the termination of the streaming is signaled by a call to the event method *OnVoiceStreamed*.

**See also**

SendVoiceFile, SendVoiceFiles, StopSending, StopRecording

## StopRecording

Terminates the recording of voice data.

object.StopRecording ( )

**Parameter**

None

**Returns**

None

**Remarks**

This is a synchronous method. If events are enabled a confirmation via a call to the event method *OnRecordEnded* is signaled. The recorded file is closed and can be accessed when the function returns.

**See also**

SendVoiceFile, SendVoiceFiles, StopSending, RecordVoiceFile

## DivaConference Properties

This section contains the following DivaConference properties:

- MemberCount
- AsyncMode
- SignalEvents

### MemberCount

Gets the amount of calls in the conference.

Dim MemberCount as long

MemberCount = ConfObject.MemberCount

**Type**

long

**Availability**

Read only

**Remarks**

The property returns the amount of members in the conference.

**See also**

No references.

### AsyncMode

Sets the operation mode to synchronous or asynchronous.

CallObj.AsyncMode = True

**Type**

Boolean

**Default value**

False

**Availability**

Read and write

**Remarks**

The *AsyncMode* property is available for reading and writing. On write it enables or disables the asynchronous mode. On read it returns the current state of the operation mode. A value of false for the *AsyncMode* sets the operation mode to synchronous, which is the default.

**Note:** The signaling of the events is not changed by this property but by *SignalEvents*.

**See also**

SignalEvents

## SignalEvents

Enables or disables the signaling of events.

CallObj.SignalEvents = True

**Type**

Boolean

**Default value**

False

**Availability**

Read and write

**Remarks**

The *SignalEvents* property is available for reading and writing. On write it enables or disables the signaling of events. On read it returns the current state of the event signaling.

**Note:** This property does not change between synchronous and asynchronous operation mode.

**See also**

AsyncMode

## DivaConference Defines

This section contains the following DivaConference Defines:

### DivaConfMemberRights

Conference members have by default the rights to speak and to listen. The rights can be modified at any time. The following rights are available:

```
typedef enum
{
        DivaConfMemberRightSpeak = 1,
        DivaConfMemberRightListen = 2,
        DivaConfMemberRightSpeakListen = 3
} DivaConfMemberRights;
```

*DivaConfMemberRightSpeak*

The member is allowed to speak to the members that have listen rights.

*DivaConfMemberRightListen*

The member is allowed to listen to the conference.

*DivaConfMemberRightSpeakListen*

The member is allowed to listen to the conference and also to speak to the other members that have listen rights.

## DivaConference Events

This section contains the following DivaConference Events:

- OnVoiceStreamed
- OnRecordEnded
- OnMembersChanged

### OnVoiceStreamed

The event *OnVoiceStreamed* is triggered when the audio streaming on the conference object has finished.

ConferenceObject_OnVoiceStreamed ( Reason As LONG )

**Parameter**

*Reason*

The parameter contains the reason for signaling the event. See Remarks.

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when the streaming, initiated by one of the streaming functions, has finished.

The reason for signaling the event may be DivaSendVoiceEndReason_Cancelled, if the user terminates the streaming. If a continous streaming wraps around, the reason is DivaSendVoiceEndReason_Restarted.

**See also**

SignalEvents, SendVoiceFile, SendVoiceFiles

### OnRecordEnded

The event *OnRecordEnded* is triggered when the audio recording has finished.

Conferencet_OnRecordEnded ( Reason as LONG )

**Parameter**

*Reason*

The parameter contains the reason for signaling the event. See Remarks.

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when the recording, initiated by *RecordVoiceFile*, has finished.

The parameter *Reason* specifies the reason of the end, either the maximum duration has been reached or the user terminated the recording.

**See also**

SignalEvents, RecordVoiceFile

### OnMembersChanged

The event *OnMembersChanged* is triggered when the member information of the conference has changed.

Conference_OnMembersChanged ( )

**Parameter**

None

**Remarks**

The signaling of events must be enabled using the property *SignalEvents*. The event is signaled when the member information changes, e.g., because a call is disconnected and therefore implicitly removed from the conference.

**See also**

No references.

# CHAPTER 11

## DivaToneResult References

The object DivaToneResult is used to retrieve the detector result for a single or dual tone. The usage of the object is optional. Applications that use the DivaCall events *OnSingleToneDetected* or *OnDualToneDetected* and only need the detected frequencies, do not require this object.

Applications that work in synchronous mode or require additional information, e.g., energy of the detected tone must obtain a DivaToneResult object via GetToneDetectorResult. The method returns an object if available. Note that the application is responsible for cleanup of the object.

## DivaToneResult Properties

The following table lists the properties, their types, availability and description.

| Property Name | Type | Available for | Description |
|---|---|---|---|
| DualTone | Boolean | Single / Dual | If true, dual tone, else single tone. |
| SignalNoiseRatio | Long | Single / Dual | Signal to noise ratio in dB. |
| Frequency | Long | Single | Frequency in Hz. |
| Energy | Long | Single | Energy in dB. |
| AmplitudeVariation | Long | Single | Variation of the amplitude during detection in dB. |
| FrequencyVariation | Long | Single | Variation of the frequency during the detection in dB. |
| FrequencyToneLow | Long | Dual | Frequency of the lower tone in Hz. |
| FrequencyToneHigh | Long | Dual | Frequency of the higher tone in Hz. |
| EnergyToneLow | Long | Dual | Energy of the lower tone in Hz. |
| EnergyToneHigh | Long | Dual | Energy of the lower tone in Hz. |

## Using DivaToneResult

The following sample script shows the usage of the object DivaToneResult in asynchronous and synchronous environments.

### Asynchronous processing

The following script extract contains the handler for the connect event (Call_OnConnected) and for the detection event of a single tone (Call_OnSingleToneDetected). The Call_OnConnect handler enables the single tone detector for tones with a minimum duration of 100 milliseconds. The Call_OnSingleToneDetector prints the frequency given with the event and retrieves the additional information like energy and prints them as well.

```
sub Call_OnConnected
   wscript.echo "Connected..."
   Result = CallObj.EnableSingleToneDetector(100)
   wscript.echo "Enable detector returned: " & Result
end sub



sub Call_OnSingleToneDetected ( Frequency )
   If (Frequency <> 0 ) Then
      wscript.echo "Single tone detected, Frequency: " & Frequency
      Set ToneObj = CallObj.GetToneDetectorResult ()
      If ( ToneObj Is Nothing ) Then
            wscript.echo "No Tone Information"
      Else
            wscript.echo "Signal to Noice Ratio " &
                         ToneObj.SignalNoiseRatio
            wscript.echo "Frequency " & ToneObj.Frequency & "
                         Energy " & ToneObj.Energy
            wscript.echo "Variation (A/F) " &
                         ToneObj.AmplitudeVariation & "/" & _
                         ToneObj.FrequencyVariation
      End If
   Else
      wscript.echo "Single tone ended”
   End If

end sub
```

### Synchronous processing

The following sample script for synchronous processing initiates an outbound call. Once the call is connected it enables the single tone detector for tones in the range of 1000 to 1590 Hz. Then it streams an audio file in synchronous mode. Once a tone is detected the method SendVoiceFile terminates with the result code DivaResultToneDetected (21). The script retrieves the details for the tone via GetToneDetectorResult and prints the detection result retrieved via the properties of the returned DivaToneResult object.

```
Dim ToneObj

Set CallObj = CreateObject ( "DivaSDK.DivaCall" )

wscript.echo "Connecting to <11223344>"
retVal = CallObj.Connect ( "11223344" )
If ( retVal = DivaResultSuccess ) Then
     CallObj.EnableSingleToneDetector(100)
     CallObj.SingleToneDetectorMinFrequency = 1000
     CallObj.SingleToneDetectorMaxFrequency = 1590
     wscript.echo "Connected, stream message"
     retVal = CallObj.SendVoiceFile ("testlong.wav")
     If ( retVal = 0 ) Then
         wscript.echo "Voice file successfully streamed."
     Else
         If (retVal = 21 ) Then
              wscript.echo "Streaming stopped, Tone received."

              Set ToneObj = CallObj.GetToneDetectorResult ()
              If ( ToneObj Is Nothing ) Then
                  wscript.echo "No Tone Information"
              Else
                  wscript.echo "Frequency " & ToneObj.Frequency &
                                " Energy " & ToneObj.Energy
                  wscript.echo "Signal to Noice Ratio " &
                                ToneObj.SignalNoiseRatio
                  wscript.echo "Variation (A/F) " &
                                ToneObj.AmplitudeVariation & "/" &
                                _ ToneObj.FrequencyVariation
              End If

             wscript.Sleep(5000)
         Else
              wscript.echo "Voice sent failed with result: " &
                           retVal
         End If
     End If

    wscript.echo "Disconnecting"
    CallObj.Disconnect ( )
Else
    wscript.echo "Connect failed with result: " & retVal
End If
```

# CHAPTER 11

## DivaCustomTone References

The object *DivaCustomTone* is used to specify the characteristics of a custom tone. The custom tone definition can be used for the answering machine detector to detect the end of the message typically indicated by a beep tone.

To specify a custom tone, the application creates a DivaCustomTone object and passes the object to the Dialogic® Diva® Component API. The Diva Component API reads the tone specification from the object. The object is no longer required by the Diva Component API when the method or property returns. The application is responsible for the cleanup of the object.

## DivaCustomTone Properties

The following table lists the properties, their types, availability for single or dual tones and description.

| Property Name | Type | Available for | Description |
|---|---|---|---|
| Frequency | Long | Single | Frequency of a single tone, given in Hz. |
| FrequencyVariation | Long | Single | The frequency variation of the single tone, given in Hz. |
| FrequencyLow | Long | Dual | The low frequency of a dual tone, given in Hz. |
| FrequencyHigh | Long | Dual | The high frequency of a dual tone, given in Hz. |
| FrequencyLowVariation | Long | Dual | The frequency variation of the high frequency of the dual tone, given in Hz. |
| FrequencyHighVariation | Long | Dual | The frequency variation of the high frequency of the dual tone, given in Hz. |
| Duration | Long | All | The duration the tone must be active, given in milliseconds. The maximum value is 8000 including the variation. |
| DurationVariation | Long | All | The variation of the duration, given in milliseconds. |
| Pause | Long | All | The silence following the tone, given in milliseconds. The maximum value is 8000 including the variation. |
| PauseVariation | Long | All | The variation of the silence, given in milliseconds. |

The properties of DivaCustomTone allow an application to specify a single or dual tone with a duration, and optionally, with silence after the tone. The frequency and time parameters are specified as value and a variation of the value. This allows for specifying a range for the detector. Below are samples for a single and dual tone. Note that the property *AMDRecordingTone* allows only tone definitions with a pause set to zero.

For detecting a tone with a frequency of 800 to 1200Hz with a duration between 200 and 400 milliseconds, the following definition must be set:

```
Dim Tone as DivaSDKLib.DivaCutomTone
Tone.Frequency=1000
Tone.FrequencyVariation=200
Tone.Duration=300
Tone.DurationVariation=100
```

For detecting a dual tone with a low frequency of 1200 Hz, a high frequency of 1600 Hz, both with a variation of 10 Hz, and a duration of 100 to 400 milliseconds, the following definition must be set:

```
Dim Tone as DivaSDKLib.DivaCutomTone
Tone.FrequencyLow=1200
Tone.FrequencyLowVariation=10
Tone.FrequencyHigh=1600
Tone.FrequencyHighVariation=10
Tone.Duration=250
Tone.DurationVariation=150
```

## Using DivaToneResult

The following sample script shows the usage of the object DivaToneResult in asynchronous and synchronous environments.

### Asynchronous processing

The following script extract contains the handler for the connect event (Call_OnConnected) and for the detection event of a single tone (Call_OnSingleToneDetected). The Call_OnConnect handler enables the single tone detector for tones with a minimum duration of 100 milliseconds. The Call_OnSingleToneDetector prints the frequency given with the event and retrieves the additional information like energy and prints them as well.

```
sub Call_OnConnected
   wscript.echo "Connected..."
   Result = CallObj.EnableSingleToneDetector(100)
   wscript.echo "Enable detector returned: " & Result
end sub



sub Call_OnSingleToneDetected ( Frequency )
   If (Frequency <> 0 ) Then
      wscript.echo "Single tone detected, Frequency: " & Frequency
      Set ToneObj = CallObj.GetToneDetectorResult ()
      If ( ToneObj Is Nothing ) Then
            wscript.echo "No Tone Information"
      Else
            wscript.echo "Signal to Noice Ratio " &
                          ToneObj.SignalNoiseRatio
            wscript.echo "Frequency " & ToneObj.Frequency & "
                          Energy " & ToneObj.Energy
            wscript.echo "Variation (A/F) " &
                          ToneObj.AmplitudeVariation & "/" & _
                          ToneObj.FrequencyVariation
      End If
   Else
      wscript.echo "Single tone ended”
   End If

end sub
```

### Synchronous processing

The following sample script for synchronous processing initiates an outbound call. Once the call is connected it enables the single tone detector for tones in the range of 1000 to 1590 Hz. Then it streams an audio file in synchronous mode. Once a tone is detected the method SendVoiceFile terminates with the result code DivaResultToneDetected (21). The script retrieves the details for the tone via GetToneDetectorResult and prints the detection result retrieved via the properties of the returned DivaToneResult object.

```
Dim ToneObj

Set CallObj = CreateObject ( "DivaSDK.DivaCall" )

wscript.echo "Connecting to <11223344>"
retVal = CallObj.Connect ( "11223344" )
If ( retVal = DivaResultSuccess ) Then
     CallObj.EnableSingleToneDetector(100)
     CallObj.SingleToneDetectorMinFrequency = 1000
     CallObj.SingleToneDetectorMaxFrequency = 1590
     wscript.echo "Connected, stream message"
     retVal = CallObj.SendVoiceFile ("testlong.wav")
     If ( retVal = 0 ) Then
         wscript.echo "Voice file successfully streamed."
     Else
         If (retVal = 21 ) Then
              wscript.echo "Streaming stopped, Tone received."

              Set ToneObj = CallObj.GetToneDetectorResult ()
              If ( ToneObj Is Nothing ) Then
                  wscript.echo "No Tone Information"
              Else
                  wscript.echo "Frequency " & ToneObj.Frequency &
                                " Energy " & ToneObj.Energy
                  wscript.echo "Signal to Noice Ratio " &
                                ToneObj.SignalNoiseRatio
                  wscript.echo "Variation (A/F) " &
                                ToneObj.AmplitudeVariation & "/" &
                                _ ToneObj.FrequencyVariation
              End If

             wscript.Sleep(5000)
         Else
              wscript.echo "Voice sent failed with result: " &
                           retVal
         End If
     End If

    wscript.echo "Disconnecting"
    CallObj.Disconnect ( )
Else
    wscript.echo "Connect failed with result: " & retVal
End If
```