

**Digi
Voice**

Digivoice Tecnologia em Eletrônica

Digivoice VoicerLib 4

Sistema de Desenvolvimento para placas Digivoice.

Guia do Programador

MAN0048 - Versão 4.0.7.4

Impressão: 12/01/2007, 11:01

Copyright © 2006 Digivoice Eletrônica

Conteúdo

Parte I Introdução	11
Parte II Recursos para o Desenvolvedor	13
Parte III Primeiros Passos	15
1 Windows	15
Instalação	15
Testando a placa instalada	17
2 Linux	19
Instalação	19
Preparando o ambiente	20
Módulos do kernel	21
Compilando a VoicerLib	23
Exemplo em linguagem C	24
Parte IV Guia de Programação	26
1 Conceitos Básicos - API	27
2 Conceitos Básicos - ActiveX	31
3 Guia de Migração de Versões Anteriores	32
4 Inicializando os Serviços	36
5 Finalizando os Serviços	37
6 Detectando Ring	37
7 Atendendo e Desligando	38
8 Supervisão de Linha	39
9 Detecção de Silêncio	49
10 Detecção de Tons	50

11	Detecção de Dígitos	52
12	Identificação de Chamadas	56
13	Portas Virtuais	57
14	Gravando uma Conversa	60
15	Reproduzindo Mensagens	65
16	Conferência entre portas	67
17	Streaming de Áudio	69
18	Gravação em Paralelo	72
	Placas FXO	72
	Placas E1	73
19	Programação da Placa VB6060PCI	78
	Configurações de Sincronismo	78
	Alarmes	80
	Protocolo R2D MFC	82
	Inicialização.....	83
	Efetuando Chamadas.....	83
	Recebendo Chamadas.....	86
	Funções de Controle da Thread E1.....	87
	Protocolo CAS Customizado	88
	Introdução.....	89
20	Barramento H100	89
	Introdução	89
	Conectando Portas	91
	Desconectando Portas	93
	Outras funções H100	93
	Conexão em placas E1 sem H100	95
21	Funções Especiais	96
	Introdução	96
	Funções de Idle	97
	Funções de Prompt	99
	Funções de Menu	100
	Funções de Discagem e Transferência	101
	Reproduzindo Data	105
	Reproduzindo Números Cardinais	106

Reproduzindo Números Dígito a Dígito	107
Reproduzindo Valores por Extenso	108
Reproduzindo Lista de Mensagens	109
22 Distribuindo uma Aplicação	112

Parte V Guia de Referência 115

1 Mensagens de Erro	115
2 Funções/Métodos	119
AbortCall	120
CancelGetDigits (dg_CancelGetDigits)	120
ClearDigits (dg_ClearDigits)	121
ChatAddPort (dg_ChatAddPort)	122
ChatDisablePort (dg_ChatDisablePort)	123
ChatEnablePort (dg_ChatEnablePort)	124
ChatRemovePort (dg_ChatRemovePort)	125
ConfigCallProgress (dg_ConfigCallProgress)	126
ConfigE1Thread (dg_ConfigE1Thread)	130
ConfigCustomCAS (dg_ConfigCustomCAS)	132
ConnectBus (dg_ConnectBus)	134
CreateCallProgress (dg_CreateCallProgress)	136
CreateChatRoom (dg_CreateChatRoom)	138
CreateCustomCAS (dg_CreateCustomCAS)	139
CreateE1Thread (dg_CreateE1Thread)	140
CreateLoggerControl	141
DefinePortResource (dg_DefinePortResource)	143
DestroyCallProgress (dg_DestroyCallProgress)	144
DestroyChatRoom (dg_DestroyChatRoom)	145
DestroyCustomCAS (dg_DestroyCustomCAS)	146
DestroyLoggerControl	147
DestroyE1Thread (dg_DestroyE1Thread)	148
DisconnectBus (d_DisconnectBus)	149
Dial (dg_Dial)	152
DisableAGC (dg_DisableAGC)	153
DisableAnswerDetection (dg_DisableAnswerDetection) ..	154
DisableAutoFramers (dg_DisableAutoFramers)	155
DisableCallProgress (dg_DisableCallProgress)	156

DisableDTMFFilter (dg_DisableDTMFFilter)	157
DisableDebug (dg_DisableDebug)	158
DisableInputBuffer (dg_DisableInputBuffer)	159
DisablePulseDetection (dg_DisablePulseDetection)	160
DisableE1Thread (dg_DisableE1Thread)	161
DisableEchoCancelation (dg_DisableEchoCancelation) .	162
DisableSilenceDetection (dg_DisableSilenceDetection) .	163
EnableAnswerDetection (dg_EnableAnswerDetection) ...	164
EnableAGC (dg_EnableAGC)	165
EnableCallProgress (dg_EnableCallProgress)	166
EnableDebug (dg_EnableDebug)	168
EnableDTMFFilter (dg_EnableDTMFFilter)	169
EnableE1Thread (dg_EnableE1Thread)	171
EnableEchoCancelation (dg_EnableEchoCancelation) ...	172
EnableInputBuffer (dg_EnableInputBuffer)	173
EnablePulseDetection (dg_EnablePulseDetection)	175
EnableSilenceDetection (dg_EnableSilenceDetection) ...	176
Flash (dg_Flash)	177
ForceSingleSpan (dg_ForceSingleSpan)	179
GetE1Number (dg_GetE1Number)	180
GetLoggerCallType (dg_GetLoggerCallType)	181
GenerateMF (dg_GenerateMF)	182
GetAbsolutePortNumber	183
GetAlarmStatus (dg_GetAlarmStatus)	184
GetCallerID (dg_GetCallerID)	186
GetCardBus (dg_GetCardBus)	187
GetCardInterface (dg_GetCardInterface)	188
GetPortInterface (dg_GetPortInterface)	189
GetCardType (dg_GetCardType)	189
GetCardSlot (dg_GetCardSlot)	190
GetCardNumber (dg_GetCardNumber)	191
GetCardPortsCount (dg_GetCardPortsCount)	192
GetCardsCount (dg_GetCardsCount)	193
GetDigits (dg_GetDigits)	193
GetDriverEnabled (dg_GetDriverEnabled)	195
GetE1ThreadStatus (dg_GetE1ThreadStatus)	196
GetPortsCount (dg_GetPortsCount)	197
GetPortStatus (dg_GetPortStatus)	198

GetPortCardType (dg_GetPortCardType)	199
GetPlayFormat (dg_GetPlayFormat)	199
GetRecordFormat (dg_GetRecordFormat)	200
GetRelativeChannelNumber	201
GetVersion (dg_GetVersion)	202
h100_AllocPort (dg_h100_AllocPort)	203
h100_DisconnectAll (dg_h100_DisconnectAll)	204
h100_EnablePort (dg_h100_EnablePort)	205
h100_HalfDuplexDisconnect	206
h100_HalfDuplexConnect (dg_h100_HalfDuplexConnect)	207
h100_FullDuplexConnect (dg_h100_FullDuplexConnect)	209
h100_FullDuplexDisconnect	210
h100_GetFreePortByRange	211
h100_SetDefault (dg_h100_SetDefault)	212
HangUp (dg_HangUp)	213
IsCallInProgress	214
IdleAbort (dg_IdleAbort)	215
IdleStart (dg_IdleStart)	216
IdleSettings (dg_IdleSettings)	217
IsPlaying (dg_IsPlaying)	218
IsRecording (dg_IsRecording)	219
LocalBridgeConnect (dg_LocalBridgeConnect)	220
LocalBridgeDisconnect (dg_LocalBridgeDisconnect)	221
MakeCall	222
MenuAbort	223
MenuErrorSettings	224
MenuStart	225
MicOff (dg_MicOff)	227
MicOn (dg_MicOn)	228
PauseInputBuffer (dg_PauseInputBuffer)	229
PhoneOff (dg_PhoneOff)	230
PhoneOn (dg_PhoneOn)	231
PlayBuffer (dg_PlayBuffer)	232
PlayCardinal	234
PlayCurrency	235
PlayDate	236
PlayFile (dg_PlayFile)	237
PlayList	239

PlayListAdd	240
PlayListClear	241
PlayListGetCount	242
PlayListRemoveItem	242
PlayNumber	243
PlayTime	244
PickUp (dg_PickUp)	245
PromptAbort	246
PromptSettings	247
PromptStart	248
R2AskForID (dg_R2AskForId)	250
R2AskForGroupII (dg_R2AskForGroupII)	251
R2SendGroupB (dg_SendGroupB)	253
ReadDigits (dg_ReadDigits)	254
RecordPause (dg_RecordPause)	255
RecordFile (dg_RecordFile)	256
ResetError (dg_ResetError)	258
ResetPortResource (dg_ResetPortResource)	259
dg_SetEventCallback	260
SendR2Command (dg_SendR2Command)	261
SetAlarmMode (dg_SetAlarmMode)	263
SetAnswerSensitivity (dg_SetAnswerSensitivity)	264
SetAnswerThreshold (dg_SetAnswerThreshold)	265
SetAudioInputCallback (dg_SetAudioInputCallback)	266
SetCardDetections (dg_SetCardDetections)	267
SetCardSyncMode (dg_SetCardSyncMode)	269
SetCallAfterAnswer	270
SetCallAfterPickup	272
SetCallBusyPhrase	273
SetCallBusyReturnFlash	274
SetCallFlashTime	275
SetCallNoAnswerPhrase	276
SetCallNoAnswerRingCount	277
SetCallNoAnswerReturnFlash	278
SetCallPauseBeforeAnalysis	279
SetCallStartFlash	280
SetCallWaitForDialTone	281
SetDetectionType (dg_SetDetectionType)	282

SetDialDelays (dg_SetDialDelays)	284
SetDigitGain(dg_SetDigitGain)	285
SetDigitFrequency(dg_SetDigitFrequency)	287
SetDTMFAttenuating (dg_SetDTMFAttenuating)	288
SetDTMFConfig (dg_SetDTMFConfig)	289
SetE1CRC4Option (dg_SetE1CRC4Option)	290
SetFastDetection (dg_SetFastDetection)	291
SetFaxFrequencies (dg_SetFaxFrequencies)	293
SetFramerLoop (dg_SetFramerLoop)	294
SetFrequency (dg_SetFrequency)	296
SetGSMMode (dg_SetGSMMode)	297
SetH100 (dg_SetH100)	298
SetPlayFormat (dg_SetPlayFormat)	302
SetPortGain (dg_SetPortGain)	304
SetPortID (dg_SetPortID)	305
SetPortChatLog(dg_SetPortChatLog)	306
SetRecordGain (dg_SetRecordGain)	307
SetRecordFormat (dg_SetRecordFormat)	308
SetSilenceThreshold (dg_SetSilenceThreshold)	310
SetStartE1RxCount (dg_SetStartE1RxCount)	311
SetNextE1RxCount (dg_SetNextE1RxCount)	312
SetTwist (dg_SetTwist)	314
ShutdownVoicerLib (dg_ShutdownVoicerLib)	315
StartVoicerLib (dg_StartVoicerLib)	316
StopPlayBuffer (dg_StopPlayBuffer)	318
StopPlayFile (dg_StopPlayFile)	319
StopRecordFile (dg_StopRecordFile)	320
3 Eventos	321
OnAfterDial (EV_AFTERDIAL)	322
OnAfterFlash (EV_AFTERFLASH)	323
OnAfterMakeCall	323
OnAfterPickUp (EV_AFTERPICKUP)	324
OnAnswerDetected (EV_ANSWERED)	325
OnAudioSignalDetected (EV_AUDIO_SIGNAL)	326
OnBusyDetected (EV_BUSY)	327
OnCallerID (EV_CALLERID)	328
OnCalling (EV_CALLING)	328

OnCallStateChange	329
OnDialToneDetected (EV_DIALTONE)	330
OnDigitDetected (EV_DTMF)	331
OnDigitsReceived (EV_DIGITSRECEIVED)	332
OnErrorDetected (EV_ERRORDETECTED)	333
OnE1Alarm (EV_E1_ALARM)	334
OnE1GroupB (EV_GROUP_B)	335
OnE1FramerResponse (EV_FRAMER)	336
OnE1StateChange (EV_E1CHANGESTATUS)	337
OnFaxDetected (EV_FAX)	338
OnLineOff (EV_LINEOFF)	339
OnLineReady (EV_LINEREADY)	340
OnLoggerEvent (EV_LOGGEREVENT)	341
OnMenu	343
OnPlayStart (EV_PLAYSTART)	343
OnPlayStop (EV_PLAYSTOP)	344
OnPrompt	345
OnR2Received (EV_R2)	346
OnRecording (EV_RECORDING)	347
OnRecordStart (EV_RECORDSTART)	348
OnRecordStop (EV_RECORDSTOP)	349
OnRingDetected (EV_RINGS)	350
OnSilenceDetected (EV_SILENCE)	350
4 Propriedades exclusivas do ActiveX	351
ConfigPath	352
StockSigPath	352

Parte VI Utilizando o Configurador da placa E1 356

1 Iniciando o Programa	357
2 Configurando as Placas	358
3 Salvando as configurações	362
4 Testando a Placa em Loop	363
5 Verificando o Log	366

Índice Remissivo.....0

Parte



Introdução

1 Introdução

Bem vindo ao Guia do Programador da VoicerLib 4!

A VoicerLib 4 é a versão da VoicerLib para a nova família de placas DigiVoice tanto para Windows como para Linux. As placas suportadas são:

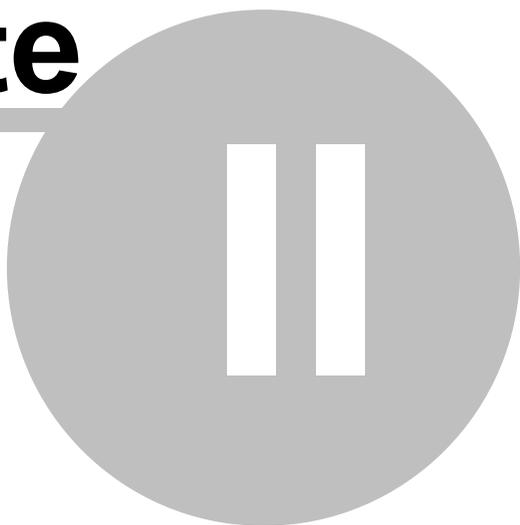
- **VB6060PCI** - todos os modelos das placas digitais E1
- **VB0408PCI** - placas FXO de 4 ou 8 canais

A partir de agora, todas as novas placas CTI da DigiVoice serão suportadas pela versão 4 da VoicerLib, tanto na plataforma Windows como Linux.



As placas **VoicerBox PCI/1** e **VoicerPhone PCI/4** não são suportadas por esta versão da VoicerLib. Para estes modelos deve ser utilizada a **VoicerLib 2.x** para Windows e a **VoicerLib 3.1.x** para Linux.

Parte



Recursos para o Desenvolvedor

2 Recursos para o Desenvolvedor

Ao adquirir as placas DigiVoice e a VoicerLib, o desenvolvedor recebe uma coleção de exemplos prontos com o objetivo de explicar o funcionamento das diversas funções da VoicerLib.

Além disso, o desenvolvedor tem disponível a área de suporte do site da DigiVoice na internet, através do endereço **<http://www.digivoice.com.br/suporte>**. Lá, estão disponíveis um fórum de discussões, arquivos para download (exemplos atualizados, etc..) e uma seção de Perguntas Frequentes para ajudar o desenvolvedor nas dúvidas mais comuns.

É muito importante também ler com atenção toda a seção **Guia de Programação** deste manual pois contém toda a base necessária para programar com as placas DigiVoice, importante para novatos ou para desenvolvedores que já trabalharam com outros hardwares.

Parte



Primeiros Passos

3 Primeiros Passos

Neste capítulo será discutido os passos necessários para instalar a VoicerLib e executar os primeiros procedimentos para verificar se tudo está funcionando a contento.

3.1 Windows

3.1.1 Instalação

Requisitos mínimos de instalação:

- Windows XP Professional SP1/SP2
- Windows 2000 Professional/Server com SP2
- Windows 2003 Server
- Celeron 1 Ghz com 256Mb de memória

Existem dois programas de instalação da VoicerLib4:

- **setup_vlib.exe** - Programa de instalação para o desenvolvedor, contendo os drivers, documentação e exemplos.
- **setup_kit.exe** - Programa voltado à máquinas de produção. Instala somente os drivers.

A instalação física das placas DigiVoice é explicada pelo manual **Kit Integrador** que é fornecido impresso acompanhando o hardware. Quanto à instalação do software leia os tópicos a seguir deste manual.

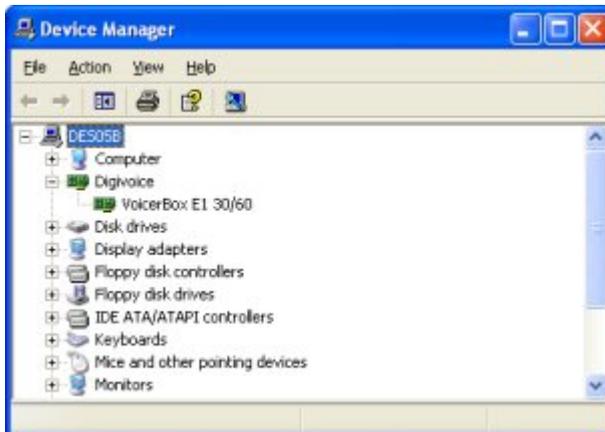
Se você já instalou fisicamente a placa, o Windows a reconhecerá na primeira inicialização. Se você não instalou o

software ainda, simplesmente cancele a tela que o Windows mostrar.

Em seguida, localize e execute o arquivo **setup_vlib.exe**. Após seguir os passos indicados nas telas, reinicie o computador quando isso for solicitado pelo programa de instalação.

Ao reiniciar, o Windows mostrará novamente que detectou um novo hardware (Controlador Multimedia). Desta vez, siga os passos para instalação do hardware. O Windows perguntará se você quer instalar automaticamente ou a partir de um caminho específico. Escolha a segunda opção, apontando para o diretório **C:\Arquivos de Programas\VoicerLib4**.

Isto deverá ser suficiente para que o Windows reconheça a placa como **Digivoice VB6060PCI** ou **Digivoice VB0408PCI**. Se tudo estiver correto, a placa estará pronta para ser utilizada. Caso tenha alguma dúvida, acesse o Gerenciador de Dispositivos do Windows, que deverá apresentar um novo item Digivoice:



O programa de instalação criará o grupo de programas **Digivoice VoicerLib4** contendo:

- **Configurador da placa E1** - Programa de configuração e testes
- **Programa de diagnósticos Vlib_Diag** - Programa de testes para a placa analógico
- **Manual do Programador** (Este manual)
- **Link para o site da DigiVoice**
- **Atalho para remover a instalação**



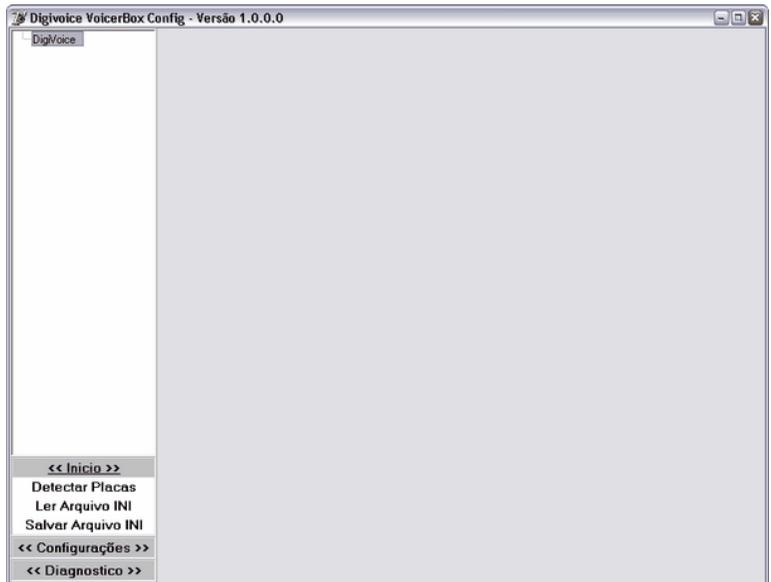
IMPORTANTE: Na pasta **C:\Arquivos de Programas\VoicerLib4\Samples** estão disponíveis diversos exemplos que poderão ser estudados para complementar as informações do manual.

3.1.2 Testando a placa instalada

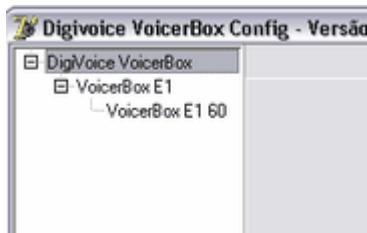
Placa E1

Para verificar se a placa foi corretamente instalada e está operacional, execute o programa **Configurador da placa E1** que está no menu Iniciar do Windows, sob o grupo **Digivoice VoicerLib4**.

A seguinte tela aparecerá:



A primeira ação a fazer é clicar sobre a opção **Detectar Placas**, no menu apresentado na parte inferior esquerda da janela. O Configurador iniciará a detecção e inicialização das placas. Se tudo estiver correto, a placa aparecerá sob o título Digivoice, no canto superior esquerdo da tela, como mostrado a seguir:



Se houverem mais placas instaladas no computador, vários itens **VB6060PCI** aparecerão.

Com isso, é possível ter certeza que as placas foram corretamente instaladas e estão operacionais. Para maiores detalhes de configuração e monitoração usando o **Configurador**, leia o capítulo "**Utilizando o Configurador da placa E1**".

3.2 Linux

3.2.1 Instalação

Requisitos mínimos de instalação:

- Celeron 1 Ghz com 256Mb de memória
- Distribuição Linux com kernel versão **2.4.x.** ou **2.6**

O arquivo que contém todo o material necessário chama-se **voicerlib-4.x.x.tar.gz** (4.x.x será a versão corrente) sendo que para descompactá-lo, utilize o comando:

```
tar xvzf voicerlib-4.x.x.tar.gz
```

Com isso, o diretório **voicerlib-4.x.x** será criado contendo os fontes da voicerlib e do device driver.

- voicerlib-4.x.x – Contém os arquivos relativos à API

```
|
|___/doc
|       |__ Documentação
|
|___/firmware
|       |__ Arquivos dos firmwares das placas e outros
de configuração (CallProgress, etc...)
|
|___/samples/dll_so
|       |__ c           - programa exemplo
```

```
desenvolvido em ANSI C
|   |__ dg_utils   - programas para testes de
r2d, callprogress e lista de canais reconhecidos
pelo sistema
|
|__ /driver/linux
|   |__ Código-fonte dos device drivers para as
placas Digivoice.
|
|__ /src_linux
|   |__ Código-fonte da VoicerLib (shared object)
para Linux
|
|__ /src_common
|   |__ Código-fonte da VoicerLib - arquivos
comuns multiplataforma
|
```



Para trabalhar com o **Asterisk**, é necessário também baixar e instalar o **dgvchannel - channel driver para Asterisk**. Acesse o site <http://www.digivoice.com.br>.

3.2.2 Preparando o ambiente

Para estar apto a compilar a VoicerLib no seu ambiente será necessário que esteja disponível um sistema de desenvolvimento (gcc, make, etc...) bem como os **fontes do kernel**. Todos estes arquivos normalmente encontram-se disponíveis no CD original de sua distribuição. A VoicerLib foi desenvolvida e testada em versões **2.4** e **2.6** do kernel, utilizando o compilador gcc versão 3.0.1 ou maior (A versão 2.95 do gcc causará diversos erros de compilação). O gcc 4

também tem sido utilizado sem problemas.

Para compilar o exemplo em C, será necessário a biblioteca **ncurses-devel** versão 5.3 ou maior.

A maioria dos passos necessários para colocação da VoicerLib e aplicativos para funcionar precisam ser executados com o usuário **root**, portanto todo o cuidado é necessário para não danificar o seu sistema. Lembramos que, em ambiente de produção, não é necessário que o usuário tenha direitos de administrador (aliás o contrário e recomendado).

A partir da versão 4.0.5, é possível compilar e instalar todas as bibliotecas, drivers e aplicativos através do comando:

```
make; make install; make config
```

a partir do diretório raiz.

Após a execução destes comandos, o ambiente já estará pronto para operar. Um teste de verificação útil é a execução do programa **dglist** que lista todos os canais reconhecidos pelo sistema.

Nos próximos itens deste capítulo será explicado a compilação e instalação de cada módulo separadamente, porém a sua execução só será necessária caso haja algum problema no passo que acabou de ser explicado.

3.2.3 Módulos do kernel

Os módulos ou device drivers são os responsáveis em criar e gerenciar a troca de dados entre o hardware e os aplicativos. Caso eles não estejam funcionando corretamente, nada mais

funcionará.

No sub-diretório **driver/linux** encontram-se todos os arquivos necessários para compilar e instalar os módulos. O mesmo device driver é responsável por gerenciar todos os modelos de placas.

Para compilar o módulo desejado, você precisa executar os seguintes comandos:

```
make          (compila o módulo)
```

```
make install  (instala o módulo no diretório correto e  
carrega-o na memória)
```

```
make config  (instala o módulo na inicialização do  
sistema - testado em Debian e RedHat e derivações)
```

Após a compilação (make) deverá ser gerado o arquivo **vlibd.o** ou **vlibd.ko**

Para carregar o módulo na memória manualmente depois de reiniciar (se não tiver sido executado o make config), execute novamente o **make install**.

Para verificar se o módulo foi instalado corretamente, digite em linha de comando '**dmesg**'. Este programa exibe informações dos device drivers instalados. Se tudo estiver correto, você verá uma mensagem como a apresentada a seguir:

```
vlibd: Driver was load successfully!!
```

Se a mensagem exibida for diferente, aparentando algum tipo de erro, verifique e repita os passos anteriores ou mesmo se a placa está fisicamente instalada.



ATENÇÃO USUÁRIOS RED HAT 3.2 AS/ES ou CentOS-3. Para compilar o módulo execute:

`make RH3=1`

Os outros procedimentos não são alterados

3.2.4 Compilando a VoicerLib

Como a VoicerLib é multiplataforma, existem diretórios específicos para plataforma Linux e Windows. Para compilar a VoicerLib para Linux, vá para o diretório `src_linux` e execute os seguintes comandos:

```
make
```

```
make install
```

O comando **make install** copiará a o arquivo resultante (`libdigivoice.so.4.0`) para o diretório **`/usr/lib`** e o tornará disponível para todas as ferramentas de desenvolvimento como uma *shared object*. Também copia os *headers* (`*.h`) para o diretório padrão **`/usr/include/voicerlib`** permitindo o uso da `voicerlib` por outros aplicativos. Os arquivos de configuração e firmware são instalados em **`/var/lib/voicerlib`**. Todos os aplicativos de teste irão buscar os arquivos de firmware neste diretório como padrão.

O funcionamento da VoicerLib será tratado no capítulo **Guia de Programação** e seus comandos estão listados no capítulo **Guia de Referencia**.

3.2.5 Exemplo em linguagem C

O programa no diretório **samples/dll_so/c** chama-se **vlib_diag**. Trata-se de um programa em modo caracter (console) que permite interagir com as placas Digivoice. É a melhor forma de testar o funcionamento da placa bem como estudar o funcionamento da API.

Este projeto necessita da biblioteca **ncurses-devel** com versão maior ou igual a 5.3.

Para compilar, digite 'make'.

Ao executar `./vlib_diag` voce poderá efetuar diversos testes com as placas E1 ou FXO.

Verificando placas instaladas

Existe também no diretório **samples/dll_so/dg_utils** o programa **dglist** que lista todas as placas e canais instalados, conforme a ordem que foram detectados pelo sistema operacional.

Especificamente para as placas E1, o programa de teste **dgr2loop** permite efetuar ligações dos canais E1 em loop. Este programa também esta no diretórios **dg_utils**.

Parte



IV

Guia de Programação

4 Guia de Programação

A VoicerLib 4 pode ser utilizada tanto no ambiente Windows como no Linux, sendo que existem semelhanças e algumas diferenças no modo de funcionamento da biblioteca nas duas plataformas.

Como semelhanças, a VoicerLib tem uma camada de funções primitivas que foram implementadas na forma de uma API em C. Em Linux, utiliza-se dos *Shared Objects* e em Windows de *DLLs* para interfacear com as aplicações. Desta forma, apesar de ter sido construída em C, a **VoicerLib API** permite sua utilização em qualquer ferramenta de desenvolvimento que tenha acesso à *DLLs* ou *SOs*. Para isso, é necessário que os headers da **VoicerLib API** sejam traduzidos de C para a linguagem desejada.

Na camada de API, a VoicerLib fornece todos os métodos primitivos de acesso aos recursos da placa. Entenda-se por métodos primitivos, todas as funções que não se enquadram na classe de funções especiais, discutidas mais adiante.

A **diferença** é que, para oferecer uma compatibilidade quase total com a VoicerLib série 2.x, a DigiVoice fornece, **somente para ambiente Windows**, um ActiveX, chamado VoicerLib.OCX, que tem por objetivo minimizar os esforços de migração de aplicações desenvolvidas, disponibilizando praticamente todo o conjunto de funções existentes na versão **2.x**. Nesse conjunto, estão incluídas todas as funções especiais (MakeCall, PlayList, etc...). Aos poucos, essas funções serão disponibilizadas em todas as camadas, como por exemplo, as funções de Idle, que já estão disponíveis na DLL/SO e no ActiveX.

Na versão 2.x, o componente chama-se VoicerLib200.OCX enquanto nessa nova série 4, foi chamado simplesmente de

VoicerLib.OCX. Dessa forma, é possível manter o mesmo aplicativo fazendo referência aos dois componentes, desde que alguns cuidados sejam tomados, dependendo da ferramenta de desenvolvimento utilizada.

A seguir serão mostrados os conceitos específicos de cada modo de programação. Mais a frente no manual, os conceitos diversos serão explicados independente do uso da API ou do ActiveX, sempre sendo feito menção dos métodos e/ou funções utilizadas em cada modo.

Se você programa em Linux ou pretende programar em C para Windows acessando as funções primitivas, leia somente a seção **VoicerLib API**. Caso a plataforma utilizada seja Visual Basic, Delphi, C#, etc... recomenda-se a utilização do ActiveX, por isso leia a seção **VoicerLib ActiveX**.



ATENÇÃO: Para facilitar a documentação as explicação das funções API e ActiveX são feitas conjuntamente, sempre colocando a função API e o método ActiveX entre parênteses, salvo situações em que as diferenças são explícitas.

4.1 Conceitos Básicos - API

Para os desenvolvedores acostumados com a VoicerLib em Windows, o conceito predominante eram as propriedades, eventos e métodos, comuns às linguagens *orientada a eventos* como o Visual Basic. Como a VoicerLib API foi desenvolvida

para ser um Shared Object (Linux) e uma DLL (Windows), o funcionamento é um pouco diferente.

Numa tradução simplificada, as **propriedades** e **métodos** serão acessados através de funções comuns com nome e parametrização similar à versão em Windows (ex.: o método *PickUp* tornou-se a função *dg_pickup*). Todas as funções da VoicerLib API começam com **dg_** para evitar conflitos de nomes com outras bibliotecas dinâmicas que por ventura existam no ambiente.

Os **eventos** existentes no ActiveX em Windows eram chamados pela VoicerLib de forma transparente, bastando ao programador associar uma função a um determinado evento (ex.: *OnRingDetected* -> *TrataRing*). Em um shared-object os eventos continuam existindo, porém, a rotina que associa determinado evento (ou mensagem) a uma porção de código (ou função) passa a ser de responsabilidade do programador.

Na prática, seria necessário criar uma função que receba todos os eventos da placa. Esta função será chamada de **Gerenciador de Eventos**. Um "esqueleto" dessa função é mostrado a seguir:

```
void ReceiveEvents(void *context_data)
{
    struct dg_event_data_structure *EventContext;

    /* Copy received Data */
    EventContext = ((struct
dg_event_data_structure*)context_data);

    switch (EventContext->command)
    {
        case EV_RING:
            //recebeu ring
            //....
        case EV_DTMF:
            //recebeu digito
```

```
        //...  
    }  
}
```

Mesmo conhecendo pouco a linguagem C, é fácil de perceber que o **Gerenciador de Eventos** recebe um parâmetro chamado **context_data**. Neste parâmetro estarão as informações pertinentes a cada evento (porta, dígito, etc...). Caberá ao programador analisar o conteúdo de **context_data->command** para saber qual evento ocorreu. A variável **context_data->port** indica a porta que ocorreu o evento e a variável **context_data->data** fornece um dado relativo ao evento (alguns eventos não possuem dado associados).

Context_data é definido como:

```
struct dg_event_data_structure  
*context_data;
```

sendo que **dg_event_data_structure** está definido no arquivo **voicerlib.h** conforme segue:

```
typedef struct {  
    unsigned short command;  
    unsigned short data;  
    unsigned short port;  
    unsigned short data_aux;  
    unsigned short card;  
} dg_event_data_struct;
```

Um primeiro conceito que já é conhecido dos desenvolvedores Windows e que foi mantido na versão Linux é que todo o funcionamento da VoicerLib é **assíncrono**. Isto significa que ao executar a chamada a uma determinada função, o programa seguirá seu fluxo normal. As respostas às funções são manipuladas através de uma função eleita para tratar todas as mensagens, que no exemplo é a **ReceiveEvents**.

Por exemplo, quando a função **dg_GetDigits** é chamada, o

programa continua normalmente e só após os dígitos serem recebidos (ou der time-out) a VoicerLib chamará a função `ReceiveEvents` passando no `context_data` a mensagem `EV_DIGITSRECEIVED` e o status de retorno da função.

Este tipo de funcionamento é muito importante pois a biblioteca tem que gerenciar vários canais simultaneamente. Se a aplicação ficasse presa na execução de uma função, não conseguiria tratar os outros eventos dos outros canais, por isso o tratamento de determinada mensagem dentro do *switch-case* deve ser o mais rápido possível.

A função **ReceiveEvents** na verdade pode ter qualquer nome mas sempre deverá receber o endereço da estrutura `context_data`, independente da linguagem de programação utilizada.

No início do programa, **antes** de iniciar a placa com o método **dg_StartVoicerLib**, é obrigatório associar a função de tratamento de mensagens à VoicerLib através da função **dg_SetEventCallback**. Se a associação não for feita corretamente, o programa gerará falhas de execução (Segmentation Fault).

Então, um *esqueleto* de programa utilizando a VoicerLib API deverá ter a seguinte característica:

```
void ReceiveEvents(void *context_data)
{
    //tratamento de todos os eventos vindos da
placa
}

void main() //Inicio do programa
{
    //primeiro associa a função callback
dg_SetEventCallback(ReceivedEvents,&event_cont
ext);

    //Inicia a voicerlib
```

```
        dg_StartVoicerLib("../../firmware");
    }

void finaliza()
{
    //Finaliza o driver
    dg_ShutdownVoicerlib();
}
```

Consulte o Guia de Referência VoicerLib API no próximo capítulo a respeito de todas as funções disponíveis.



As placas VB6060PCI têm um comportamento diferente em relação à vários tópicos abordados neste capítulo, porém, a DigiVoice procurou deixar a manipulação destas características da maneira mais similar às placas FXO, visando minimizar o tempo de aprendizado. A leitura do tópico **Programação da Placa VB6060PCI** é de suma importância para a compreensão destas diferenças e do modo de operação a serem utilizados.

4.2 Conceitos Básicos - ActiveX

O componente VoicerLib é baseado na estrutura de um looping infinito, que fica monitorando os eventos que acontecem no hardware (ring, tons, etc...) e recebendo e passando os comandos gerados a partir da aplicação (propriedades e

métodos).

Devido a esta característica, a maioria das funções da biblioteca são assíncronas. Isto significa que ao executar a chamada a um determinado método, o programa seguirá seu fluxo normal. As respostas aos métodos são manipuladas através de eventos específicos relativos a cada acontecimento.

Por exemplo, quando o método GetDigits é chamado, o programa continua normalmente e só após os dígitos serem recebidos (ou der time-out) que a resposta ao GetDigits será tratada dentro do evento OnDigitsReceived

Este tipo de funcionamento é muito importante pois a biblioteca tem que gerenciar vários canais simultaneamente.

Se a aplicação ficasse presa na execução de um método, não conseguiria tratar os outros eventos dos outros canais. Lembre-se que é possível que um canal esteja reproduzindo a mensagem enquanto outro esteja recebendo um ring.

4.3 Guia de Migração de Versões Anteriores

Apesar de ter como objetivo manter a biblioteca o mais compatível possível com as versões anteriores, algumas alterações nos fontes serão necessárias para se atualizar aplicações para a nova versão, principalmente devido às novas tecnologias empregadas nas placas VB0408PCI e VB6060PCI. Essas alterações fizeram com que novos métodos fossem criados, outros fossem excluídos e alguns outros tiveram alterações nos parâmetros que recebem.

Recomendamos a leitura deste capítulo antes de efetuar a adaptação de sistemas desenvolvidos com versões anteriores

da VoicerLib.

1. Métodos/Propriedades removidas

- **SetAnswerSensitivity** – agora é um parâmetro da função ConfigCallProgress (*Maiores detalhes no capítulo Supervisão de Linha*)
 - **SetAnswerThreshold** - Esse tipo de configuração não é mais necessária (*veja também SetSilenceThreshold*)
 - **SetVolume** - sem função nas novas placas - exclusivo para placa de 1 canal com headset
 - **SetImpedance** - sem função nas novas placas - exclusivo para placa de 1 canal com headset
 - **SetDTMFAttenuatingHigh/Low** - A configuração de ganho de dígitos é mais ampla e feita pela função SetDigitGain
 - **SetDTMFTwist/SetToneTwist** - Devido aos novos algoritmos não é mais necessária a configuração de twists
 - **Propriedade CardType** - Como agora é permitida a *mistura* de placas diferentes, não é possível utilizar uma propriedade para atribuir ou ler o tipo de placa
 - **SetFrequencyTime** - agora é um parâmetro da função ConfigCallProgress (*Maiores detalhes no capítulo Supervisão de Linha*)
 - **AutoClearDigits** - A propriedade foi retirada por causar certa confusão no funcionamento dos métodos que detectavam dígito. Agora o programador deverá sempre apagar explicitamente os dígitos com a função ClearDigits, sempre que for necessário
 - **Formato ffSig** está sendo removido gradualmente por ser idêntico em tamanho ao ffWaveULaw ou ffWaveALaw. Para manter a compatibilidade, ao escolher este formato, será utilizado no lugar o ffWaveULaw.
 - Propriedades **DelayDot**, **DelayComma** e **DelaySemicolon** foram removidas, sendo agora necessário utilizar o método **SetDialDelays**.
-

2. Mudanças de nome e/ou parâmetros

- **SetFastDetection** – Agora para setar o modo rápido, é preciso chamar esta função (o parâmetro enable pode receber os valores DG_ENABLE /DG_DISABLE). O modo de detecção rápida foi inserido apenas na versão 3 da VoicerLib e para atividades especiais.
- **EnableDetections** mudou para **SetDetectionType** – Agora a voicerlib não detecta nenhum tipo como padrão. O programador deve chamar explicitamente esta função sempre quando for necessário. Os métodos de detecção mudaram bastante portanto leia atentamente o guia de referência sobre esta função.
- Evento **OnAnswerDetected** recebe um novo parâmetro chamado AnswerType, indicando se o atendimento foi por detecção de áudio (AUDIO_DETECTED) ou timeout (TIMEOUT_DETECTED).
- **SetCardSyncMode** para placas E1 - Para setar o sincronismo deve ser chamado o SetCardSyncMode passando como parâmetros a placa e o tipo de sincronismo SYNC_INTERNAL, SYNC_LINE_A, SYNC_LINE_B
- **dg_StartVoicerLib** na DLL e no SO só tem o path como parâmetro (Isso não é aplicado ao ActiveX) e agora retorna EXIT_SUCCESS (0) no caso de sucesso e não mais 99.
- **SetBusConnection** foi dividida em **ConnectBus** e **DisconnectBus** para facilitar o entendimento do uso destas funções.

3. Novas funções / eventos

- **Evento OnAudioSignal** indicando para a aplicação o tipo de áudio detectado. Útil para análises de tipos de tom e suas cadências
- **Funções de Canais virtuais** permitindo utilizar a numeração de portas independentemente da posição física da porta na placa
 - **DefinePortResource** – associa uma determinada porta

(lógica) a uma placa/canal físico

- **ResetPortResource** – Reseta a configuração, sendo assumida o padrão de inicialização

Funções de CallProgress - O CallProgress agora é mais completo e versátil e é necessário criar a *thread* de controle de CallProgress antes de usá-lo. Leia atentamente o capítulo de **Supervisão de Linha** - Funções relacionadas: **CreateCallProgress**, **ConfigCallProgress**, **DestroyCallProgress** e **EnableCallProgress**

- **EnableAnswerDetection/DisableAnswerDetection** – a função para habilitar deve ser chamada explicitamente após chamad o EnableCallProgress, para que a thread de controle de callprogress gere o evento de atendimento.
 - **EnableInputBuffer/DisableInputBuffer** – Agora para efetuar a gravação de canais na voicerlib, é necessário habilitar o envio de amostras da placa através da função EnableInputBuffer. Isso deve ser feito antes de se chamar o RecordFile. Esta alteração foi feita para suportar streaming de áudio das placas para as aplicações diretamente.
 - **PlayBuffer / StopPlayBuffer** - Envia amostras diretamente para a placa reproduzir. Útil para aplicações VOIP.
 - **SetGSMMode** podendo assumir GSM_DIGIVOICE ou GSM_RAW (padrao). Ambos os formatos têm a mesma codificação sendo que no padrão Digivoice é inserido um cabeçalho no arquivo.
 - **GetPortCardType** - Devolve o tipo de placa a partir da porta informada
 - Implementada nova **detecção de silêncio**, através das funções **EnableSilenceDetection/DisableSilenceDetection** e do evento **OnSilenceDetection (EV_SILENCE)**
-

4.4 Inicializando os Serviços

Sempre que iniciar a aplicação, antes de executar qualquer outra função, é necessário iniciar os serviços (device driver). Isto pode ser feito através da função **dg_StartVoicerLib** (*StartVoicerLib*).

A VoicerLib detecta todas as placas disponíveis no computador e as inicializa na ordem de colocação dos slots do computador. Com isso é possível ter, por exemplo, uma placa E1 de 60 canais e uma placa analógica de 8 canais. Se a placa E1 for lida primeiro, receberá as portas de 1 a 60 e a placa analógica de 61 a 68. A posição física dos slots dos computadores variam de fabricante, modelo e sistema operacional, portanto é necessário que a configuração na máquina de produção seja versátil para se adaptar a essas diferenças.



ATENÇÃO: No mesmo computador, uma vez instaladas, as placas sempre serão reconhecidas na mesma ordem.

Exclusivo API: Como a API não gera eventos, como conhecido nos componentes visuais, é necessário criar uma função para manipular os eventos da placa, conforme foi explicado no tópico anterior: Conceitos Básicos. Lembre-se que a associação da CALLBACK de eventos deverá ser feita **antes** de inicializar o driver.

Como a inicialização é a primeira coisa a ser feita, recomenda-se sua colocação no início da aplicação. Não é prática recomendável ficar inicializando e finalizando a VoicerLib diversas vezes na aplicação.

4.5 Finalizando os Serviços

Antes de fechar a aplicação é necessário chamar a função **dg_ShutdownVoicerLib (ShutdownVoicerLib)**, que finaliza todos os serviços e reseta a placa.

Caso a função não seja chamada, recursos de memória continuarão alocados mesmo após a finalização do aplicativo. Neste caso será necessário reiniciar o computador para que estes recursos sejam liberados.

A não utilização desta função poderá causar travamento no sistema operacional ou algum comportamento imprevisível.

É possível verificar se os serviços do hardware foram finalizados corretamente através do retorno da função, que deve ser 0.

O melhor local para se colocar esta função é em algum evento finalizador, que antecede o encerramento da aplicação (Unload, OnClose, etc...).

4.6 Detectando Ring

A VoicerLib permite detectar quando tem uma ligação entrante chegando através do ring.

Sempre quando o ring for detectado, a função que atua como **Gerenciador de Eventos** receberá na variável **context_data->command** o valor **EV_RINGS** (evento **OnRingDetected** no ActiveX).

Nas placas FXO (VB0408PCI), como o Ring ocorre em situações bem específicas a sua detecção está sempre

habilitada não havendo métodos para ligar ou desligar.

Já na placa **VB6060PCI**, o Ring é um evento que ocorrerá sempre quando a *thread* de controle R2D estiver habilitada (*Maiores detalhes no capítulo "Programação da Placa E1 VB6060PCI"*).

4.7 Atendendo e Desligando

Para atender a ligação ou tomar uma linha para discagem, a função que deve ser utilizada é o **dg_Pickup (PickUp)**. A função pede 2 parâmetros. O primeiro é a porta de atendimento que vai de 1 a N. O segundo parâmetro é uma pausa após o atendimento, em milissegundos (1000 = 1 segundo). Esta pausa é útil em aplicações de atendimento automático em instalações com bloqueio de chamada a cobrar, etc... Ela deve ser utilizada em conjunto com o evento EV_AFTERPICKUP (OnAfterPickUp).

Para desligar a ligação, a função utilizada é o **dg_Hangup (HangUp)**. Deve ser passado somente a porta que se deseja desligar. Ambas as funções retornam zero caso tenham sido executadas com sucesso.

Um diferencial em relação aos modelos anteriores é que quando a ligação é atendida ou desligada os eventos **OnLineReady** e **OnLineOff** são gerados mesmo que a linha/ramal não esteja em paralelo.

A placa **VB6060PCI (E1)** é um caso especial, pois o **dg_Pickup (PickUp)** refere-se a *solicitação de ocupação* de um canal. Quando executado no contexto da *thread* (dg_CreateE1Thread), o *pickup* é utilizado para ocupar um canal na ligação sainte ou atender uma ligação entrante. Se a *thread* não estiver iniciada, o

signal de ocupação também é enviado, porém todo o tratamento deverá ser feito pela aplicação final.

4.8 Supervisão de Linha

A supervisão de linha permite ao programador identificar e tratar diversos eventos relativos a sinais enviados pela linha, a saber:

- Sinal de Ocupado
- Detecção de Fax
- Detecção de sinal de discagem (tom de linha)
- Detecção de sinal de chamada (*ringback*)
- Detecção de Atendimento

Nas placas digitais (E1) não existe propriamente um sinal de ocupado, tom de discagem ou sinal de chamada já que toda a parte de sinalização é feita pelo protocolo R2D, porém, a VoicerLib *simula* estes sinais e eventos para deixar a programação similar às placas FXO. Isso só é possível se utilizar as threads de controle (**dg_ConfigE1Thread**).

Acionamento das detecções nas placas FXO (VB0408PCI)

Na VoicerLib 4, a supervisão de linha foi implementada através de threads de controle que são criadas através do método **CreateCallProgress**. A thread de Call Progress é responsável

pelo acompanhamento de chamadas, ou seja, a supervisão de um tom genérico, de tons de Linha, de Chamar (ringback), de Ocupado, de Fax e o atendimento pelo assinante chamado. A função **CreateCallProgress** cria a thread de controle e em seguida *dorme* portanto pode ser criada no início da aplicação sem prejuízo de processamento, devendo ser chamada para cada porta específica.

Todos os parâmetros de configuração da thread de Call Progress ficam armazenados em um arquivo texto de configuração localizado na pasta firmware (Linux) ou \Arquivos de Programas\VoicerLib4 (Windows). O arquivo padrão é o **cp_default.cfg** porém o desenvolvedor poderá criar outros de acordo com suas necessidades, já que o nome do arquivo a ser utilizado é passado na função **CreateCallProgress**. Também é possível configurar o CallProgress sem a utilização do arquivo. Todos os parâmetros podem ser configurados através da função **ConfigCallProgress** pois cada item do arquivo de configuração tem sua constante correspondente que pode ser utilizada no **ConfigCallProgress**. Veja todas estas opções no Guia de Referência, na função **ConfigCallProgress**.

Depois da criação da thread de controle, é necessário chamar a função **EnableCallProgress** para efetivamente monitorar a linha. Como novidade, o acompanhamento de chamadas foi dividido em 4 opções básicas para agilizar sua utilização e economizar recursos de hardware:

- 1 - CP_ENABLE_GENERIC_TONE** - Desenvolvida para detectar a presença de um tom qualquer pré-configurado ou o atendimento de chamada, se esta facilidade estiver habilitada na VoicerLib pelo método **EnableAnswerDetection**.
- 2 - CP_ENABLE_LINE_TONE_OR_BUSY** - Desenvolvida para a detecção rápida de tom de discar (tom de linha) ou ocupado antes do início de uma discagem ou após um Flash.
- 3 - CP_ENABLE_BUSY_OR_FAX** - Desenvolvida para a

detecção rápida de tom de ocupado ou sinal de FAX após o atendimento de uma chamada.

4 - CP_ENABLE_ALL - Desenvolvida para a detecção de tons de discar, de chamada, de ocupado, de FAX e atendimento entre uma discagem e o atendimento pelo assinante chamado. A princípio, esta opção atende qualquer situação, mas devido às restrições das várias cadências dos tons, a discriminação de um determinado tom pode demorar mais tempo que numa opção específica como a **LINE_TONE_OR_BUSY**. Isso significa que o desenvolvedor poderá simplificar sua aplicação sempre habilitando o **callprogress** utilizando esta opção, mas terá uma detecção mais rápida e eficiente se utilizar a opção específica para cada situação.

Então, como sugestão, o desenvolvedor pode utilizar a seguinte regra em uma discagem com supervisão:

- Com a porta desligada, ao iniciar uma ligação (**PickUp**), chama-se o **EnableCallProgress** passando o valor **CP_ENABLE_LINE_TONE_OR_BUSY**
- Ao detectar o tom de linha, efetua a discagem e após a discagem, chama-se novamente o **EnableCallProgress** passando o valor **CP_ENABLE_ALL**.
- Após a discagem, deve-se habilitar a detecção de atendimento também, utilizando-se o método **EnableAnswerDetection**.
- Logo após o atendimento, deve-se desabilitar sua detecção chamando o método **DisableAnswerDetection** e chamar o **EnableCallProgress** com o valor **CP_ENABLE_BUSY_OR_FAX**, já que após o atendimento só é necessário esperar o tom de ocupado e o fax.

O **DisableCallProgress** deve ser chamado para desabilitar qualquer tipo de supervisão e o método **DestroyCallProgress** deve ser chamado no término da aplicação para evitar a perda de recursos de memória (*memory leak*).

Conforme já foi dito, todos os parâmetros de configuração da thread de Call Progress ficam armazenados em um arquivo texto de configuração **cp_default.cfg** (padrão) que contém diversos comentários explicando cada parâmetro. Por isso ele é reproduzido abaixo:

```
[CallProgress]
;-----
; AnswerSensitivity refere-se a quantos "audios" deverão
; ser recebidos na sequencia para considerar atendimento
;-----
AnswerSensitivity=1

;-----
; AnswerSensitivityTime é a duração mínima de um áudio
; para se considerar atendimento. Para diminuir a
sensibilidade
; aumente o valor. O padrao é 200ms
;-----
AnswerSensitivityTime=200

;-----
; GenericToneTimeout determina o tempo máximo que o sistema
; esperará para reconhecer o tom genérico. Caso não venha
; neste tempo, gerará evento de timeout.
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_GENERIC_TONE
;-----
GenericToneTimeout=15000

;-----
; GenericToneTime determina o tempo *mínimo* para que
; o sistema reconheça o audio como tom genérico
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_GENERIC_TONE
;-----
GenericToneTime=500

;-----
; LineToneTimeout determina o tempo *máximo* que o sistema
; esperará para reconhecer o tom de linha. Caso não venha
; neste tempo, gerará evento de timeout.
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_LINETONE_OR_BUSY
;-----
LineToneTimeout=15000

;-----
; LineToneTime determina o tempo *mínimo* para que
```

```
; o sistema reconheça o audio como tom de linha
; Valor em milisegundos
; *** PRECISA SER MAIOR QUE BusyMaxTime ***
; Opções de Call Progress:
; CP_ENABLE_LINETONE_OR_BUSY
;-----
LineToneTime=1000
;-----
; FaxToneTimeout determina o tempo *máximo* que o sistema
; esperará para reconhecer o tom de fax. Caso não venha
; neste tempo, gerará evento de timeout.
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_BUSY_OR_FAX
;-----
FaxToneTimeout=15000
;-----
; FaxToneTime determina o tempo *mínimo* para que
; o sistema reconheça o audio como tom de linha
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_BUSY_OR_FAX e CP_ENABLE_ALL
;-----
FaxToneTime=500
;-----
; CallProgressTimeout é o tempo de espera após a discagem
; para se considerar que houve um atendimento por timeout
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
CallProgressTimeout=15000
;-----
; BusyMinTime determina o tempo mínimo do tom de ocupado
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_LINETONE_OR_BUSY e CP_ENABLE_BUSY_OR_FAX e
; CP_ENABLE_ALL
;-----
BusyMinTime=100
;-----
; BusyMaxTime determina o tempo máximo do tom de ocupado
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_LINETONE_OR_BUSY e CP_ENABLE_BUSY_OR_FAX e
; CP_ENABLE_ALL
;-----
BusyMaxTime=500
;-----
; CallingMinToneTime determina o tempo mínimo para se
```

```
; considerar um tom de chamada (apos a discagem). Será
; testado como intervalo junto com CallingMaxToneTime
; Valor em milisegundos
; *** PRECISA SER MAIOR QUE BusyMaxTime ***
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
CallingMinToneTime=600
;-----
; CallingMaxToneTime determina o tempo maximo para se
; considerar um tom de chamada (apos a discagem). Será
; testado como intervalo junto com CallingMinToneTime
; Valor em milisegundos
; No CP_ENABLE_ALL este tempo + 500ms também é utilizado
; para medir o tom de discagem
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
CallingMaxToneTime=2000
;-----
; CallingMinSilTime o tempo minimo para se
; considerar silencio. Será testado como intervalo
; junto com CallingMaxSilTime
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
CallingMinSilTime=700
;-----
; CallingMaxSilTime determina o tempo maximo para se
; considerar silencio. Será
; testado como intervalo junto com CallingMinSilTime
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
CallingMaxSilTime=5000
;-----
; ToneInterruptionMinTime determina o tempo minimo do
; intervalo entre cada tom de linha. O detecção do tom
; de chamando depende tambem dos tempos do intervalo entre
; eles.
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
ToneInterruptionMinTime=20
;-----
; ToneInterruptionMaxTime determina o tempo minimo do
; intervalo entre cada tom de linha. O detecção do tom
```

```

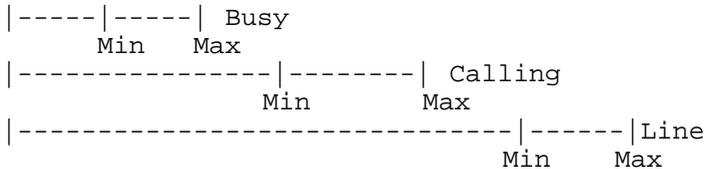
; de chamando depende tambem dos tempos do intervalo entre
; eles.
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
ToneInterruptionMaxTime=380
;-----
; LineToneMinTime determina o tempo minimo para se
; considerar o tom de linha
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
LineToneMinTime=1000
;-----
; LineToneMaxTime determina o tempo maximo para se
; considerar o tom de linha
; Valor em milisegundos
; Opções de Call Progress:
; CP_ENABLE_ALL
;-----
LineToneMaxTime=2500
;-----
; Configuracao das frequencias para cada tipo de deteccao
; Aqui deverá ser informada o indice de configuracao da
; voicerlib:
; CP_SILENCE  usar valor -> (0x20) 32
; CP_AUDIO   usar valor -> (0x21) 33
; CP_TONE1   usar valor -> (0x22) 34 (425Hz)
; CP_TONE2   usar valor -> (0x23) 35 (1100Hz)
; CP_TONE3   usar valor -> (0x24) 36 (2100Hz)
; custom comands
; CP_TONE4   usar valor -> (0x25) 37
; CP_TONE5   usar valor -> (0x26) 38
; CP_TONE6   usar valor -> (0x27) 39
; CP_TONE7   usar valor -> (0x28) 40
; CP_TONE8   usar valor -> (0x29) 41
;-----
Audio=0x21                ;default CP_AUDIO - nao mudar!!!
Silence=0x20              ;default CP_SILENCE - nao mudar!!!
LineToneFreq=0x22         ;default refere-se a CP_TONE1
CallingToneFreq=0x22      ;default refere-se a CP_TONE1
BusyToneFreq=0x22         ;default refere-se a CP_TONE1
Fax1ToneFreq=0x23         ;default refere-se a CP_TONE2
Fax2ToneFreq=0x24         ;default refere-se a CP_TONE3
GenericToneFreq=0x25      ;default refere-se a CP_TONE5

```

IMPORTANTE: No caso da opção CP_ENABLE_ALL, como no

Brasil todos os tons tem a mesma frequência (425Hz) a distinção de Chamando, Ocupado e tom de Discar (Linha) é feita pela análise de cadência, ou seja, duração de tons e silêncio, portanto deve-se tomar cuidado com os tempos acima respeitando-se a seguinte relação:

(BusyMaxTime < CallingMinToneTime) e
 (CallingMaxToneTime < LineToneMinTime)



Acionamento das detecções nas placas E1

Conforme foi dito anteriormente, nas placas E1 não é necessário habilitar as detecções de tom de linha, ocupado ou atendimento pois estas respostas vêm automaticamente pelo R2D. Já a detecção de fax deve ser habilitada explicitamente através da chamada à função **dg_SetDetectionType (SetDetectionType)** para as frequências 1100Hz e 2100Hz. Estas frequências já são previamente definidas através das constantes DETECT_TONE2 e DETECT_TONE3, as quais deverão ser passadas no segundo parâmetro da função **dg_SetDetectionType(SetDetectionType)**. Consulte o *Guia de Referência* para maiores detalhes.

Reconhecimento dos sinais de supervisão

- **Sinal de Ocupado:**

Ao ser detectado o sinal de ocupado, o evento **EV_BUSY (OnBusyDetected)** é gerado, podendo ser tratado pelo

usuário da maneira como quiser.

EXCLUSIVO Placas FXO: O evento é gerado sempre que for detectado o ocupado, portanto, caso o sinal de ocupado permaneça por muito tempo sem desligar, o programa gerará várias vezes o evento. Para evitar isso, o programador deve desabilitar a detecção com a função

dg_DisableCallProgress (dg_DisableCallProgress), após o primeiro ocupado. Outro efeito que pode ocorrer muito raramente é o sinal de ocupado ser detectado durante a conversação. Isto acontece com certos tons e cadências de voz. Para prever esta situação, recomendamos ao programador sempre esperar pelo menos dois sinais de ocupado antes de tomar alguma atitude. Isto reduz muito os casos de interpretação errada pois é muito difícil uma voz gerar dois tons de ocupado na mesma cadência do tom real.

EXCLUSIVO VB6060PCI (E1): O evento de ocupado é gerado no caso de canal bloqueado, término de ligação, etc. Como não se trata de um tom intermitente e sim de dado R2D, **o evento OnBusyDetected só ocorrerá uma vez**. Por característica do protocolo R2D, é possível detectar o desligamento independente de quem tenha gerado a ligação.

- **Detecção de Fax**

A detecção de fax é útil principalmente em aplicações de atendimento automático. É possível fazer uma rotina, por exemplo, que ao perceber um sinal de fax, transfere para o ramal do aparelho de fax. O evento que é gerado neste caso é o **EV_FAX (OnFaxDetected)**.

- **Detecção de Atendimento**

A VoicerLib permite ao programador detectar quando uma ligação foi atendida do outro lado da linha. O evento **EV_ANSWERDETECTED (OnAnswerDetected)** é gerado sempre quando este sinal for detectado. Lembre-se que, nas

placas FXO, é necessário desabilitar a detecção de atendimento logo após a ocorrência da primeira para evitar de se detectar falsos atendimentos durante a conversação.

- **Detecção de sinal de discagem**

Em linhas analógicas, o sinal de discagem é aquele tom contínuo que se houve ao tirar o fone do gancho. Em alguns casos, este sinal pode demorar alguns segundos, principalmente em centrais congestionadas. Este evento é controlado pelas funções **dg_EnableCallProgress (EnableCallProgress)** e **dg_DisableCallProgress (DisableCallProgress)**. O evento **EV_DIALTONE (OnDialToneDetected)** é gerado sempre quando este sinal for detectado.

Já em linhas digitais (E1), não existe o tom de discagem propriamente dito. No momento de ocupar o canal, já é enviado imediatamente se o canal estiver bloqueado, por exemplo. Quando o canal da placa recebe a confirmação de ocupação, significa que está pronto para discar. A VoicerLib gera o evento **OnDialToneDetected** quando esta confirmação é recebida.

Em uma situação prática, é mais correto esperar o tom de discagem para então utilizar o método Dial para discar e não atender e discar em seguida.

- **Detecção de sinal de chamada**

Ao discar para um determinado número, sempre ouvimos o sinal de chamada, que indica que o telefone está tocando do "outro lado" da linha. Este evento também é controlado pelas funções **dg_EnableCallProgress (EnableCallProgress)** e **dg_DisableCallProgress (DisableCallProgress)**. Com este sinal podemos contar quantos toques são dados até a ligação ser atendida ou ainda detectar que a ligação não foi completada (no caso de o sinal não ser detectado).

O evento **EV_CALLING (OnCalling)** ocorre sempre quando o sinal for detectado, o que faz com que ele seja chamado várias vezes até a ligação ser atendida.

Nas placas E1 com a *thread* de controle habilitada, o evento OnCalling é gerado independente de se chamar Enable/DisableCallProgress e independe da chegada do tom audível pois funciona através de um *timer* interno da VoicerLib. Para a aplicação final, essa diferença de funcionamento não altera nenhum procedimento. O tom de controle de chamada (*ringback*) ou ocupado é gerado localmente pela placa para que o usuário tenha conhecimento do andamento da ligação.

4.9 Detecção de Silêncio

A VoicerLib 4 permite agora fazer detecção de silêncio através de um comando específico para isso. Em muitas situações, perceber a existência de silêncio é muito importante. Um exemplo típico de aplicação é a gravação de mensagens de correio de voz onde o desenvolvedor queira interromper a gravação caso a ligação fique em silêncio por *N* segundos. Outro exemplo é na gravação de conversas telefônicas com a linha em paralelo pois permite que a gravação seja pausada ou terminada caso haja silêncio por um tempo pré-determinado.

O método que habilita esta funcionalidade é o **EnableSilenceDetection** onde deve ser passado como parâmetro a porta, o tempo mínimo para se considerar silêncio (*silence_time*) e o tempo mínimo para se considerar áudio (*audio_time*). Estes dois últimos parâmetros devem ser informados em milissegundos (1000 = 1s).

O parâmetro **audio_time** permite ainda receber o valor zero, indicando que ao primeiro sinal de áudio, a aplicação será avisada. Dependendo da aplicação e do ambiente de produção este valor pode ser aumentado para um valor maior pois assim a VoicerLib desprezará sinais de áudio que tiverem duração menor que a especificada. Esse comportamento equivale a uma diminuição da sensibilidade da detecção de áudio. Se não houver situações de ruído na linha, etc, recomenda-se utilizar o valor zero neste parâmetro.

O evento **OnSilenceDetected** (EV_SILENCE) será gerado sempre que houver uma mudança no estado, ou seja, quando for detectado o silêncio, o evento será gerado uma vez, informando no parâmetro **SignalCode** o valor DG_SILENCE_DETECTED (1). Quando receber um sinal de áudio, o evento será gerado novamente com o parâmetro **SignalCode** recebendo o valor DG_AUDIO_DETECTED (0). Depois disso, o evento só será gerado de novo se for detectado um silêncio e assim por diante.

Para desabilitar esta detecção, é necessário chamar o método **dg_DisableSilenceDetection**.

4.10 Detecção de Tons

Uma das maiores evoluções da VoicerLib 4 em relação à versão 2.x é nas detecções de tons. Agora existem muito mais opções de detecções, dando ao desenvolvedor grande liberdade de trabalho mesmo em situações incomuns.

Por padrão, a VoicerLib permite a detecção de até oito tipos de

tons, utilizados em supervisão de linha, etc. Destes 8 tipos de tons, temos pré-configurados três:

- DETECT_TONE1 - Tom puro 425Hz, o padrão brasileiro para tons de supervisão de linha
- DETECT_TONE2 - Tom de 1100Hz, utilizado para detectar FAX
- DETECT_TONE3 - Tom de 2100Hz, utilizado para detectar FAX

Estes tons puros são suficientes para a supervisão de linha padrão e detecção de fax na maioria dos casos. O tópico anterior "**Supervisão de Linha**" aborda detalhadamente como utiliza-lo e a chamada a estas funções está embutido nas chamadas de *callprogress*, portanto, uma configuração manual não é necessária a não ser em casos especiais.

O método que habilita/desabilita as detecções é o **SetDetectionType (dg_SetDetectionType)** onde devem ser passados como parâmetros a porta, o comando (DETECT_TONE1, etc...) e o flag que habilita ou desabilita a detecção (DG_ENABLE/DG_DISABLE). Consulte o guia de referência para ver todas as opções.

A chamada deste método tem efeito cumulativo, ou seja, fazendo uma chamada para habilitar TONE3 por exemplo, e outra chamada para habilitar TONE1 ocasionará a detecção dos dois tipos.

Esta mesma função habilita a detecção de dígitos que será abordado no tópico seguinte.

Mudando as Configurações Padrão

O desenvolvedor poderá reconfigurar todos os tipos de tons (TONE1 a TONE8) a serem detectados, através do método **SetCardDetections**, passando como parâmetros a placa, o tom a ser modificado e o par de frequências que passará a ser

monitorado. No Guia de Referência das funções e métodos são apresentadas todas as opções.



ATENÇÃO: É importante ressaltar que a alteração das frequências a serem detectadas só podem ser feitas na placa inteira e não por porta individualmente. Isso não constitui uma limitação já que normalmente uma alteração deste tipo afeta todo o sistema e não portas específicas.

4.11 Detecção de Dígitos

A VoicerLib permite detectar dígitos tanto em tons multifrequenciais como em pulso nativamente.

A detecção de tons multifrequenciais é habilitada ou desabilitada através do método **SetDetectionType** onde devem ser passados como parâmetros a porta, o comando (DETECT_DTMF etc...) e o flag que habilita ou desabilita a detecção (DG_ENABLE/DG_DISABLE). Consulte o guia de referência para ver todas as opções.

O segundo parâmetro, o comando pode assumir as seguintes opções:

- DETECT_DTMF - Detecção do DTMF - o mais comum para aplicações com interação com o usuário
- DETECT_MFF e DETECT_MFT - São os sinais multifrequenciais comuns à sinalização R2D
- DETECT_MF - Sinal multifrequencial customizável através do método **SetCustomMF**

A detecção de pulso é habilitada pelo método **EnablePulseDetection** e desabilitada pelo método **DisablePulseDetection**. No método **EnablePulseDetection** é passado a sensibilidade de detecção, sendo o padrão zero. Esta sensibilidade pode ser alterada caso se perceba uma dificuldade ou facilidade maior nas detecções de pulso.

O funcionamento das duas detecções é praticamente o mesmo exceto pelo fato que a detecção de tom pode ser feita durante a reprodução de uma mensagem e a de pulso somente no silêncio após a mensagem.



Habilitando o cancelamento de eco, é possível ter boa performance na detecção de pulsos sobre a mensagem.

A VoicerLib permite tratar o reconhecimento de dígitos de duas maneiras diferentes.

A primeira é através do evento **OnDigitDetected (EV_DTMF)**, que é gerado sempre que um dígito qualquer for detectado pela placa. O dígito detectado será passado através do parâmetro **Digit** do evento. Este parâmetro conterá o código ASCII do dígito.

Caso o programador implemente algum tratamento neste evento, deve tomar cuidado de saber o momento em que ele ocorreu, pois a detecção poderá ser gerada inclusive pelos dígitos gerados pela própria placa. O mais aconselhável é só habilitar as detecções nos momentos que ela seja necessária, para evitar situações de *talk-off*. É possível otimizar a detecção utilizando a função **SetTwist** (consulte o Guia de Referência para maiores detalhes).



TALK-OFF: A voz humana, em uma conversa normal, pode conter a mesma frequência dos dígitos detectados pela placa, portanto, quando o operador ou o interlocutor falar, algum dígito pode ser detectado e se o tratamento no OnDigitDetected não for adequado, o sistema pode interpretar erroneamente os sinais recebidos.

Uma aplicação típica que se recomenda a utilização do evento OnDigitDetected é na identificação de chamadas (BINA) nas placas FXO.

Exemplo:

Neste exemplo, o dígito é detectado o tempo inteiro e mostrado na tela.

```
Private Sub VoicerLibX1_OnDigitDetected(Port As Integer, Digit As Integer)
    lblStatus.Caption = "Detectou Dígito " + Chr$(voicerlibx1.ReadDigits(Port))
End Sub
```

A segunda maneira de detectar dígitos é através do método **GetDigits**.

O método **GetDigits** inicia a monitoração de dígitos, sendo que é possível determinar o número máximo de dígitos, se esperará um dígito terminador, e tempos máximos para recepção destes dígitos.

Após o início da monitoração, o evento **OnDigitsReceived** pode ocorrer a qualquer momento. Este evento ocorre quando uma das condições impostas pelo método **GetDigits** for satisfeita. O método **ReadDigits** permite ler o conteúdo do buffer de dígitos

da porta.

Durante a monitoração é possível cancelar o andamento do **GetDigits** através do método **CancelGetDigits**. Se esse método for chamado, o **GetDigits** será cancelado mas o evento **OnDigitsReceived** *não* será gerado e o buffer de dígitos será apagado. O método **CancelGetDigits** deverá ser chamado principalmente em situações onde a ligação foi terminada durante o **GetDigits**.

Uma mudança importante em relação às versões anteriores da VoicerLib é que agora o buffer de dígitos deve sempre ser apagado explicitamente pelo programa, através do método **ClearDigits**.

Exemplo:

No exemplo abaixo o GetDigits inicia esperando até 5 dígitos, ou até receber o terminador # por no máximo 10 segundos de espera total ou 5 segundos de intervalo entre cada dígito:

```
Private Sub Espera Digito()  
  
    VoicerLibX1.GetDigits Porta,5,"#",10000,5000  
End Sub  
  
'No evento OnDigitsReceived é que será tratado os  
'digitos  
'recebidos, ou verificado timeout  
Private Sub VoicerLibX1_OnDigitsReceived(Port As  
Integer,  
Status As VoicerLib.TxWaitDigit)  
  
    Select Case Status  
        Case edMaxDigits:  
            'Alcançou o máximo de digitos, disca  
para o ramal  
            '.....  
        Case edTermDigit:  
            'Recebeu um número com # no fim
```

```
        '.....
    Case edDigitTimeOut:
        'Time out global de 10 segundos
        '.....
    Case edInterDigitTimeOut:
        'Deu time-out entre dois dígitos
        '.....

    End Select
End Sub
```

O evento **OnDigitsReceived** também pode ter a variável **Status** com valor **edDigitOverMessage**. Neste caso, o evento terá sido gerado pela detecção de um dígito durante a reprodução de uma mensagem a partir dos métodos **Playxxx**.

4.12 Identificação de Chamadas

A indentificação de chamadas (BINA) está mais facilmente disponível através dos métodos **IdleXXX** nas placas FXO e da **thread** de controle nas placas E1 (**CreateE1Thread**). Consulte os referidos tópicos para maiores detalhes.

Entretanto, o desenvolvedor também pode desenvolver sua própria rotina de detecção de identificador de chamadas, normalmente fazendo uso do método **SetDetectionType** para configurar o tipo de sinalização e manipulando estes dígitos através do evento **OnDigitDetected (EV_DTMF)**. Neste caso, toda a montagem da string com o número identificado é manual.

4.13 Portas Virtuais

Quando a VoicerLib é inicializada, todas as placas presentes no computador são analisadas e o número da porta é assumido na ordem em que as placas estão posicionadas no barramento PCI. Por exemplo, se houverem 2 placas FXO VB0408PCI de 8 canais cada e uma placa E1 VB6060PCI de 60 canais, as portas de 1 a 16 serão atribuídas às duas placas FXO e as portas 17 a 77 serão atribuídas à placa E1.



Aparentemente não existe padrão entre os fabricantes de motherboard quanto à ordem das placas no barramento. Também os sistemas operacionais podem reconhecer as placas de maneiras diferentes no mesmo hardware. Entretanto, não existem alteração nessa ordem depois de tudo instalado corretamente.

Essa alocação automática de portas é adequada à maioria das aplicações, porém existem situações onde é interessante o desenvolvedor poder determinar o número da porta de cada canal físico da placa. A gravação em paralelo em placas VB6060PCI é um exemplo típico pois somente os primeiros 30 canais de cada placa são utilizados (na placa de 60 canais) o que faz com que a aplicação tenha que gerenciar os canais de 1-30, 61-90, 121-150 e assim por diante. Com a utilização das funções de canais virtuais, o desenvolvedor pode, no início da aplicação associar os canais de maneira a ficar com uma numeração de portas continua (de 1 a 90, por exemplo).

Associando um canal físico à uma porta virtual

O método **DefinePortResource** (dg_DefinePortResource) permite atribuir uma numeração qualquer a um canal físico de determinada placa, tendo a seguinte sintaxe:

```
DefinePortResource(PortaVirtual, Placa,  
Canal)
```

Onde **PortaVirtual** é o número lógico a ser utilizado, **Placa** é o número da placa (de 1 a n , na ordem do barramento PCI) e **Canal** é o canal físico de **Placa** (ex.: uma placa FXO de 8 canais tem **Canal** de 1 a 8). **PortaVirtual** deverá respeitar o número máximo de portas reconhecida pelo sistema, ou seja, não poderá ter valor maior do número máximo de portas, valor esse que pode ser obtido através do método **GetPortsCount**.

A única verificação que o método **DefinePortResource** faz é com relação aos valores de **Placa** e **Canal**. Não existe nenhum *feedback* se determinada porta virtual já foi alocada ou não. Este controle deverá ser feito pela aplicação.

Após fazer a associação, **PortaVirtual** passará a ser utilizada pela VoicerLib em todos os métodos e eventos para referenciar o canal físico da placa selecionada.

Restaurando a configuração padrão de inicialização

A qualquer momento é possível voltar a numeração das portas para o reconhecimento automático, bastando para isso chamar o método **ResetPortResource**. Este método não exige nenhum parâmetro e fará com que a VoicerLib efetue o reconhecimento de portas feito na inicialização.

Verificando a configuração de portas virtuais

Pode ser útil ao desenvolvedor saber como está a configuração das portas virtuais. Para isso pode-se fazer uso dos métodos **GetPortsCount**, **GetCardNumber** e **GetRelativeChannelNumber**. O seguinte código foi extraído do exemplo VirtualPorts em VisualBasic e serve para exibir a configuração atual das portas do sistema:

```
'funcao que retorna string com identificação da
```

```
placa
Private Function PegaNomePlaca(nCard As
Integer) As String
    Select Case voicerlib1.GetCardType(nCard)
        Case VBE13060PCI
            PegaNomePlaca = "E1 30/60 rev.1"
        Case VB0408PCI
            PegaNomePlaca = "VoicerBox 0408"
        Case VBE13060PCI_R
            PegaNomePlaca = "E1 30/60 rev.2"
        Case VBE16060PCI
            PegaNomePlaca = "E1 30/60 slim"

    End Select
End Function

Private Sub cmdAnalisa_Click()
    Dim i As Integer
    Dim sMsg As String

    Dim card As Integer, card_ch As Integer

    MostraStatus 0, "Numero total de portas = "
& voicerlib1.GetPortsCount()
    For i = 1 To voicerlib1.GetPortsCount()
        card = voicerlib1.GetCardNumber(i)
        sMsg = PegaNomePlaca(card)
        MostraStatus i, "Placa " & card & "
tipo: " & sMsg & " - ch da placa: " &
voicerlib1.GetRelativeChannelNumber(i)
    Next i

End Sub
```

4.14 Gravando uma Conversa

A VoicerLib permite gravar conversas telefônicas tanto com o ramal/linha conectada diretamente à porta da placa como em paralelo. A gravação em paralelo tem um tópico à parte, porém os conceitos básicos são apresentados aqui.

É possível efetuar a gravação em diversos formatos de arquivos, com nível de compactação diferentes entre si. Observe a tabela abaixo:

Formato	Taxa (Kbits/s)	MB/h (*)	Kbytes/h (*)
ffWavePCM	128	57.6	57600
ffWaveULaw	64	28.8	28800
ffWaveALaw	64	28.8	28800
ffGsm610	13.2	5.9	5940

(*) **MB/h e KBytes/h** indica a taxa de ocupação do arquivo gerado pela gravação no formato especificado.

O método **SetRecordFormat (dg_SetRecordFormat)** indica para a VoicerLib qual o formato que determinada porta deverá assumir.



O que determina o formato de gravação é o **SetRecordFormat** e não a extensão do arquivo, portanto, simplesmente associar a extensão **.gsm** ao arquivo não determinará o seu formato!

Para aplicações com extensivo uso de gravação recomenda-se o formato GSM que oferece uma excelente relação entre tamanho e qualidade de áudio. Diferentemente da VoicerLib2, a codificação/decodificação do GSM é feita diretamente pelo processador (DSP) da placa sendo portanto o formato de áudio mais eficiente pois o computador não gasta tempo de CPU nos processos de codificação/decodificação e o I/O no barramento

PCI também é muito menor que nos formatos Wave.

Existe também o método **SetGSMMode** que indica a VoicerLib se deverá ser utilizado o GSM compatível com o Asterisk(c) (GSM_RAW) ou o GSM padrão da Digivoice (GSM_DIGIVOICE). A diferença entre os dois formatos consiste apenas na existência de um cabeçalho no padrão Digivoice. A codificação em si é a mesma e por isso não foi criado um novo formato de gravação. Este método afeta todas as portas de todas as placas e o padrão assumido atualmente é o GSM_RAW que não contém cabeçalho no arquivo.

Para efetuar a gravação de canais na voicerlib, é necessário habilitar o envio de amostras da placa através da função **EnableInputBuffer (dg_EnableInputBuffer)**. Esta função espera dois parâmetros: a porta que gerará as amostras para a aplicação e flag que habilita ou não o AGC (Controle Automático de Ganho). Após chamar o **EnableInputBuffer**, o método **RecordFile (dg_RecordFile)** efetivamente iniciará a gravação.



É muito importante verificar os valores de retorno das funções **EnableInputBuffer** e **RecordFile**. Dependendo da carga de processamento, é possível que ao chamar o método **RecordFile** a thread de controle iniciada pelo **EnableInputBuffer** ainda não esteja pronta. Neste caso a melhor alternativa é que, em caso de erro, o desenvolvedor chame a função **RecordFile** novamente.

O método **RecordFile** também permite determinar se algum dígito será utilizado como finalizador de gravação. Esta característica permite uma implementação do tipo: *"Grave seu recado e ao final digite # se quiser falar com a telefonista"*.

O dígito detectado (se usado) é armazenado e deve ser recuperado utilizando-se a função **dg_ReadDigits**

(**ReadDigits**), e pode ser tratado posteriormente.

Dependendo do fluxo de operação do sistema desenvolvido, é necessário que se vá acumulando os dígitos detectados. Este é o comportamento padrão da VoicerLib. Consulte a função **dg_SetAutoClearDigits** no Guia de Referência para maiores detalhes caso se prefira apagar os dígitos automaticamente (não recomendado).

Sempre quando for iniciada a gravação o evento **EV_RECORDSTART (OnRecordStart)** é gerado, permitindo tratamento nesta situação. É possível saber se determinado canal está gravando chamando a função **dg_IsRecording (IsRecording)**.

Ao término da gravação o evento **EV_RECORDSTOP (OnRecordStop)** é gerado automaticamente, inclusive informando qual o *status* da interrupção da gravação.

Este *status* pode ser uma ação direta do operador (com a chamada da função **dg_StopRecordFile (StopRecordFile)**), um dígito recebido ou ainda um erro qualquer (disco cheio, por exemplo).

Para desativar a thread de controle de gravação (iniciada pelo **EnableInputBuffer**) deve ser chamado o método **DisableInputBuffer**, passando como parâmetro a porta. O melhor lugar para desabilitar a thread de controle de gravação é no evento **EV_RECORDSTOP (OnRecordStop)** pois ali sabemos que a gravação foi efetivamente finalizada e a thread pode ser destruída.



A thread de controle de gravação é o processo que consome processamento pois com ela ocorre um intenso I/O entre o device driver e o hardware. Por isso é recomendado que ela seja sempre habilitada e desabilitada quando do início e fim das gravações.

Recomenda-se chamar o EnableInputBuffer e DisableInputBuffer no começo e no fim da aplicação, respectivamente, pois o processo de criação de threads pode ser um pouco lento em relação à algumas situações onde o começo e fim de gravação é muito rápido (grande volume de ligações) e fazer com que a repetida chamada do EnableInputBuffer possa retornar erro devido ao fato do DisableInputBuffer anterior ainda não ter fechado o arquivo.

Sempre quando o EnableInputBuffer for chamado uma única vez no início da aplicação, o método **PauseInputBuffer** deverá ser chamado para evitar o consumo desnecessário de processamento em momentos onde não for necessária sua utilização. Este método é chamado com o parâmetro *porta* e com um segundo parâmetro que indica para colocar a thread em pausa (DG_PAUSE) ou sair de pausa (DG_RELEASE);

Exemplo (usando o ActiveX):

Após falar: *"Grave seu recado e ao final digite # se quiser falar com a telefonista"*, o sistema iniciará a gravação. Se for detectado o #, disará para o ramal 200.

```
Private Sub DeixarRecado()  
    '.... código para falar a mensagem ....  
    'Inicia gravação  
    VoicerLibX1.EnableInputBuffer 1,DG_ENABLE  
    VoicerLibX1.RecordFile(1,"c:\recado.sig","#")  
    'Lembre que o programa segue o fluxo normal  
    após iniciada
```

```
'a gravação

End Sub

'Rotina de tratamento do click de um botão para o
caso de o
'operador parar a gravação manualmente
Private Sub cmdParaGravar_Click()
    VoicerLibX1.StopRecordFile(1)
End Sub

'Evento que ocorre quando a gravação é finalizada
'É necessário analisar o motivo
Private Sub VoicerLibX1_OnRecordStop(ByVal Port
as Integer,
                                Status As
VoicerLib.TxStopStatus)
Select Case Status
Case ssStopped:
    'Gravação interrompida pelo operador
    VoicerLibX1.HangUp 1
Case ssDigitReceived:
    'Recebeu o dígito finalizador
    'Transfere para a telefonista
    if Digits = "#" then
        VoicerLibX1.Flash 1,600,1000
        VoicerLibX1.Dial 1,"200",1000
        VoicerLibX1.HangUp 1
    end if
    End Select
    VoicerLibX1.DisableInputBuffer Port
End sub
```

Durante a gravação é possível monitorar o andamento da gravação através do evento **EV_RECORDING (OnRecording)**. No ActiveX, o parâmetro **ElapsedTime** indica a quantidade de segundos decorridos até aquele momento. O mesmo ocorre na variável `context_data` da API.

A gravação pelo monofone conectado nas placas VoicerPhone (1 canal) ou através do cabo especial da VoicerBox PCI/4 pode ser iniciado através do evento **EV_LINEReady (OnLineReady)** e finalizada através do evento **EV_LINEOff (OnLineOff)**. Leia

detalhes deste método de funcionamento no tópico seguinte:
Gravação em Paralelo.

A função **dg_RecordPause (RecordPause)**, que permite interromper temporariamente uma gravação e em seguida continuar no mesmo arquivo. Neste método é necessário passar a porta e um flag booleano indicando se coloca em pausa (TRUE) ou retira da pausa (FALSE).

4.15 Reproduzindo Mensagens

A VoicerLib permite reproduzir qualquer mensagem gravada em formato Wave ou GSM através do método **PlayFile (dg_PlayFile)**.

O **PlayFile** detecta automaticamente o tipo de arquivo baseado na extensão. Se encontrar um arquivo com a extensão **.gsm** tentará reproduzir o arquivo usando o codec GSM. Se for encontrada a extensão **.wav**, a Voicerlib efetuará uma análise do cabeçalho para identificar o tipo de wave (Linear, Lei A ou Lei Mi). Se for utilizado uma extensão desconhecida, fica valendo o que está especificado no **SetPlayFormat**.

Além disso, é possível determinar se algum dígito será utilizado como finalizador da reprodução. Esta característica é interessante para, por exemplo, implantar um menu de opções em um sistema de auto-atendimento com a possibilidade de digitar a opção sobre a mensagem.

O dígito detectado (se usado) é armazenado internamente para cada porta da placa e pode ser recuperado através do método **ReadDigits(Porta)** podendo ser tratado posteriormente.

A VoicerLib atualmente assume que o programador sempre

deverá chamar explicitamente o método `ClearDigits` para apagar o buffer de dígitos sempre quando for necessário. O método **`SetAutoClearDigits (dg_SetAutoClearDigits)`** pode mudar este comportamento, apagando automaticamente o buffer de dígitos.

Sempre quando for iniciada a reprodução o evento **`OnPlayStart`** é gerado. Uma finalidade deste evento é permitir atualizações de interface, como por exemplo, desabilitar botões e exibir mensagens ao usuário.

Ao término da reprodução o evento **`OnPlayStop`** é gerado automaticamente, inclusive informando qual o motivo da interrupção da reprodução. Este motivo pode ser uma ação direta do operador (com a chamada do método `StopPlayFile`), um dígito recebido, o simples término da mensagem ou ainda um erro qualquer (disco cheio, por exemplo).

Se um dígito for detectado durante a reprodução (desde que o parâmetro `TermDigits` tenha sido configurado), além do evento **`OnPlayStop`**, é gerado também o evento **`OnDigitsReceived`** passando na variável `Status` o valor **`edDigitOverMessage`**. Isto facilita a criação de menus de atendimento, já que toda a consistência do que foi digitado pode ser feita apenas no evento **`OnDigitsReceived`**.

O último parâmetro do `PlayFile (Origin)` permite começar a reproduzir a mensagem a partir de um determinado ponto. Por exemplo, para reproduzir uma mensagem a partir de 10 segundos do início dela, basta executar colocar 10 neste parâmetro. Isto é útil para funções de reprodução de mensagens. Se for passado (0) ZERO como parâmetro, a mensagem será reproduzida do início. Se for passado -1 como parâmetro a mensagem começará a ser reproduzida do final menos 2 segundos.

Exemplo:

```
VoicerLibX1.PlayFile(1, "c:\boas.wav", "345", 0)
```

Neste exemplo é iniciada a reprodução de um arquivo chamado **boas.wav** e os dígitos 3,4 e 5 poderão interromper a reprodução da mensagem, caso sejam detectados.



Devido às características de implementação, a reprodução de mensagens não necessita que o buffer de amostras seja ativado antes da reprodução, como é feito na gravação. Para maiores detalhes de como enviar amostras para a placa diretamente, consulte o tópico **Streaming de Áudio**.

4.16 Conferência entre portas

A VoicerLib, na placa VoicerBox E1 30/60, oferece recursos que permitem que determinadas portas possam *conversar* com outras. Com isso é possível criar salas de conferência onde várias pessoas possam conversar entre si. Este mesmo recurso foi utilizado para a criação da *thread* de Logger, utilizada na gravação em paralelo para placas E1 explicada anteriormente.

É suportado até 30 salas de conferência com número variável de portas por sala. Para criar uma sala de conferência utiliza-se a função **CreateChatRoom** onde deve ser informado a placa que gerenciará o recurso e o número máximo de portas que esta sala poderá receber. O retorno desta função é o identificador/handle da sala e será necessário nas demais funções, portanto, sempre armazene o retorno da função de maneira a poder ser utilizado posteriormente.

Para excluir uma sala, a função utilizada é a **DestroyChatRoom**, que sempre deverá ser chamada antes de se finalizar uma aplicação, pois além de alocar recursos da

placa, também é utilizado estruturas de memória que só são liberadas quando uma sala é explicitamente destruída.

A inclusão de portas em uma sala de conferência deve sempre ser feita pela função **ChatAddPort** passando a porta e o identificador da sala. Ao incluir uma porta na sala, esta não estará ainda disponível para ouvir e falar com os outros participantes, precisando ainda habilitar a porta na sala. Essa separação entre os procedimentos de adicionar e habilitar foi utilizada para que seja possível uma porta interagir com menus, por exemplo, sem deixar a sala.

Habilitar uma porta na sala é feito com a função **ChatEnablePort** que ainda permite dizer se a porta só *falará*, só *ouvirá* ou ambos em determinada sala, através do parâmetro **Direction**. Isso dá extrema versatilidade em termos de aplicações possíveis utilizando a VoicerLib.

A função **ChatDisablePort** permite desabilitar uma porta da sala, sem removê-la definitivamente, como já foi explicado anteriormente. Em uma aplicação, isso permite, por exemplo que o participante possa, durante a conversa, teclar um dígito e entrar em um menu de opções qualquer. Depois de interagir com este menu, poderá voltar novamente à sala chamando-se a função **ChatEnablePort** novamente.

Para remover uma porta de uma sala definitivamente, a função **ChatRemovePort** deve ser utilizada. Esta situação ocorreria quando o usuário fosse embora da sala de conferência, desligando ou movendo-se para outra sala.

Por fim, um recurso adicional é a possibilidade de gravação do que se fala na sala de conferência. Isto é possível através da função **SetPortChatLog**, que habilita ou desabilita determinada porta à receber o áudio de todas as outras portas participantes de uma conferência. É importante entender que a porta escolhida para *ouvir* e gravar a conferência **não** poderá estar participando desta ou de qualquer outra conferência enquanto a

opção estiver habilitada.

4.17 Streaming de Áudio

Uma das alterações mais profundas na VoicerLib 4 é o tratamento de streaming de áudio. A utilização deste recurso é necessária quando a aplicação precisa receber da placa ou enviar para a placa amostras de áudio sem manipulação de arquivos, somente utilizando a memória. O exemplo típico é uma aplicação que "conversa" com outra pela rede (VOIP). O suporte ao Asterisk, disponível nesta versão da VoicerLib foi feito utilizando dos recursos de streaming.

Um conceito importante é que o streaming de áudio pode ser feito em qualquer formato suportado pela VoicerLib, inclusive o GSM (excelente para aplicações VOIP). Todos estes formatos são tratados pelo processador de cada placa, não utilizando recursos da CPU do micro para codificação/decodificação de formatos de áudio.

Enviando Áudio para um canal da Placa

Para indicar qual o formato utilizado para envio de amostras para uma porta da placa Digivoice deve ser utilizado o método **SetPlayFormat (dg_SetPlayFormat)**, o mesmo utilizado para a reprodução de mensagens gravadas. O método para enviar amostras para a placa é o **PlayBuffer** que deve passar como parâmetro a porta, um ponteiro de memória com o buffer de amostras a ser enviado e a quantidade de amostras que deverá ser enviada.

Quando uma aplicação enviar amostras para a placa, é importante respeitar a cadência de tempo que estas amostras devem ser enviadas para que se tenha uma reprodução contínua de áudio, sendo que no formato GSM, são enviadas 33 amostras a cada 20ms e no caso do wave (uLaw ou aLAW) são

16 amostras a cada 2ms (muito rápido). Esta necessidade de velocidade que também reforça a utilização do GSM para este tipo de aplicação.

O método **PlayBuffer** permite o envio, numa única chamada, de múltiplos destes valores(33 ou 16 amostras). Por exemplo, no formato wave é possível passar 32,64,etc amostras pois internamente essas amostras são *bufferizadas* para serem enviadas para a placa no momento certo. Porém é preciso tomar certos cuidados para não enviar uma quantidade de amostras maior do que a VoicerLib possa tratar. Nesta situação, o áudio perderá algumas amostras, aparecendo *picotado*.

O último parâmetro do **PlayBuffer** permite que seja passado um ponteiro para um inteiro que receberá, no retorno da função, a quantidade de bytes livres no buffer.

A aplicação deverá monitorar se existem amostras a serem enviadas para a placa. Caso não haja, deverá chamar o método **StopPlayBuffer** para que a placa não fique reproduzindo o último buffer de amostras, o que acarretaria num zumbido. Caso a aplicação queira enviar um ruído de conforto (confort noise), deverá gerar este ruído por conta própria e enviá-lo para a placa.

Recebendo Áudio de um canal da Placa

Este procedimento é exatamente o inverso do anterior. Aqui as amostras que o canal da placa recebe são enviadas para a aplicação para daí serem tratadas quando for necessário. Como neste caso, é a VoicerLib que sabe quando as amostras estão disponíveis, o mecanismo é um pouco mais complexo do que o uso do **PlayBuffer/StopPlayBuffer**.

Conforme já foi explicado no tópico "**Gravando uma Conversa**" a VoicerLib 4 exige que o envio de amostras da placa para a aplicação seja habilitada através do método **EnableInputBuffer** e desabilitada pelo **DisableInputBuffer**. Uma vez que as

amostras já ficam disponíveis, a VoicerLib pode decidir se vai gravar estas amostras em disco (RecordFile) ou as enviará para a aplicação final. Neste segundo caso, é necessário a utilização de uma função do tipo *Callback*.

Uma função do tipo *Callback* é uma função convencional na linguagem de programação utilizada pelo desenvolvedor. A única diferença é que quem chamará esta função é a VoicerLib e não a aplicação final. Quando houver amostras disponíveis, a VoicerLib chamará diretamente a função da aplicação disponibilizando assim as amostras de áudio. Esse procedimento é muito similar aos eventos, porém com um compromisso de tempo real que os eventos não podem garantir. A utilização de linguagem C é recomendada para aplicações deste tipo, devido à facilidade que este tipo de recurso tem nesta linguagem. Observe o exemplo:

```
void CALLBACK InputStreaming(short port, int
count, void *data)
{
    //pega as amostras em *data
}
```

Para informar a VoicerLib qual a função Callback, a função **SetAudioInputCallback** que passa como parâmetro o ponteiro da função callback. Depois de feita essa associação e de habilitado o envio de amostras (**EnableInputBuffer**) a função callback será acionada toda vez que houverem amostras disponíveis.

```
SetAudioInputCallback(InputStreaming);
```

É importante ressaltar que é de responsabilidade da aplicação ter capacidade de receber e tratar todas as amostras recebidas, criando *buffers* se forem necessários.

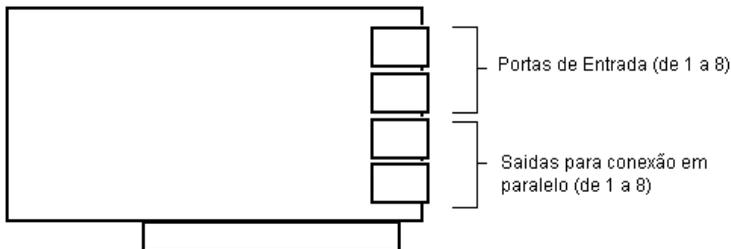
4.18 Gravação em Paralelo

A gravação em paralelo é utilizada quando as placas DigiVoice ficam apenas *monitorando* (em paralelo) a linha, sem interferir no andamento da conversação. Neste caso, nunca se utiliza os métodos PickUp ou HangUp pois não se trata de uma ligação *terminada* num canal da placa.

Para montar a estrutura de gravação em paralelo é necessário que a forma como os cabos são conectados seja respeitada para cada tipo de placa.

4.18.1 Placas FXO

Na placa VB0408PCI pode ser configurada com 4 ou 8 canais. No primeiro e no segundo conector (de cima para baixo) estão as portas de 1 a 4 e de 5 a 8, respectivamente, sendo um par de fios para cada porta. O terceiro e quarto conectores devem ser utilizados para a saída da porta para a ligação em paralelo.



Exemplo: A porta 1 entrará pelo primeiro par de fios do primeiro conector (mais acima) e sairá pelo primeiro par de fios do

terceiro conector. (Em uma gravação terminada (sem paralelo), os dois conectores inferiores não são utilizados)



A explicação detalhada da conexão física está no manual do kit integrador da placa VB0408PCI.

Ao tirar o telefone do gancho é gerado o evento **OnLineReady** e ao colocar o telefone no gancho é gerado o evento **OnLineOff**. Estes eventos deverão ser utilizados para iniciar (**RecordFile**) ou finalizar (**StopRecordFile**) a gravação. Para saber se determinada ligação é entrante ou sainte, o aplicativo deverá monitorar tons de linha, dígitos e a chegada de *Bina*. Não existe uma função ou propriedade na VoicerLib que dê essa informação pronta ao programador.

4.18.2 Placas E1

A placa **VB6060PCI** com a finalidade de gravação deverá ter obrigatoriamente 60 canais para que seja possível gravar 30 canais simultâneos.



A conexão física, com a descrição dos cabos necessários é explicada no manual do Kit Integrador, que acompanha a placa VB6060PCI

A VoicerLib também gera os eventos **EV_LINEREADY (OnLineReady)** e **EV_LINEOFF (OnLineOff)** para a placa E1, mas isso não é automático e sua configuração difere bastante das placas FXO.

É muito importante entender que a placa E1 para gravação em paralelo será sempre uma placa de 60 canais (dois E1) mas disponibilizará somente os 30 primeiros canais para gravação. No caso da utilização de mais de uma placa, o programador deverá estar atento pois todos os eventos e métodos deverão referenciar apenas os 30 primeiros canais de cada placa portanto existirão apenas os canais 1-30, 61-90, ... e assim sucessivamente. Para evitar este "salto" de canais, o desenvolvedore poderá utilizar as funções de portas virtuais, onde é possível associar um número de porta qualquer a um canal físico da placa. Veja maiores detalhes no tópico "**Portas Virtuais**".

Para controlar a gravação uma *thread* de controle deverá ser criada através do método **dg_CreateLoggerControl (CreateLoggerControl)**. Deve ser passado a porta e o tipo de protocolo utilizado. Atualmente a VoicerLib suporta apenas o RD2 MF, portanto o segundo parâmetro receberá sempre a constante **LOGGER_R2DMF (100)**.

Ao criar esta *thread* de controle, todos os canais ficarão em estado de espera, monitorando a linha. Quando uma ligação estiver sendo iniciada (entrante ou sainte) a *thread* irá monitorar toda a troca de sinalização e o evento **EV_LOGGEREVENT (OnLoggerEvent)** será gerado diversas vezes, indicando para a aplicação as diversas etapas até o início da conversação propriamente dito.

O evento **EV_LOGGEREVENT (OnLoggerEvent)** receberá o parâmetro *LoggerStatus*, que poderá assumir os seguintes valores:

- **LOGGER_FREE_WITH BILLING** - Linha de Assinante Livre

com Tarifação

- **LOGGER_BUSY** - Canal Ocupado
- **LOGGER_NUMBER_CHANGED** - Linha de assinante mudado
- **LOGGER_CONGESTION** - Congestionamento
- **LOGGER_FREE_WITHOUT BILLING** - Assinante Livre sem tarifação
- **LOGGER_FREE_RETENTION** - Assinante Livre com tarifação e colocar retenção sob ctrl do assinante chamado
- **LOGGER_LEVEL_NUMBER_AVAILABLE** - Nível ou Número Vago
- **LOGGER_B_ENDCALL** - Assinante B desligou. Quando o assinante B desliga não significa necessariamente que a ligação terminou.
- **LOGGER_B_RETURN** - Assinante B retornou a ligação. Ocorre após o **LOGGER_B_ENDCALL** caso o assinante B retorna à ligação
- **LOGGER_LINEREADY (*)** - LineReady significa o local do início efetivo da conversação
- **LOGGER_LINEOFF (*)** - LineOff significa o local do término efetivo da conversação

(*) O início e fim de conversação poderá ser tratado tanto nos eventos **EV_LINEREADY (OnLineReady)** e **EV_LINEOFF (OnLineOff)** como nos *status* **LOGGER_LINEREADY** e **LOGGER_LINEOFF**.

No início da conversação, a VoicerLib permite que seja extraído o número discado (através do método **GetE1Number**), a identificação de A (método **GetCallerID**) e também se a ligação é de entrada ou saída (método **GetLoggerCallType**). Observe o fragmento de código a seguir, que mostra todas as informações necessárias, inicia e termina a gravação:

(ActiveX)

```
Private Sub vlib_OnLoggerEvent(ByVal Port As Integer, ByVal LoggerStatus As Integer)
```

```
Dim sBina As String, sE1 As String, sTipo As
String
Dim ret As Integer

Select Case LoggerStatus
Case LOGGER_LINEREADY:
    sE1 = vlib.GetE1Number(Port)    'pega
    número do E1
    sBina = vlib.GetCallerID(Port) 'pega
    BINA
    If vlib.GetLoggerCallType(Port) =
    INCOMINGCALL Then
        sTipo = "Entrante"
    Else
        sTipo = "Sainte"
    End If
    MostraStatus Port, "Logger LineReady ("
+ sTipo + ") (Caller ID: " & sBina & ", E1
Number: " & sE1 & ")"
    'Aqui inicia a gravação
    vlib.EnableInputBuffer Port, DG_ENABLE
    ret = vlib.RecordFile(Port, App.Path &
"\grava" & Port & ".gsm", "")
    If ret <> 0 Then
        MostraStatus Port, "ERRO: Não
foi possível gravar arquivo"
    Else
        MostraStatus Port, "Iniciando
Gravação"
    End If
Case LOGGER_LINEOFF:
    'termina gravação
    ret = vlib.StopRecordFile(Port)
    vlib.DisableInputBuffer Port
    If ret <> 0 Then
        MostraStatus Port, "ERRO ao tentar
interromper gravação"
    Else
        MostraStatus Port, "Gravação
Finalizada"
    End If
End Select
End Sub
```

(API)

```
void ReceiveEvents(void *context_data)
{
    struct dg_event_data_structure *EventContext;
    char szCallerID[25];
    char szE1Number[25];
    char szFileName[255];

    short ret;

    /* Copy received Data */
    EventContext = ((struct
dg_event_data_structure*)context_data);

    switch (EventContext->command)
    {
        case EV_LOGGEREVENT:
            switch(EventContext->data)
            {
                case LOGGER_LINEREADY:
                    dg_getcallerid(EventContext-
>port,szCallerID);
                    dg_getelnumber(EventContext-
>port,szE1Number);
                    printf("LineReady (Caller ID: %s, E1
Number: %s)\n",szCallerID,szE1Number);
                    if
(dg_getloggercalltype==INCOMINGCALL)
                        printf("Ligação Entrante\n");
                    else
                        printf("Ligação Sainte\n");

                    //cria arquivo de gravacao
                    dg_setrecordformat(EventContext-
>port,ffGSM610);
                    sprintf(szFileName,"grava%d.gsm",
EventContext->port);
                    ret = dg_recordfile(EventContext-
>port,szFileName,"");
                    if (ret !=0)
                        printf("não foi possivel iniciar
gravacao");
                    break;
            }
    }
```

```
        case LOGGER_LINEOFF:
            printf("LineOff - Fim da gravacao");
            ret =
dg_stoprecordfile(EventContext->Port);
            break;
        }
    }
}
```

Sempre deverá ser chamado o método

dg_DestroyLoggerControl (DestroyLoggerControl) para fechar as threads de controle antes de finalizar a VoicerLib.

4.19 Programação da Placa VB6060PCI

Em um feixe E1 podemos ter vários protocolos de sinalização. Esta documentação diz respeito à sinalização de linha R2D e de registro MFC 5C. Cada feixe E1 comporta 30 canais, mas como podemos ter 2 E1 em uma placa e mais que uma placa em um PC trataremos como porta o número sequencial de canais nos vários E1, ou seja, podemos ter porta 1,2,3....121,122,123 etc.

4.19.1 Configurações de Sincronismo

Devido à características existentes somente nas linhas digitais (E1) e também devido à existência do conector H100 utilizado para interconexão de placas, o desenvolvedor deve prestar atenção às configurações de sincronismo após a inicialização. Sempre após inicializar o driver (StartVoicerLib) é necessário configurar o modo de operação e o sincronismo das placas. Caso esta configuração seja omitida, a aplicação poderá

apresentar erros de sinalização, etc...

As placas VB6060PCI poderão operar em modo **MASTER** ou **SLAVE**. O modo **MASTER** deverá ser utilizado sempre quando houver uma ou mais placas instaladas no sistema sem que estejam conectadas entre si através do conector H100.

Se as placas estiverem conectadas entre si através via H100, obrigatoriamente uma deverá ser configurada com **MASTER** e as demais como **SLAVE**.



Com as placas interconectadas, **NUNCA** configure mais de uma como **MASTER**. Além de não fazer sentido em termos de operação, este procedimento poderá causar danos físicos às placas.

O sincronismo dependerá basicamente do ambiente onde as placas estão ligadas (tronco E1). Normalmente, se as placas estão conectadas diretamente à rede pública, esta fornece o sincronismo necessário, portanto as placas deverão ser configuradas com sincronismo **EXTERNO**. Em ambientes onde a placa VoicerBox seja a responsável por fornecer o sincronismo, este deverá ser configurado como **INTERNO**.

Tanto a informação de Master/Slave como as de sincronismo pode ser configuradas através da função **dg_SetH100Sync (SetH100Sync)**. Esta função permite a configuração completa de todo o barramento H100 porém seu funcionamento é relativamente complexo devido às muitas possibilidades existentes. Na maioria das aplicações que envolvam apenas placas Digivoice, somente quatro configurações diferentes são necessárias, é recomendado utilizar a função **dg_SetCardSyncMode (SetCardSyncMode)** ao invés da **dg_SetH100Sync (SetH100Sync)**.

A função **dg_SetCardSyncMode (SetCardSyncMode)** tem

apenas 2 parâmetros, sendo que o primeiro parâmetro é a placa na qual será aplicada a configuração e o segundo indicará qual dos 4 modos de operação será configurado, a saber:

- **MASTER_INTERNAL_SYNC (Valor 0)** - Placa MASTER com sincronismo interno
- **MASTER_SYNC_A (Valor 1)** - Placa MASTER com sincronismo externo no E1 A (Primeiro E1)
- **MASTER_SYNC_B (Valor 2)** - Placa MASTER com sincronismo externo no E1 B (Segundo E1, em placas de 60 canais)
- **SLAVE_MODE (valor 3)** - Placa operando em modo SLAVE

Uma forma de verificar se o sincronismo está configurado de maneira errada é a existência de eventos OnE1Alarm, principalmente com o status ALARM_RSLIP.

Exemplo (ActiveX):

```
Private Sub Iniciar()  
  
    'Inicia Driver  
    ret = VoicerLibX1.StartVoicerLib  
    if ret = 99 then  
        'placa iniciada, configura sincronismo  
        interno (hardcoded na placa 1)  
        VoicerLibX1.SetCardSyncMode 1,  
        MASTER_INTERNAL_SYNC  
    end if  
End Sub
```

4.19.2 Alarmes

A placa VB6060PCI gera eventos de alarmes indicando

qualquer tipo de problema nos troncos E1. Estes alarmes podem ser causados por falhas de configuração de sincronismo, discutido no tópico anterior, ou por algum tipo de problema de conexão física (cabos, etc...). A monitoração dos alarmes é essencial para o funcionamento das aplicações que utilizam a placa VoicerBox E1 30/60.

Por padrão inicial a notificação dos alarmes é manual, ou seja, a placa não envia estes alarmes para a VoicerLib e esta conseqüentemente não gera o evento **OnE1Alarm**. Se for necessário saber o *status* do alarme, é necessário chamar a função **GetAlarmStatus**.

Este comportamento pode ser alterado através da função **SetAlarmMode** na qual passa-se a placa e o modo de operação que pode ser *manual (padrão)* ou *automático*. Como já foi dito, no modo manual (ALARM_MANUAL_NOTIFY), o evento *OnE1Alarm* só é gerado após a chamada da função **GetAlarmStatus**. Já no modo automático (ALARM_AUTOMATIC_NOTIFY), o evento **OnE1Alarm** ocorre sempre quando um alarme for detectado. Recomenda-se utilizar o modo **automático** mas somente depois de se configurar o **sincronismo** das placas.

Os tipos de alarme possíveis são:

- ALARM_RSLIP (0x01) - Escorregamento (problema de sincronismo)
 - ALARM_RAIS (0x02) - Alarme remoto
 - ALARM_AISS (0x04) - Indicação de alarme
 - ALARM_AIS16S (0x08) - Indicação de alarme canal 16
 - ALARM_LOSS (0x10) - Perda de sinal
 - ALARM_RESERVED (0x20) - Reservado
 - ALARM_MFSYNC (0x40) - Sincronismo de multiquadro
 - ALARM_SYNC (0x80) - sincronismo de quadro
-

4.19.3 Protocolo R2D MFC

4.19.3.1 Inicialização

Para efetuar ou receber chamadas, a VoicerLib já traz embutida uma *thread* que executa o protocolo RD2 MFC 5C, a qual permite interagir com as centrais públicas, PABX com troncos E1, etc...

Após iniciar o driver através do método `StartVoicerLib`, é necessário iniciar uma *thread* individual para cada porta. Esta *thread* inicia internamente todo o tratamento de troca de sinalização com a "outra ponta" da ligação. Para iniciar esta *thread*, utiliza-se a função **`dg_CreateE1Thread (CreateE1Thread)`**, passando como parâmetro a porta desejada. Ao iniciar a *thread*, a porta é automaticamente colocada como **Livre**.

Também é necessário configurar corretamente as opções de sincronismo e modo de operação das placas E1. Recomenda-se utilizar o **`VBoxConfig`** para configurar todas estas opções, conforme mostrado no tópico **Utilizando o Configurator da placa E1** e, após chamar o método **`dg_CreateE1Thread (CreateE1Thread)`**, chamar o método **`dg_ReadConfigurationFile (ReadConfigurationFile)`** para que as configurações sejam lidas e aplicadas na aplicação.

4.19.3.2 Efetuando Chamadas

Sempre que for efetuar ligações em troncos E1 R2D MFC, é necessário que a porta do E1 tenha capacidade de enviar dados como sua identificação de A, categoria de assinante (grupo II), etc...

A identificação de A da porta (*BINA* a ser fornecido ao outro terminal) é configurada através do método **`dg_SetPortId (SetPortId)`** passando como parâmetro a porta e uma string com o número desejado (ex. `dg_setportid(porta, "1141952557")`).

Cada porta tem uma identificação própria na VoicerLib.

Todos os parâmetros de configuração relacionadas ao protocolo E1 são configurados, sempre **após** chamar o método **dg_CreateE1Thread (CreateE1Thread)** (início da thread) , através de chamadas à função **dg_ConfigE1Thread (ConfigE1Thread)**. O primeiro parâmetro é a porta, o segundo indica o dado a configurar e o terceiro o valor deste dado. O dado a configurar é indicado pelas constantes mostradas abaixo:

- E1CFG_MAXDIGITS_RX - Número máximo de dígitos a receber
- E1CFG_SEND_ID_AFTERDIGIT - Envio de solicitação de identificação após o dígito *n*
- E1CFG_GROUP_B - Grupo B
- E1CFG_CATEGORY - (Grupo II) Categoria do Assinante



É obrigatório chamar dg_ConfigE1Thread após a chamada de dg_CreateE1Thread.

Estando as informações sobre a porta corretamente configuradas, efetuar uma ligação segue os mesmo procedimentos que as placas analógicas PCI/4. Chama-se o método **dg_PickUp (PickUp)** para *ocupação*. Logo que vier a confirmação de ocupação, indicando que a porta está livre para discar, será gerado o evento OnDialToneDetected. No protocolo E1, não é gerado um tom de discagem como se conhece na telefonia convencional mas o evento de tom de discagem é gerado para compatibilizar as aplicações e informar ao programa a hora certa de iniciar a discagem.

No evento **EV_DIALTONE (OnDialToneDetected)**, deve-se chamar a função **dg_Dial (Dial)** indicando o número a discar. O primeiro parâmetro do método é a porta, o segundo indica o

número e o terceiro, que seria a pausa após a discagem, é desprezado no caso de discagens com a *thread* E1 criada.

A partir daí, ocorre toda uma troca de sinalização entre os terminais. Essa troca de sinalização pode ser monitorada através do evento **EV_E1CHANGESTATUS** (**OnE1ChangeStatus**). Este evento passa para a aplicação o estado da chamada na variável **Status**, que pode ter os seguintes valores:

- C_NOTCOMPLETED - Ligação não foi completada
- C_B_ENDCALL - Assinante B desligou
- C_ANSWERED - Assinante Atendeu ou retornou depois de um C_B_ENDCALL
- C_CONGESTION - Congestionamento
- C_SEIZURE - Ocupação
- C_NUMBER_RECEIVED - Número recebido durante a troca de sinalização
- C_ID_RECEIVED - O mesmo que EV_CALLERID
- C_GROUP_II - Recebeu categoria
- C_GRUPO_B - Recebeu pedido de grupo B
- C_UNAVAILABLE - Canal não disponível
- C_GROUP_I - Pedido de grupo I
- C_E1_SEIZURE_ACK
- C_E1_IDLE

Mais uma vez simulando uma ligação analógica convencional, a VoicerLib gerará os seguintes eventos a partir da discagem:

- EV_AFTERDIAL - Término da discagem
 - EV_BUSY - Linha ocupada, canal bloqueado ou timeout de atendimento
 - EV_ANSWERED - Destino atendeu
 - EV_CALLING - Ligação chamando. Este evento é gerado a cada 5 segundos.
-

Repare que estes eventos poderão ocorrer em situações iguais às indicadas pelos **Status** de **EV_E1CHANGESTATUS (OnE1ChangeStatus)**. Por exemplo, o evento **EV_BUSY (OnBusyDetected)** também ocorrerá quando for detectado uma situação de canal não disponível (**C_UNAVAILABLE**).

A qualquer momento durante a chamada, pode-se chamar a função **dg_HangUp (HangUp)** para finalizá-la.

Quando se faz ligações com as placas analógicas, não há como perceber se o assinante B desligou já que o telefone fica mudo até que passe os 90 segundos e venha o tom de ocupado da central pública. Na placa E1 30/60 é possível saber quando o assinante B desligou, monitorando-se o status **C_B_ENDCALL** dentro do evento **EV_E1CHANGESTATUS (OnE1ChangeStatus)**. Isso é muito útil em discadores que reproduzem mensagens automáticas, por exemplo. Caso o assinante B retorne à ligação, é gerado o evento **EV_E1CHANGESTATUS** com valor **C_ANSWERED**. Este evento também ocorre quando ocorre o primeiro atendimento.

4.19.3.3 Recebendo Chamadas

O recebimento de chamadas através de um canal digital também é facilmente manipulado a partir da chamada do método **dg_CreateE1Thread (CreateE1Thread)**.

Para ligações entrantes é necessário chamar à função **dg_ConfigE1Thread (ConfigE1Thread)** para configurar:

- **E1CFG_MAXDIGITS_RX** - Número máximo de dígitos a receber
- **E1CFG_SEND_ID_AFTERDIGIT** - Envio de solicitação de identificação após o dígito *n* (*Valor 0 não pede ID*)

- E1CFG_GROUP_B - Grupo B
- E1CFG_GROUP_II - Categoria do Assinante

O E1CFG_MAXDIGITS_RX deve ser informado porque o protocolo R2D exige que se saiba quantos dígitos deverão ser detectados. Já o E1CFG_SEND_ID_AFTERDIGIT indicará para a *thread* E1 o momento de se pedir a identificação do chamador. O **Grupo B** e o **Grupo II** variam conforme a aplicação e vêm com valor padrão 1.

Durante o recebimento de uma chamada, ocorre toda a troca de sinalização de linha R2D e de registro MFC 5C, conforme explicado no começo deste capítulo. Vários destes eventos podem ser monitorados através do evento **EV_E1CHANGESTATUS (OnE1ChangeStatus)**. Para simplificar a detecção da chegada de uma ligação, a VoicerLib também gera um evento **EV_RINGS (OnRingDetected)** logo após da troca de sinalização, permitindo que a aplicação atenda a chamada.

O atendimento de uma ligação *entrante* se dará a partir da chamada do método **dg_PickUp (PickUp)** e sua finalização através do método **dg_HangUp (HangUp)**.

Caso o *chamador* desligue, será gerado o evento **EV_BUSY (OnBusyDetected)** para a aplicação.

A identificação de chamadas (BINA) é feita pelo método **GetCallerID**. O evento **OnCallerID** sinaliza quando esta informação já está disponível.

4.19.3.4 Funções de Controle da Thread E1

Em algumas situações pode ser necessário parar a thread E1

para efetuar algum controle manualmente. Para isso foram criados os métodos **EnableE1Thread** (**dg_EnableE1Thread**) e **DisableE1Thread** (**dg_DisableE1Thread**). Com eles é possível desabilitar a thread ou reabilitá-la sem ter que destruir ou recriá-la, que são processos com maior consumo de processamento. Nas duas funções o único parâmetro é a porta.



Ao chamar o método CreateE1Thread não é necessário habilitá-la, pois ao criar, ela estará automaticamente habilitada. O EnableE1Thread só é útil após o uso de um DisableE1Thread

Também é possível saber se a thread E1 está habilitada ou não através do método **GetE1ThreadStatus** (**dg_GetE1ThreadStatus**). Este método recebe a porta como parâmetro e devolve os seguintes valores:

- DG_E1_THREAD_ENABLED (1)
- DG_E1_THREAD_DISABLED (0)

4.19.4 Protocolo CAS Customizado

4.19.4.1 Introdução

As placas VoicerBox E1 podem ser utilizadas também em posição de ramal, em PABX que oferecem este recurso. Normalmente, existem protocolos específicos por fabricante para emular a operação de um telefone analógico (Atender, Desligar, Flash, etc...).

A VoicerLib oferece algumas funções que permitem implementar essas funcionalidades:

```
dg_CreateCustomCAS (CreateCustomCAS)
dg_DestroyCustomCAS (DestroyCustomCAS)
dg_ConfigCustomCAS (ConfigCustomCAS)
```



Nem todas as situações de ramais CAS poderão estar previstas nessas funções. Novas implementações serão feitas pela DigiVoice à medida que novos protocolos apareçam.

4.20 Barramento H100

4.20.1 Introdução

O barramento H.100 está disponível na placa VoicerBox E1 30/60 e permite que canais de placas diferentes, ou mesmo canais de E1s diferentes da mesma placa possam ser comutados. Este processo também pode ser chamado de *bridge*.

A comutação entre canais é o processo onde é possível

conectar o áudio entre canais localmente (na mesma placa) ou entre canais de placas diferentes, através do barramento H100.

O barramento H100 disponibiliza vias (compostas de frames e slots) por onde o áudio será enviado, podendo ser de dois tipos:

Half Duplex - O áudio é enviado num único sentido. Neste caso um canal só reproduz e o outro só *ouve*

Full Duplex - O áudio é enviado nos dois sentidos. Os dois canais reproduzem e *ouvem* o áudio, típico de uma conversação.

Como exemplo, pode-se imaginar um sistema de atendimento automático onde existe uma opção de se transferir uma chamada para outro ramal. Em um sistema independente da existência de PABX, essa transferência pode ser feita através da comutação de canais. Uma seqüência simplificada será:

1. Canal **A** atende a chamada e reproduz as opções
2. O usuário escolhe uma opção para transferir para determinado ramal **XYZ**
3. O canal **B** efetua a chamada para o ramal **XYZ**, aguardando o atendimento.
4. Quando o usuário do ramal XYZ atender a ligação, o sistema conecta (comuta) o canal **A** com o canal **B**
5. O usuário conversa com o ramal **XYZ** entrando pelo canal A comutado com o canal **B**.

Obs.: Essa seqüência é mostrada na prática no exemplo de Comutação disponível no diretório de instalação ou no site da DigiVoice.

Todo esse processo de conexão entre canais pode ser feito exclusivamente pela função **SetBusConnection** porém todo o controle de canais, frames e slots utilizados deverá ser feito pelo programador, o que é uma tarefa relativamente complexa. Considerar usar esta função somente em casos de interconexão das placas DigiVoice com as de outro fabricante, ou mesmo

conexão entre portas de maneira não convencional.

Visando facilitar esse processo, a VoicerLib oferece um conjunto de funções (**h100_XXXXX**) que permitem o gerenciamento simplificado do barramento H100, *encapsulando* todos os detalhes da estrutura interna do H100.



IMPORTANTE: A DigiVoice fornece as placas digitais com ou sem o barramento H100. Caso a placa não tenha este barramento disponível as funções apresentadas neste capítulo serão ignoradas pela VoicerLib. Placas sem H100 ainda tem a opção de utilizar a função **LocalBridgeConnect** para conexões entre canais de uma mesma placa (veja *Conexão de placa sem H100*).

4.20.2 Conectando Portas

A VoicerLib sempre trata o sistema com as portas iniciando de 1, indo até N que dependerá do número de placas, de canais das placas e dos tipos de placas associadas. Por exemplo, em um sistema com duas placas da 60 canais teremos as portas numeradas de 1 à 120. Desta maneira, se for necessário conectar a porta 1 da primeira placa com a porta 1 da segunda placa, o programador deverá, na verdade, conectar a porta 1 à porta 61.

Sempre antes de utilizar uma porta no barramento H100, é necessário habilitá-la nesse barramento através do método **h100_EnablePort** (**dg_h100_EnablePort**). No caso de uma

conexão bi-direcional (mostrada abaixo) é preciso chamar esta função para cada porta a ser conectada. Ao chamar essas funções, o sistema aloca recursos do barramento automaticamente.

O processo de conexão é feito através das funções **h100_HalfDuplexConnect** (**dg_h100_HalfDuplexConnect**) e **h100_FullDuplexConnect** (**dg_h100_FullDuplexConnect**).

A conexão **half duplex** apenas envia o áudio em um sentido, da porta de origem à porta de destino, que são os dois parâmetros passados na função **h100_HalfDuplexConnect** (PortSource e PortTarget).

A conexão **full duplex**, fornecida pela função **h100_FullDuplexConnect** nada mais é que duas conexões **half duplex**, uma em cada sentido. Então, ao invés do programador fazer duas chamadas à função **h100_HalfDuplexConnect** conectando A->B e B->A, pode chamar apenas uma vez a função **h100_FullDuplexConnect**.



IMPORTANTE: Ao conectar-se o áudio de duas portas, as mesmas são desconectadas dos seus respectivos recursos de DSP, impedindo, nesta hora a detecção de dígitos, gravação, etc... Caso estes recursos sejam necessários durante a conversação, é necessário utilizar as funções descritas no tópico **Gravação em Paralelo das placas E1 30/60**.

4.20.3 Desconectando Portas

A desconexão das portas sempre deve ser feita para que os controles via DSP da placa voltem a atuar sobre o canal, permitindo detectar dígitos, reproduzir mensagens, etc...

A desconexão deve ser feita utilizando-se as funções **h100_HalfDuplexDisconnect** (**dg_h100_HalfDuplexDisconnect**) para conexões half duplex ou **h100_FullDuplexDisconnect** (**dg_h100_FullDuplexDisconnect**).

É importante ressaltar que mesmo na conexão half-duplex é necessário passar como parâmetros a porta de origem e a porta a qual esta está conectada. O controle de qual porta está conectada aonde deve ser feito pela aplicação final.

Estas funções desconectam a porta do barramento e retornam seu controle para os recursos de DSP da placa.

4.20.4 Outras funções H100

Uma função muito útil ao se utilizar os recursos de conexão do H100 é a **h100_GetFreePortByRange** (**dg_h100_GetFreePortByRange**). Ela serve para que o programador saiba qual porta, dentro de um determinado intervalo, está livre para uso. Isso facilita os processos de transferência onde é necessário alocar uma porta do E1 para discagem, etc...

Existem mais algumas funções no conjunto H100:

- **h100_AllocPort** (**h100_AllocPort**)
- **h100_EnablePort** (**dg_h100_EnablePort**)
- **h100_SetDefault** (**dg_h100_SetDefault**)
- **h100_DisconnectAll** (**dg_h100_DisconnectAll**)



Nenhuma destas funções precisa ser chamada diretamente pelo programador, já que as funções de conexão o fazem automaticamente, quando necessário. São citadas aqui somente para que se conheça melhor a estrutura interna da VoicerLib.

Em um primeiro momento, é necessário alocar um recurso do barramento H100 para uma determinada porta. No barramento deverá haver um *frame/slot* disponível para uma conexão full-duplex de duas portas. Isso é feito pela função **h100_AllocPort** (**h100_AllocPort**) passando a porta como parâmetro. Neste momento, o recurso do barramento foi alocado, porém nenhuma conexão ainda foi feita.

Em uma situação padrão, as portas estão sempre conectadas à recursos de DSP da placa, permitindo funções como: reprodução de mensagens, reconhecimento de dígitos, etc...

Após alocar um recurso do barramento para uma porta, é necessário habilitá-la nesse recurso. Isso é conseguido através da chamada da função **h100_EnablePort** (**dg_h100_EnablePort**) .

Na prática, ao habilitar uma porta no barramento não modificará seu comportamento porém, internamente, o canal passa a estar conectado no barramento e dali, conectado ao recurso de DSP da placa.

A função **h100_SetDefault** (**dg_h100_SetDefault**) ao ser chamada, faz com que a placa indicada no parâmetro passado

volte ao seu estado inicial, onde o framer E1 está conectado diretamente ao DSP. Essa é uma boa opção para desfazer eventuais conexões feitas e retornar a um estado inicial.

Também é possível desconectar todos os canais do barramento H100, utilizando-se a função **h100_DisconnectAll** (**dg_h100_DisconnectAll**). Esta função deixa todos os canais desconectados e é útil para o caso de se querer iniciar as conexões manualmente. Lembre-se que ao desconectar todos os canais com esta função, a placa deixará de funcionar até que as conexões sejam refeitas.

4.20.5 Conexão em placas E1 sem H100

Uma placa VBE16060 sem o barramento H100 disponível permite que dois canais sejam conectados, desde que sejam canais da mesma placa. Isso permitirá utilizar esse tipo de placa para aplicações onde seja necessário a conexão do áudio de 2 canais distintos, como por exemplo, sistemas de URA colocados entre o PABX e o tronco E1.

Para conectar dois canais, é necessário chamar a função **LocalBridgeConnect** (**dg_LocalBridgeConnect**) passando como parâmetro as duas portas que se deseja conectar. Caso sejam passadas portas de placas diferentes, a função retornará erro.

Após o uso dos canais conectados (durante uma conversação, por exemplo), é necessário reconectá-los ao DSP correspondente que fará o tratamento de detecções de dígitos,

reprodução de arquivos, etc... Para isso a função **LocalBridgeDisconnect** (**dg_LocalBridgeDisconnect**) deverá ser chamada com os mesmos parâmetros da função **LocalBridgeConnect** .

4.21 Funções Especiais

4.21.1 Introdução

As funções especiais são destinadas a executar tarefas comuns em telefonia de maneira mais simples para o programador.

Todas estas tarefas são plenamente realizáveis através das funções primitivas (PickUp, Dial, etc), porém procuramos reunir as tarefas mais comuns em grupos específicos:

- **Funções de Idle** – Atendimento Automático, Bina, Integração com PABX
- **Funções de Prompt** – Entrada de dados com ou sem conferência e confirmação (Senhas, etc...)
- **Funções de Menu** – Reprodução de menu de opções com consistência
- **Funções de Discagem e Transferência** – Discagem através de linha direta ou ramal, com opções para supervisão de ocupado, etc...

Os métodos e eventos apresentados a seguir são sempre indexados pelo parâmetro Port o que significa que as configurações são totalmente independentes por canal.



Para o pleno entendimento destas novas funcionalidades é muito importante estudar atentamente os exemplos fornecidos em Delphi e Visual Basic que encontram-se no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores.

4.21.2 Funções de Idle

API

ActiveX

As funções de **Idle** permitem ao programador simplificar os processos de espera de ligações:

- Atendimento Automático após n toques
- Detecção de Identificação do Assinante A (Bina) através do evento OnCallerID
- Detecção de Dígitos após o atendimento, facilitando integrações com diversos modelos de Pabx.

As funções de Idle são **IdleSettings**, **IdleStart** e **IdleAbort** (ou suas correspondentes dg_XXXX disponíveis na API).

Atendimento Automático



IMPORTANTE: As funções IdleXXXX são exemplificadas em um programa específico encontrado no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores. Acompanhe e teste o exemplo.

A VoicerLib permite ao programador, a partir de simples parâmetros no método **IdleSettings**, configurar um atendimento automático a partir do *enésimo* toque e ainda esperar um tempo pré-definido antes de chamar o evento **OnAfterPickup**.

Os parâmetros do **IdleSettings** que interferem no funcionamento desta função são o segundo, terceiro e quarto, sendo:

- 2º. Parâmetro (AutoPickUp) – Campo booleano (true/false) que indica se atende automático ou não
- 3º. Parâmetro (RingCount) – Número de toques para o atendimento
- 4º. Parâmetro (PauseAfterPickUp) – Pausa após o atendimento. Esta pausa pode ser utilizada para dar um tempo antes de iniciar a reprodução da mensagem de boas vindas, o que é útil em atendedores com bloqueio DDC (Chamada a cobrar).

Monitoração de Dígitos

A monitoração de dígitos pode ser feita antes do atendimento, como nos casos de identificação de chamadas direto do tronco ou depois do atendimento, particularmente útil em integração do tipo *inband* com centrais PABX.

Estas configurações também são feitas a partir do método **IdleSettings** do 5º. ao último parâmetro:

- 5º. Parâmetro (WatchTrunkBefore) – Indica que deve ser acionada a monitoração de dígitos antes do Ring. Deve ser utilizado principalmente em casos de identificação de chamadas.
- 6º. Parâmetro (WatchTrunkAfter) – Indica que deve ser acionada a monitoração de dígitos depois do Atendimento.
- 7º. Parâmetro (Format) – O formato indica como a espera de dígitos será tratada:
 1. wtDTMF/wtMFP – Deve ser utilizado somente com o

WatchTrunkBefore ligado para tratamento de Bina DTMF (wtDTMF) ou Bina MFP (wtMFP). Neste caso será gerado o evento OnCallerID contendo o número identificado.

2. wtCustom – Este formato traz os dígitos exatamente como foram detectados, sem nenhum tratamento. O resultado é dado no evento OnDigitsReceived. Funciona praticamente igual a um GetDigits.
- 8º. Parâmetro (TimeOut) – É o tempo máximo que o sistema esperará pelos dígitos a partir do primeiro detectado. Funciona como parâmetro InterdigitTimeout do método GetDigits e só tem utilidade quando o formato for wtCustom (neste caso o timeout global é dado pelo parâmetro PauseAfterPickup).
- 9º. Parâmetro (Max) – Indica qual o número máximo de dígitos que o sistema deverá esperar. Se não for utilizado, deve ser setado como 0 (zero).
- 10º. Parâmetro (TermDigits) – Indica qual o dígito terminador. Deixar vazio se não for utilizado.

4.21.3 Funções de Prompt



As funções de Prompt, disponíveis somente no ActiveX, facilitam os processos de entrada de dados e conferência em uma aplicação de URA. Como exemplo de utilização, observe a frase abaixo:

- *"Digite sua senha Você digitou 123.... Tecle # para confirmar ou * para cancelar"*
 - *"Disque o ramal desejado...."*
-



Para que os dígitos sejam reproduzidos corretamente, você deverá configurar corretamente a propriedade **StockSigsPath** apontando para a pasta onde os arquivos **wave** padrão estão armazenados

Basicamente as funções de prompt permitem reproduzir uma mensagem inicial (ou lista de mensagens), esperar por dígitos específicos (**maxdigits**, **termdigits**, etc...), reproduzir opcionalmente uma mensagem de conferência com as informações digitadas e ainda confirmar a digitação, permitindo a reentrada dos dados. Tudo isto pode ser feito através de parâmetros passados aos métodos **PromptSettings** e **PromptStart**. Exemplos destas funções encontram-se no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores, tanto em VB como em Delphi.

Ao chamar o método **PromptStart** a reprodução da mensagem é iniciada e o fluxo de execução segue normalmente, da maneira análoga ao método **PlayFile**, por exemplo. Quando as condições forem cumpridas de acordo com o configurado, o evento **OnPrompt** será acionado e receberá a variável **Status** contendo o que aconteceu na execução da função e o parâmetro **Value** contendo o dado digitado pelo usuário. Os detalhes do significado e funcionalidade de cada parâmetro deve ser consultado no Guia de Referência deste manual.

4.21.4 Funções de Menu



As funções de menu, disponíveis somente no ActiveX,

automatizam o processo de atendimento com opções de menu.
Ex.:

- *"...Para vendas disque 3, expedição 4 ..."*
- *".... Para saldo disque 1, para falar com o atendente 3..."*

O método **MenuErrorSettings** configura as respostas às situações de erro: frase de erro ("...Opção Inválida!..."), número de tentativas em caso de erro e frase para quando o usuário não discar nada.

O método **MenuStart** inicia a reprodução da frase de menu sendo passado como parâmetro a frase de opções, os dígitos válidos e o tempo máximo para digitação após a frase. Também deve ser informado se discagem por pulso será monitorada também.

Assim como o prompt, após a chamada do método **MenuStart**, o fluxo de execução segue normalmente. Após o usuário interagir com o menu, o evento **OnMenu** será chamado, recebendo o Status e a opção selecionada (OptionSelected).

Para interromper a execução do menu basta chamar o método **MenuAbort**. Os detalhes do significado e funcionalidade de cada parâmetro deve ser consultado no Guia de Referência deste manual. Exemplos dessas funções são fornecidos no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores, tanto em VB quanto em Delphi.

4.21.5 Funções de Discagem e Transferência



Os métodos e eventos que compõem esta funcionalidade

automatizam todo o processo de discagem, transferência, supervisão e atendimento através da chamada de um único método (MakeCall).



Para acompanhar a explicação, observe atentamente o exemplo de Transferência encontrado no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores, tanto em VB quanto em Delphi.

Com este grupo de métodos será possível efetuar de maneira simples:

- Discagem externa com supervisão de atendimento, verificando se o destino atendeu ou não ou ainda se deu ocupado
- Transferência entre ramais de um pabx com ou sem supervisão.
- Frases automáticas em caso de ocupado e não atender

Toda a configuração é feita através dos métodos SetCallxxxx.

Você pode programar um flash no início da discagem através do método **SetCallStartFlash**, útil em casos de transferência. Em termos de flash também é possível programar um flash específico para retomada em caso de ocupado (**SetCallBusyReturnFlash**) e outro em caso de não atendimento (**SetCallNoAnswerReturnFlash**). Em todos os casos pode-se definir dígitos e pausa após o flash e também pausa após dígito que são suficientes para adaptação em qualquer central PABX. O tempo de flash é uma configuração geral para todos os casos e é definida pelo método **SetCallFlashTime**.

É possível definir frases a serem reproduzidas quando se detecta o ocupado (**SetCallBusyPhrase**) ou quando não atende

(**SetCallNoAnswerPhrase**). O sistema utilizado o método **SetCallNoAnswerRingCount** para determinar com quantos toques será considerado um "não atendimento".

Em caso de atendimento é possível programar uma pausa e uma frase a ser reproduzida para o usuário. Isto é feito pelo método **SetCallAfterAnswer**.

O início da discagem é feito através da chamada ao método **MakeCall**, onde são definidos o tipo de discagem (externa ou com flash), a frase a ser reproduzida inicialmente, o número que será discado e se a discagem será feita com ou sem supervisão. Se for feita sem supervisão, tão logo o número é discado a execução do método é finalizada e o evento **OnAfterMakeCall** é chamado.

O tipo de discagem define uma série de comportamentos importantes:

- **Externa** – Inicia a discagem com um pickup pois entende que o telefone está desligado. Utiliza as definições de dígito e pausa após o pickup definidos pelo método **SetCallAfterPickUp**.
- **Com Flash** – Executa um flash antes de iniciar a discagem pois entende que a linha está conectada a um PABX e o que vai ser feito é uma transferência entre ramais.

Além das configurações anteriores, é possível determinar se o tom de discagem será esperado obrigatoriamente (**SetCallWaitForDialTone**) e o tempo que a supervisão será iniciada após a discagem (**SetCallPauseBeforeAnalysis**). Este último é útil em supervisões de transferência para casos onde o PABX gera ruídos durante o flash que podem induzir a placa a detecções erradas de atendimento. Definindo um tempo com este método a supervisão só iniciará a nnn milissegundos após a discagem, o que permitirá ignorar estes ruídos.

Assim como todos os outros métodos da VoicerLib, o **MakeCall** é assíncrono, ou seja, ao ser executado inicia o processo todo mas o fluxo de execução do programa continua.

IMPORTANTE: O MakeCall não pode ser chamado com a função IdleStart ativa. Caso isso ocorra, retornará erro código 1.

A qualquer momento é possível saber se um **MakeCall** está em curso através da monitoração do evento **OnCallStateChange**.

Por fim, o evento **OnAfterMakeCall** é chamado informando o tipo de ocorrência detectada durante o processo (ocupado, atendimento, etc...) através dos seguintes valores assumidos na variável Status:

- mkNoDialTone – Não foi detectado tom de discagem, caso esta opção tenha sido ligada (*).
- mkDelivered – Ligação entregue sem supervisão.
- mkNoAnswer – Destino não atendeu após n toques.
- mkBusy – Destino estava ocupado.
- mkAnswered – Ligação atendida.
- mkAborted – MakeCall cancelado através do método AbortCall.
- mkDialToneAfterDial - Indica recebimento de tom de linha depois da discagem.

() No caso de receber um **mkNoDialTone** depois de efetuar o Flash para transferência no PABX, você deve retomar a ligação antes de desligá-la para evitar de **prender** a linha no PABX. Uma situação de não receber tom de discagem para transferência entre ramais só pode ocorrer se as configurações de Flash estiverem erradas ou no caso de sobrecarga de processamento do PABX (ocorre principalmente em discadores automáticos).*

4.21.6 Reproduzindo Data



A VoicerLib permite ao programador implementar sistemas que falem datas através do método **PlayDate**. O formato da função é:

```
v1b.PlayDate Porta, Date, "d/m/y", "#", 0
```

O primeiro parâmetro refere-se ao canal da placa. O segundo parâmetro é uma string contendo a data a ser reproduzido (pode-se utilizar uma variável aqui ou mesmo a função Date, que retorna a data corrente).

O terceiro parâmetro é a máscara de formatação da data. Esta máscara determina como a data será falada. Pode assumir os seguintes valores:

- **d/m/y** – Ex.: "25 de setembro de 2001"
- **d/m** – Ex.: "25 de setembro"
- **m/d** – Ex.: "Setembro 25"
- **m/d/y** – Ex.: "Setembro 25 2001"

O quarto parâmetro é o TermDigits, que indica quais dígitos podem interromper a mensagem.

O último parâmetro é uma pausa a ser respeitada antes da reprodução da mensagem, expressa em milissegundos, ou seja, caso queira se esperar 1 segundo utilize o valor 1000.

Mesmo utilizando a pausa, a função retorna imediatamente após sua execução. Qualquer tratamento de seu resultado deverá ser feito no evento **OnPlayStop** ou **OnDigitsReceived**.

As frases-padrão utilizadas para reproduzir esta mensagem encontram-se no CD de distribuição, na pasta StockSigs. Os

arquivos contidos nesta pasta deverão ser distribuídos junto da aplicação desenvolvida.

4.21.7 Reproduzindo Números Cardinais



A VoicerLib permite ao programador implementar sistemas que falem números inteiros ou fracionários por extenso (ex.:Dez vírgula trinta e cinco) através do método **PlayCardinal**.

O formato da função é:

```
v1b.PlayCardinal Porta, "10,35", "#", 0
```

O primeiro parâmetro refere-se ao canal da placa. O segundo parâmetro é uma string contendo o número a ser reproduzido (pode-se utilizar uma variável aqui, é claro).



É necessário passar o valor formatado de acordo com o mostrado acima, sem pontos separadores nos milhares e com vírgula como separador decimal (nnnnnn,nn). Qualquer coisa diferente disso fará com que a função não reproduza o valor corretamente.

O terceiro parâmetro é o TermDigits, que indica quais dígitos podem interromper a mensagem.

O último parâmetro é uma pausa a ser respeitada antes da reprodução da mensagem, expressa em milissegundos, ou seja, caso queira se esperar 1 segundo utilize o valor 1000.

Mesmo utilizando a pausa, a função retorna imediatamente após sua execução. Qualquer tratamento de seu resultado

deverá ser feito no evento **OnPlayStop** ou **OnDigitsReceived**.

As frases-padrão utilizadas para reproduzir esta mensagem encontram-se no CD de distribuição, na pasta StockSigs. Os arquivos contidos nesta pasta deverão ser distribuídos junto da aplicação desenvolvida.

4.21.8 Reproduzindo Números Dígitos a Dígitos



A VoicerLib permite ao programador implementar sistemas que falem números através do método **PlayNumber**. Com isso é possível "soletrar" os dígitos de um dado qualquer (conta, cartão, etc...). Também permite falar "/" barra, "-" traço, "." Ponto, "," – vírgula.

O formato da função é:

```
v1b.PlayNumber, "456790-23", "#", 0
```

O primeiro parâmetro refere-se ao canal da placa. O segundo parâmetro é uma string contendo o número a ser reproduzido (pode-se utilizar uma variável aqui, é claro).

O terceiro parâmetro é o TermDigits, que indica quais dígitos podem interromper a mensagem.

O último parâmetro é uma pausa a ser respeitada antes da reprodução da mensagem, expressa em milissegundos, ou seja, caso queira se esperar 1 segundo utilize o valor 1000.

Mesmo utilizando a pausa, a função retorna imediatamente após sua execução. Qualquer tratamento de seu resultado deverá ser feito no evento **OnPlayStop** ou **OnDigitsReceived**.

As frases-padrão utilizadas para reproduzir esta mensagem encontram-se no CD de distribuição, na pasta StockSigsPath. Os arquivos contidos nesta pasta deverão ser distribuídos junto da aplicação desenvolvida.

4.21.9 Reproduzindo Valores por Extenso



A VoicerLib permite ao programador implementar sistemas que falem valores monetários por extenso (ex.: Um mil e quinhentos reais e trinta e dois centavos) através do método **PlayCurrency**.

O formato da função é:

```
vlb.PlayCurrency Porta, "1234,33", "#", 0
```

O primeiro parâmetro refere-se ao canal da placa. O segundo parâmetro é uma string contendo o valor a ser reproduzido (pode-se utilizar uma variável aqui, é claro).



É necessário passar o valor formatado de acordo com o mostrado acima, sem pontos separadores nos milhares e com vírgula como separador décima (nnnnnn,nn). Qualquer coisa diferente disso fará com que a função não reproduza o valor corretamente.

O terceiro parâmetro é o TermDigits, que indica quais dígitos podem interromper a mensagem.

O último parâmetro é uma pausa a ser respeitada antes da reprodução da mensagem, expressa em milissegundos, ou seja, caso queira se esperar 1 segundo utilize o valor 1000.

Mesmo utilizando a pausa, a função **PlayCurrency** retorna

imediatamente após sua execução. Qualquer tratamento de seu resultado deverá ser feito no evento **OnPlayStop** ou **OnDigitsReceived**.

4.21.10 Reproduzindo Lista de Mensagens



Uma das mais importantes características da VoicerLib é a possibilidade de montar uma lista de mensagens que serão reproduzidas como se fossem uma só, gerando apenas um evento **OnPlayStart** no início e um **OnPlayStop** no final de todas.

Com isso o programador poderá montar frases ou seqüências de mensagens de maneira muito simples.

A lista de mensagens permite que sejam inseridas mensagens baseadas em arquivos, data, hora, valor ou numeral cardinal.

Vamos supor a seguinte frase: "Olá, hoje é dia 25 de setembro de 2001 às 12:32:45". Temos 4 partes distintas nesta frase:

- "Olá, hoje é dia" – Arquivo pré-gravado do usuário
- "25 de setembro de 2001" – Data do Sistema (PlayDate)
- "às" – Arquivo pré-gravado do usuário
- "12:32:45" – Hora do Sistema (PlayTime)

Adicionando Mensagens a Lista

Para adicionar uma ou mais mensagens a uma lista, deve-se utilizar o método **PlayListAdd** que tem o seguinte formato:

```
Voicel.PlayListAdd Porta, Tipo, String, Máscara,  
PausaAntes
```

O parâmetro `Porta` refere-se ao canal a ser utilizado. A `VoicerLib` permite manter uma lista independente por canal.

O parâmetro `Tipo` indica que tipo de mensagem será reproduzida. Pode assumir os seguintes valores (representados por constantes):

- `ptCardinal` – Reproduzir um numeral cardinal (~ `PlayNumber`)
- `ptFile` – Reproduzir um arquivo `.SIG`
- `ptDate` – Reproduzir uma data (~ `PlayDate`)
- `ptTime` – Reproduzir hora (~ `PlayTime`)
- `ptCurrency` – Reproduzir um valor monetário
- `ptNumber` – Reproduz números digito a digito.

O terceiro parâmetro (`String`) deve assumir a formatação exigida pelos tipos acima, ou seja se for indicado que é do tipo `ptFile`, este parâmetro deve receber uma string com o nome do arquivo a ser reproduzido. Para saber os formatos corretos para cada tipo, consulte as funções independentes relacionadas (`PlayTime`, `PlayDate`, etc...) no começo desta seção.

A `Máscara` (4º. Parâmetro) é relacionada apenas ao tipo `ptDate` e segue a mesma padronização indicada no método `PlayDate` explicado anteriormente.

O último parâmetro é uma pausa a ser respeitada antes da reprodução da mensagem, expressa em milissegundos, ou seja, caso queira se esperar 1 segundo utilize o valor 1000. Cada mensagem da lista pode ter um valor diferente, ou seja, é possível dar pausas diferenciadas entre cada item da lista na hora da reprodução. Seguindo o exemplo de frase do começo desta seção, teríamos a seguinte codificação:

```
voicel.PlayListAdd Port, ptFile,  
"olahojeedia.sig", "", 0
```

```
voicel.PlayListAdd Port, ptDate, Date, "d/m/y", 0
voicel.PlayListAdd Port, ptFile, "as.sig", "", 0
voicel.PlayListAdd Port, ptTime, Time, "", 0
```

Apagando o Conteúdo de Uma Lista

A lista de mensagens permanece com seu conteúdo mesmo depois da reprodução portanto, é necessário apagar seu conteúdo antes de adicionar novos valores. Para isso deve ser utilizado o método `PlayListClear` passando como parâmetro o canal da placa:

```
voicel.PlayListClear Port      'Apaga a lista antes
de usá-la
voicel.PlayListAdd Port, ptFile,
"olahojeedia.sig", "", 0
(...)
```

Verificando o tamanho da lista

Para saber quantos elementos existem dentro da lista, utilize o método `PlayListGetCount`, onde o único parâmetro é o canal e o valor de retorno, é a quantidade de elementos.

```
Dim i as integer

i = voicel.PlayListGetCount(Port)
```

Removendo um item Específico da Lista

Também é possível remover um determinado elemento da lista através do método `PlayListRemoveItem`. Este método tem dois parâmetros: a Porta e o índice do item a ser excluído (0 até n-1).

```
'remove o primeiro elemento da lista
voicel.PlayListRemoveItem(Port,0)
```

Reproduzindo a Lista de Mensagens

Após inserir os elementos da lista é possível reproduzi-la através do método **PlayList**, que tem o seguinte formato:

```
voicel.PlayList Port,TermDigits
```

O primeiro parâmetro indica o canal da placa. Lembre-se que cada canal tem uma lista independente.

O parâmetro TermDigits é uma string que permite configurar quais dígitos poderão interromper a mensagem, exatamente como funciona nos outros métodos Playxxx.

O método **PlayList** também é assíncrono, ou seja, ao executar o comando o fluxo de execução do aplicativo segue imediatamente. O final da reprodução deverá ser monitorado através do evento **OnPlayStop**.

O método PlayList, apesar de reproduzir várias mensagens encadeadas, funciona da mesma forma que o **PlayFile**, ou seja, gera apenas um **OnPlayStart** no início e um **OnPlayStop** no final do último item.

Caso o parâmetro TermDigits seja utilizado e a lista de mensagens seja interrompida por dígito, toda a lista será interrompida e serão gerados os eventos **OnPlayStop** e **OnDigitsReceived**, sendo que este último receberá o valor edDigitOverMessage na variável Status.

Na dúvida, leia novamente a explicação sobre o método **PlayFile** Tudo o que se refere a TermDigits aplica-se ao **PlayList** também.

4.22 Distribuindo uma Aplicação

Visando facilitar a distribuição de todos os arquivos necessários para a execução de uma aplicação construída utilizando a VoicerLib, as placas adquiridas sempre são acompanhadas de

um software chamado Kit Integrador.

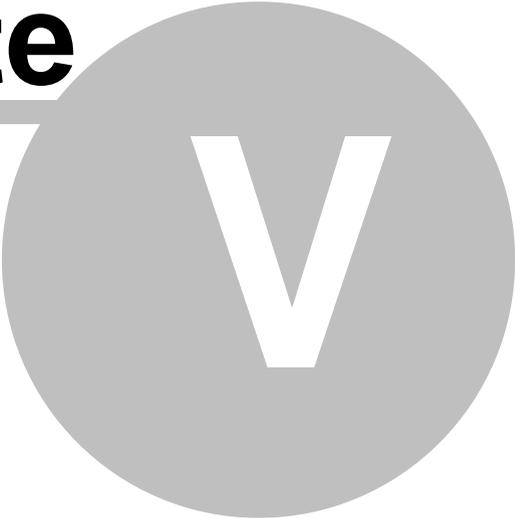
A melhor forma de distribuir uma aplicação desenvolvida é fazendo uso do Kit Integrador que acompanha a placa, instalando-o na máquina de destino. O procedimento de instalação, tanto físico como de software é abordado pelo Manual do Kit Integrador.

Com o kit integrador instalado, não é necessário mais nenhum arquivo adicional para que uma aplicação funcione. Obviamente os requisitos de cada linguagem de programação deve ser atendidos pelo desenvolvedor (runtimes, dlls, etc...).



É muito importante que a aplicação seja desenvolvida e testada com a VoicerLib na mesma versão do Kit Integrador. As versões mais atualizadas sempre estarão disponíveis no site da DigiVoice na Internet.

Parte



V

Guia de Referência

5 Guia de Referência

5.1 Mensagens de Erro

Todas as funções retornam valores indicando sucesso ou problema na sua execução. Nas funções são indicados os nomes das constantes que representam estes códigos e abaixo são enumerados as constantes com seus respectivos valores, em hexadecimal e decimal.

Constante	Hexa	Decimal	Descrição
DG_EXIT_SUCCESS	0	0	Executado com sucesso
DG_EXIT_FAILURE	700h	1792	Falha genérica, não especificada pelos códigos abaixo
DG_ERROR_MEMORY_ALLOCATION	400h	1024	Falha de alocação de memória. Pode ser por falta de memória, mas também por erro no device driver, direitos de usuário (Linux), etc
DG_ERROR_MAXCARDS	401h	1025	Falha na recuperação do número de placas instaladas.
DG_ERROR_FIRMWARE_NOT_FOUND	402h	1026	Arquivo de firmware não foi encontrado. Pode ser causado por problemas de instalação ou no caso de uso com ActiveX a propriedade ConfigPath pode estar com valores errados.
DG_ERROR_FIRMWARE_IO_TIMEOUT	403h	1027	Erro na carga do firmware. Pode ser causado por slot defeituoso. Contate a assistência técnica.
DG_ERROR_READING_PORTCOUNT	404h	1028	Falha na leitura do número de portas por placa. Pode ser causado por falha de hardware ou erro durante a instalação.
DG_ERROR_LOADING_DEVICE_DRIVER	405h	1029	Erro na carga do device driver. As mensagens de debug podem esclarecer a causa.
DG_ERROR_CREATING_EVENT	406h	1030	Erro na instalação de eventos de controle da VoicerLib. Só pode ser causado por falta de memória.
DG_ERROR_DRIVER_CLOSED	450h	1104	A Função foi chamada sem que a Voicerlib fosse iniciada

Constante	Hexa	Decimal	Descrição
DG_ERROR_CARD_OUT_OF_RANGE	451h	1105	Parâmetro placa foi passado com valor maior do que o número de placas instaladas
DG_ERROR_PORT_OUT_OF_RANGE	452h	1106	Parâmetro porta passado com valor maior que o número total de portas instaladas (somadas todas as placas).
DG_ERROR_PARAM_OUTOF_RANGE	453h	1107	Um dos parâmetros da função foi passado fora do intervalo especificado. Consulte a referência da função.
DG_ERROR_DRIVER_ALREADY_OPEN	454h	1108	Pode ocorrer se o StartVoicerLib for chamado duas vezes seguidas
DG_ERROR_CONFIGFILE_NOT_FOUND	455h	1109	O arquivo de configuração passado por parâmetro não foi encontrado. Não é necessário passar o caminho completo do arquivo.
DG_FEATURE_NOT_SUPPORTED	456h	1110	Esta função não é suportada pela placa utilizada.
DG_ERROR_RESOURCE_UNAVAILABLE	457h	1111	Normalmente associada às funções de chat, este erro ocorre quando existem "salas" disponíveis, porém não foi possível alocá-las por falta de memória
DG_ERROR_RESOURCE_FULL	458h	1112	Normalmente associada às funções de chat, este erro ocorre quando não existem mais "salas" disponíveis.
DG_ERROR_INVALIDPARAM	459h	1113	Foi passado um parâmetro com valor inválido.

Constante	Hexa	Decimal	Descrição
DG_ERROR_COULD_NOT_CREATE_THREAD	500h	1280	Falha na inicialização interna das threads de controle. Este erro normalmente é causado por falta de memória livre.
DG_ERROR_THREAD_NOT_RUNNING	501h	1281	Este erro ocorre quando se tenta chamar uma função de manipulação de threads de controle sem que a mesma tenha sido iniciada (ex. chamada do EnableCallProgress sem ter chamado CreateCallProgress).
DG_ERROR_FIFO_UNAVAILABLE	502h	1282	Fifo de comunicação entre threads não está disponível. Falta de memória pode ser a causa mais comum.
DG_ERROR_THREAD_ALREADY_RUNNING	503h	1283	Este erro é causado caso se chame uma função de criação de Callprogress, ThreadE1, etc... mais de uma vez
DG_ERROR_REC_STOPPING	600h	1536	Foi chamado o comando StopRecordFile enquanto uma gravação estava sendo finalizada
DG_ERROR_REC_OPENFILE	601h	1537	Erro na criação do arquivo de gravação.
DG_ERROR_REC_ALREADY_RECORDING	602h	1538	Já existe uma gravação em curso.
DG_ERROR_REC_NOT_RECORDING	603h	1539	Chamada do StopRecordFile quando não há gravação em curso.

Constante	Hexa	Decimal	Descrição
DG_ERROR_NOT_PLAYING	550h	1360	Tentativa de interromper a reprodução sendo que não há nenhuma em curso
DG_ERROR_ALREADY_PLAYING	551h	1361	Tentativa de iniciar uma reprodução quando já existe uma em curso
DG_ERROR_PLAY_OPENFILE	552h	1362	Não foi possível localizar ou abrir um arquivo para reprodução
DG_ERROR_PLAY_EMPTYLIST	553h	1363	Tentativa de reprodução de lista de mensagens através do PlayList, MenuStart ou PromptStart sem haver uma lista criada.
DG_ERROR_AUDIO_FORMAT_UNSUPPORTED	554h	1364	Formato de áudio não suportado pela placa
DG_ERROR_PLAY_INVALID_FILENAME	555h	1365	Nome de arquivo inválido
DG_ERROR_PLAY_BUFFER_FULL	556h	1366	Buffer de reprodução via streaming cheio, significando que as amostras de áudio estão sendo enviadas numa taxa maior do que a VoicerLib pode enviar ao hardware.

5.2 Funções/Métodos

No Guia de Referência são apresentadas todas as funções disponíveis na API (DLL e Shared Object) como os métodos disponíveis no ActiveX para Windows.

Algumas funções estão disponíveis somente para o ActiveX e outras somente na API. Também há alguma diferenciação quanto à placa que suporta determinada função. Os ícones no início de cada tópico indicam este nível de compatibilidade.

5.2.1 AbortCall



Cancela a discagem automática feita pelo método **MakeCall**.

Declarações:

ActiveX:
`SHORT AbortCall(SHORT Port);`

Descrição:

Durante o processo de discagem automática, iniciado pelo método **MakeCall**, é possível cancelar através da chamada deste método

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

Retorna DG_EXIT_SUCCESS se foi executado com sucesso.

5.2.2 CancelGetDigits (dg_CancelGetDigits)



Cancela a detecção de dígitos iniciada pelo GetDigits (dg_GetDigits).

Declarações:

ActiveX:

```
SHORT CancelGetDigits(SHORT Port);
```

API:

```
short dg_CancelGetDigits(short port);
```

Descrição:

Quando o processo de recepção de dígitos acionado pelo método GetDigits é iniciado, a VoicerLib entra em um tratamento interno que pode durar o tempo dos timeouts definidos no GetDigits. É conveniente, caso a ligação seja encerrada em um determinado canal enquanto o GetDigits esteja sendo executado, cancelar esta recepção chamando o CancelGetDigits.

Parâmetros:

Port – Indica o canal da Placa no qual será cancelado o GetDigits

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro **port** fora do intervalo permitido, excedendo o número de canais instalados

5.2.3 ClearDigits (dg_ClearDigits)

API

ActiveX

Apaga o conteúdo do buffer de dígitos de cada canal.

Declarações:

ActiveX:

```
SHORT ClearDigits(SHORT Port);
```

API:

```
short dg_ClearDigits(short port);
```

Descrição:

É utilizada para apagar o buffer interno de dígitos detectados.

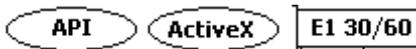
Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.4 ChatAddPort (dg_ChatAddPort)



Adiciona canal a determinada sala de conferência.

Declarações:ActiveX:

```
SHORT ChatAddPort(LONG Handle, SHORT Port);
```

API:

```
short dg_ChatAddPort(long Handle, short Port)
```

Descrição:

Esta função permite adicionar determinada porta à sala de conferência especificada por *Handle*. Para que a porta possa falar e ouvir na conferência é necessário também habilitá-la usando **ChatEnablePort**.

Parâmetros:

Handle - Obtido como retorno da função CreateChatRoom
Port - Canal físico da placa.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_RESOURCE_FULL - Sala cheia
DG_ERROR_RESOURCE_UNAVAILABLE - Sala não disponível ou não criada

5.2.5 ChatDisablePort (dg_ChatDisablePort)



Desabilita porta alocada em um recurso de conferencia.

Declarações:ActiveX:

```
SHORT ChatDisablePort(LONG Handle, SHORT Port);
```

API:

```
short dg_ChatDisablePort(long Handle, short Port);
```

Descrição:

Esta função desabilita a porta da sala mas sem removê-la. Isto é particularmente útil para que determinada porta possa interagir com o sistema (por exemplo, ouvindo um menu) sem que as outras portas ouçam. Ao desabilitar uma porta, ela é reconectada no recurso de DSP.

Parâmetros:

Handle - Obtido como retorno da função CreateChatRoom
Port - Canal físico da placa.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_RESOURCE_UNAVAILABLE - Sala não disponível ou não criada

5.2.6 ChatEnablePort (dg_ChatEnablePort)



Habilita porta alocada em um recurso de conferencia a conversar com as outras.

Declarações:ActiveX:

```
SHORT ChatEnablePort(LONG Handle, SHORT Port,  
SHORT Direction);
```

API:

```
short dg_ChatEnablePort(long Handle, short Port,  
short Direction);
```

Descrição:

Mesmo que a porta esteja alocada em uma determinada conferencia é necessário que ela esteja habilitada para que possa se comunicar com as outras portas da mesma sala. O parâmetro **Direction** define como esta porta se comportará, podendo só ouvir, só falar ou falar e ouvir na sala.

Parâmetros:

Handle - Obtido como retorno da função CreateChatRoom

Port - Canal fisico da placa.

Direction - Define mode de comportamento da porta na sala

CHAT_TALK - A porta só *falará* na sala

CHAT_LISTEN - A porta só *escutará* na sala

CHAT_TALK_LISTEN - A porta poderá *falar* e *ouvir* na

sala.

Valor de Retorno:

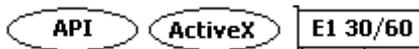
DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido (Direction)

DG_ERROR_RESOURCE_UNAVAILABLE - Sala não disponível ou não criada

5.2.7 ChatRemovePort (dg_ChatRemovePort)



Remove canal de determinada conferencia.

Declarações:

ActiveX:

```
SHORT ChatRemovePort(LONG Handle, SHORT Port);
```

API:

```
short dg_ChatRemovePort(long Handle, short Port)
```

Descrição:

Esta função desabilita e remove uma porta de determinada sala de conferência.

Parâmetros:

Handle - Obtido como retorno da função CreateChatRoom

Port - Canal físico da placa.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_RESOURCE_UNAVAILABLE - Sala não disponível ou não criada

5.2.8 ConfigCallProgress (dg_ConfigCallProgress)

API

ActiveX

Configura as opções de tratamento da *thread* para supervisão de linha

Declarações:ActiveX:

```
SHORT ConfigCallProgress(SHORT Port, SHORT Cmd,  
LONG Value);
```

API:

```
short dg_ConfigCallProgress(short port, short
```

```
command, int value);
```

Descrição:

Se na chamada ao `CreateCallProgress` não for indicando um arquivo de configuração, o método **ConfigCallProgress** poderá ser chamado para modificar qualquer um das configurações de controle de supervisão de linha. Na tabela abaixo é mostrado os valores de **Command** e os limites de **Value**.

A VoicerLib já fornece um arquivo de configuração chamado **cp_default.cfg** contendo as configurações mais comuns. A utilização do **ConfigCallProgress** é indicada quando o desenvolvedor preferir manter seu próprio arquivo de configurações.

Configurações de Tempos de detecção:

Command	Descrição	Valor padrão
CPCFG_ANSWER_SENSITIVITY	AnswerSensitivity refere-se a quantos "audios" deverão ser recebidos na sequencia para considerar atendimento	1
CPCFG_ANSWER_SENSITIVITY_TIME	Tempo mínimo para que um sinal de áudio seja considerado atendimento.	200
CPCFG_GENERICTONETIMEOUT	GenericToneTimeout determina o tempo máximo que o sistema esperará para reconhecer o tom genérico	15000
CPCFG_GENERICTONETIME	GenericToneTime determina o tempo "mínimo" para que o sistema reconheça o áudio como tom genérico	500
CPCFG_LINETONETIMEOUT	LineToneTimeout determina o tempo "máximo" que o sistema esperará para reconhecer o tom de linha	15000
CPCFG_LINETONETIME	LineToneTime determina o tempo "mínimo" para que o sistema reconheça o áudio como tom de linha	1000
CPCFG_FAXTONETIMEOUT	FaxToneTimeout determina o tempo "máximo" que o sistema esperará para reconhecer o tom de fax	15000
CPCFG_FAXTONETIME	FaxToneTime determina o tempo "mínimo" para que o sistema reconheça o áudio como tom de linha	500
CPCFG_CALLPROGRESSTIMEOUT	CallProgressTimeout é o tempo de espera após a discagem para se considerar que houve um atendimento por timeout	1500
CPCFG_BUSYMINTIME	BusyMinTime determina o tempo minimo do tom de ocupado	100
CPCFG_BUSYMAXTIME	BusyMaxTime determina o tempo máximo do tom de ocupado	500
CPCFG_CALLINGMINTONETIME	Determina o tempo minimo para se considerar um tom de chamada (apos a discagem). Será testado como intervalo junto com CallingMaxToneTime	600
CPCFG_CALLINGMAXTONETIME	Determina o tempo maximo para se considerar um tom de chamada (apos a discagem)	2000
CPCFG_CALLINGMINSILTIME	O tempo minimo para se considerar silencio. Será testado como intervalo junto com CallingMaxSilTime	700
CPCFG_CALLINGMAXSILTIME	Determina o tempo maximo para se considerar silencio	5000
CPCFG_TONEINTERRUPTIONMINTIME	Determina o tempo minimo do intervalo entre cada tom de linha. O detecção do tom de chamando depende tambem dos tempos do intervalo entre eles.	20
CPCFG_TONEINTERRUPTIONMAXTIME	Determina o tempo minimo do intervalo entre cada tom de linha. O detecção do tom de chamando depende tambem dos tempos do intervalo entre eles.	380
CPCFG_LINETONEMINTIME	determina o tempo minimo para se considerar o tom de linha	2500
CPCFG_LINETONEMAXTIME	Determina o tempo maximo para se considerar o tom de linha	2700

Configuração das frequências

Command	Descrição	Valor padrão
CPCFG_LINEFREQ	Índice da frequência do tom de linha	22h
CPCFG_CALLINGFREQ	Índice da frequência	22h
CPCFG_BUSYFREQ	Índice da frequência	22h
CPCFG_GENERICFREQ	Índice da frequência	25h
CPCFG_SILENCE	Índice da frequência	20h
CPCFG_AUDIO	Índice da frequência	21h
CPCFG_FAX1	Índice da frequência	23h
CPCFG_FAX2	Índice da frequência	24h

Na tabela de configurações de frequências não deve ser informado o valor da frequência e sim o valor do índice utilizado pela VoicerLib. Verifique no tópico "**Supervisão de Linha**" no **Guia do Programador**.

Parâmetros:

Port – Porta que será iniciado o controle

Command - Indica qual a informação será configurada de acordo com as constantes da tabela acima.

Value - Valor do dado indicado por **Command**

Valores de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso

DG_ERROR_DRIVER_CLOSED - O driver não foi iniciado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_FEATURE_NOT_SUPPORTED - Comando não suportado por este tipo de placa

DG_ERROR_THREAD_NOT_RUNNING - A thread E1 não foi inicializada ainda

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro

Command ou **Value** fora do especificado

5.2.9 ConfigE1Thread (dg_ConfigE1Thread)

API

ActiveX

E1 30/60

Configura as opções de tratamento da *thread* em relação ao protocolo.

Declarações:

ActiveX:

```
SHORT ConfigE1Thread(SHORT Port, SHORT Cmd, LONG Value);
```

API:

```
short dg_ConfigE1Thread(short port, short command, int value);
```

Descrição:

Sempre após chamar o **CreateE1Thread (dg_CreateE1Thread)**, é necessário configurar algumas informações que indicará o comportamento da sinalização R2D durante a troca de sinalização. Na tabela abaixo é mostrado os valores de **Command** e os limites de **Value**.

Command	Descrição	Value
E1CFG_MAXDIGITS_RX	Número total de dígitos a receber. Zero indica que a <i>thread E1</i> não fará tratamento automático do recebimento de dígitos.	> 0
E1CFG_SEND_ID_AFTERDIGIT	Configura a partir de qual dígito será solicitada a identificação. Zero indica que não enviará identificação	>= 0
E1CFG_GROUP_B	Informa o grupo B	>= 0
E1CFG_GROUP_II	Informa o grupo II - Categoria do Assinante	>= 0
SILENCE_THRESHOLD	Informa o limiar de silêncio durante a sinalização	-30
SILENCE_THRESHOLD_AFTER	Informa o limiar de silêncio a ser usado depois do atendimento	-28
E1CFG_SEIZURE_TIMEOUT	Timeout de ocupação	7000ms
E1CFG_RETENTION_TIMEOUT	Timeout de retenção	9000ms
E1CFG_ANSWER_TIMEOUT	Timeout de atendimento	75000ms
E1CFG_BLOCKING_ON	Tempo de canal atendido para bloqueio	1000ms
E1CFG_BLOCKING_OFF	Tempo de canal desligado para bloqueio	2000ms
E1CFG_MFT_TIMEOUT	Timeout de MF para Trás	30000ms
E1CFG_MFF_TIMEOUT	Timeout de MF para Frente	30000ms

Os parâmetros de timeouts já estão configurados de acordo com a norma do protocolo R2D e só deverá ser alterada se for realmente necessário. Os quatro primeiros parâmetros da lista acima deverão ser configurados sempre, de acordo com o ambiente de produção.

Parâmetros:

Port – Porta que será iniciado o controle

Command - Indica qual a informação será configurada de acordo com as constantes da tabela acima.

Value - Valor do dado indicado por **Command**

Valores de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso

DG_ERROR_DRIVER_CLOSED - O driver não foi iniciado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_FEATURE_NOT_SUPPORTED - Comando não suportado por este tipo de placa

DG_ERROR_THREAD_NOT_RUNNING - A thread E1 não foi inicializada ainda

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro

Command ou **Value** fora do especificado

Veja Também: CreateE1Thread, DestroyE1Thread

5.2.10 ConfigCustomCAS (dg_ConfigCustomCAS)

API

ActiveX

E1 30/60

Configura as opções de tratamento da *thread* CAS

Declarações:

ActiveX:

```
SHORT ConfigCustomCAS(SHORT Port, SHORT Cmd, LONG Value);
```

API:

```
short dg_ConfigCustomCAS(short port, short command, int value);
```

Descrição:

Sempre após chamar o **CreateCustomCAS** sem utilizar um arquivo de configuração, é necessário configurar algumas informações que indicará o comportamento da sinalização R2D em relação ao ramal CAS. Na tabela abaixo é mostrado os valores de **Command** e os limites de **Value**.

Command	Descrição
CASCFG_RING	Sinalização de RING
CASCFG_CALLER_HANGUP	Sinalização indicando que B desligou
CASCFG_IDLE	Sinalização indicando canal livre
CASCFG_ANSWER	Sinalização indicando que o equipamento atendeu uma chamada
CASCFG_PICKUP	Indicação que o fone foi "retirado do gancho" para discar
CASCFG_DROP_DELAY_BEFORE	Tempo de intervalo entre um Pickup e um Hangup
CASCFG_DROP	Comando para desligar
CASCFG_FLASH1_CMD	Comando de primeiro flash
CASCFG_FLASH1_DELAY	Pausa após o primeiro flash
CASCFG_FLASH2_CMD	Comando do segundo flash
CASCFG_FLASH2_DELAY	Pausa após o segundo flash

Parâmetros:

Port – Porta que será iniciado o controle

Command - Indica qual a informação será configurada de acordo com as constantes da tabela acima.

Value - Valor do dado indicado por **Command**

Valores de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso

DG_ERROR_DRIVER_CLOSED - O driver não foi iniciado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_FEATURE_NOT_SUPPORTED - Comando não suportado por este tipo de placa

DG_ERROR_THREAD_NOT_RUNNING - A thread E1 não foi inicializada ainda

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro

Command ou **Value** fora do especificado

5.2.11 ConnectBus (dg_ConnectBus)



Conecta recursos para uso do barramento H100

Declarações:

ActiveX:

```
SHORT ConnectBus(SHORT Card, SHORT  
ConnectionType, SHORT Param1,  
  
    SHORT Param2,  
  
    SHORT Param3)
```

API:

```
short dg_ConnectBus(unsigned short card,unsigned  
char conn_type,  
  
    unsigned char param1,  
  
    unsigned char param2,  
  
    unsigned char param3)
```

Descrição:

Esta função é utilizada para conectar portas entre si, através do barramento H100. A aplicação mais comum é a conexão para fins de comutação entre canais, importante função para sistemas de atendimento IVR.

Não precisa nunca ser chamada diretamente pois existe o grupo de funções **dg_h100_xxxx** que permitem um controle mais eficiente e fácil destas conexões. Maiores detalhes podem ser obtidos analisando o código do exemplo de **Comutação**.

Parâmetros:

Port – Indica a porta do tronco E1

Conn_Type – Indica

LOCAL_LOCAL - Conexão local entre canais e recursos DSP

LOCAL_H100 - Conexão dos canais para conexões do barramento H100

H100_LOCAL - Conexão do barramento H100 para os canais

H100_H100_CH - Canal do barramento a ser desabilitado permitindo conexão entre dois canais H100

H100_H100_SW - Link entre dois canais H100 através do barramento local

Param1 a Param3 - Estes parâmetros variam de acordo com Conn_Type, de acordo com a tabela abaixo

	__ LOCAL_LOCAL	
(128-191 DSP)	__ param1 - source (0-63 E1) or	
191 DSP)	__ param2 - target (0-63 E1) or (128-	
	__ LOCAL_H100	
(128-191 DSP)	__ param1 - *source* (0-63 E1) or	
31)	__ param2 - H100 frame *target* (0-	
127)	__ param3 - H100 slot *target* (0-	
	__ H100_LOCAL	
(128-191 DSP)	__ param1 - *target* (0-63 E1) or	
(0-31)	__ param2 - H100 frame *source*	

```

127)          |___ param3 - H100 slot *source* (0-
|___ H100_H100_CH
|___ param1 - canal local (0-255)
|___ H100_H100_SW
|___ param1 - H100 frame *source*
(0-31)
127)          |___ param2 - H100 slot *source* (0-
|___ param3 - H100 frame *target* (0-
31)
127)          |___ param4 - H100 slot *target* (0-

```

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
 DG_ERROR_DRIVER_CLOSED - Driver desabilitado
 DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido

5.2.12 CreateCallProgress (dg_CreateCallProgress)

API

ActiveX

Inicia o tratamento automático de supervisão de linha

Declarações:ActiveX:

```

SHORT CreateCallProgress(SHORT Port , LPCTSTR
ConfigFile);

```

API:

```
short dg_CreateCallProgress(short port, char  
*ConfigFile)
```

Descrição:

Este método cria a thread de controle de supervisão de linha (call progress), Todas as configurações pertinentes à supervisão de linha são lidas do arquivo informado no parâmetro **ConfigFile**. Caso este arquivo não seja fornecido, as configurações também podem ser modificadas pelo método **ConfigCallProgress**. Os arquivos de configuração do callprogress ficam na mesma pasta do firmware, sendo que o arquivo padrão é o **cp_default.cfg** que deverá atender a maioria das instalações. Este arquivo contém diversos comentários, explicando cada um dos parâmetros configuráveis.

Este método deve ser chamado antes do call progress em si ser habilitado pelo método **EnableCallProgress**. Como é uma thread com baixo consumo de processamento quando está atuando, não há maiores problemas em criá-la no início da aplicação e só destruí-la no seu término.

Parâmetros:

Port – Porta que será iniciado o controle

ConfigFile- Arquivo de configuração padrão, sem o caminho.

Valores de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso

DG_ERROR_DRIVER_CLOSED - O driver não foi iniciado

DG_ERROR_THREAD_ALREADY_RUNNING - A *thread* já havia sido iniciada anteriormente

DG_ERROR_COULD_NOT_CREATE_THREAD - Falha na inicialização da thread

5.2.13 CreateChatRoom (dg_CreateChatRoom)

API

ActiveX

E1 30/60

Cria recurso para conferencia ("cria sala de bate papo").

Declarações:

ActiveX:

```
LONG CreateChatRoom(SHORT Card, SHORT MaxPorts);
```

API:

```
long dg_CreateChatRoom(short card, short  
maxports);
```

Descrição:

A criação da conferência permite que um Número de portas determinado em *maxports* conversem entre si. Ao criar esse recurso o valor de retorno é utilizado como um código identificador (*handle*) para cada sala de conferência.

A conferência é criada para determinada placa. As portas que participarão desta conferência serão determinadas pela função **ChatAddPort**.

Atualmente a VoicerLib tem capacidade para 30 salas de conferência, mas os limites de portas por sala e capacidade de processamento das placas deverá ser considerado.

Parâmetros:

Card - Placa onde será criada a sala de conferência
MaxPorts – Maximo de canais permitido nesta sala de conferencia.

Valor de Retorno:

>= 0 é o Handle válido a ser usado nas outras funções
= 0 - Erro

5.2.14 CreateCustomCAS (dg_CreateCustomCAS)



Inicia a thread de controle para ramal CAS (E1)

Declarações:

ActiveX:

```
SHORT CreateCustomCAS(SHORT Port , LPCTSTR  
ConfigFile)
```

API:

```
short dg_CreateCustomCAS(short port, char  
*ConfigFile)
```

Descrição:

Este método cria a thread de controle de ramal CAS (E1). Todas as configurações pertinentes são lidas do arquivo informado no parâmetro **ConfigFile**. Caso este arquivo não seja fornecido, as configurações também podem ser modificadas pelo método **ConfigCustomCAS**. Os arquivos de configuração ficam na mesma pasta do firmware. Este arquivo contém diversos comentários, explicando cada um dos parâmetros configuráveis.

O arquivo de configuração contém a ação para cada sinal R2D que o PABX enviar ao ramal CAS.

Parâmetros:

Port – Porta que será iniciado o controle

ConfigFile- Arquivo de configuração padrão, sem o caminho.

Valores de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso

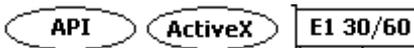
DG_ERROR_DRIVER_CLOSED - O driver não foi iniciado

DG_ERROR_THREAD_ALREADY_RUNNING - A *thread* já havia sido iniciada anteriormente

DG_ERROR_COULD_NOT_CREATE_THREAD - Falha na inicialização da thread

DG_ERROR_CONFIGFILE_NOT_FOUND - Arquivo informado não foi encontrado

5.2.15 CreateE1Thread (dg_CreateE1Thread)



Inicia o tratamento automático do protocolo R2D de troncos digitais

Declarações:

ActiveX:

```
SHORT CreateE1Thread(SHORT Port);
```

API:

```
short dg_CreateE1Thread(short port)
```

Descrição:

Antes de iniciar ou receber chamadas através do tronco E1 é necessário chamar esta função, que cria uma *thread* de controle sinalização de linha R2D e de registro MFC 5C. Desta forma o programador não necessita conhecer este protocolo para efetuar ou receber ligações em cada canal.

Parâmetros:

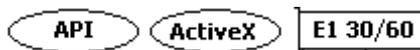
Port – Porta que será iniciado o controle

Valores de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso
DG_ERROR_DRIVER_CLOSED - O driver não foi iniciado
DG_ERROR_THREAD_ALREADY_RUNNING - A *thread* já havia sido iniciada anteriormente
DG_FEATURE_NOT_SUPPORTED - Comando enviado para uma placa FXO e não E1
DG_ERROR_FIFO_UNAVAILABLE - Erro na criação da estrutura de comunicação da thread
DG_ERROR_COULD_NOT_CREATE_THREAD - Falha na inicialização da thread

Veja Também: DestroyE1Thread, ConfigE1Thread

5.2.16 CreateLoggerControl



Inicia o tratamento automático de gravação em paralelo.

Declarações:

ActiveX:

```
SHORT CreateLoggerControl(SHORT Port, SHORT  
Type);
```

API:

```
short dg_CreateLoggerControl(short port, short  
type);
```

Descrição:

Esta função simplifica a criação de aplicações de gravação em paralelo, fazendo todo o tratamento do protocolo indicado pelo parâmetro **Type** automaticamente e gerando os eventos indicativos de início e fim da conversação.

Esta *thread* de controle também cuida de todas as chamadas as funções de conferência entre canais (CharAddPort, CreateChatRoom, etc...).

Parâmetros:

Port – Porta que será iniciado o controle

Type - Tipo de protocolo utilizado. Valores possíveis:

- **LOGGER_RD2MF** - Protocolo R2D (placas E1)

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro Type fora do intervalo permitido

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_FEATURE_NOT_SUPPORTED - Comando enviado para uma placa FXO e não E1

DG_ERROR_THREAD_ALREADY_RUNNING - A thread já está criada

DG_ERROR_FIFO_UNAVAILABLE - Erro na criação da estrutura de comunicação da thread

DG_ERROR_COULD_NOT_CREATE_THREAD - Falha na inicialização da thread

5.2.17 DefinePortResource (dg_DefinePortResource)

API

ActiveX

Associa um canal físico à uma porta virtual

Declarações:

ActiveX:

```
SHORT DefinePortResource(SHORT Port, SHORT  
Card, SHORT CardChannel);
```

API:

```
short dg_DefinePortResource(short port, short  
card, short card_channel)
```

Descrição:

Este método permite associar um canal físico de uma determinada placa à um número de porta lógica. Para isso, o método DefinePortResource deve ser chamado passando como parâmetro a porta lógica, a placa e o canal da placa.

Se por acaso já houver um outro canal físico associado à uma porta lógica e se tentar associar um novo canal, a configuração anterior será sobreposta.

O valor de porta virtual é o que deverá ser utilizado a partir daí para se referenciar qualquer coisa daquele canal físico.

Parâmetros:

Port – Indica o número da porta virtual

Card - Valor da placa (1 a n)

CardChannel - Valor da canal física da placa (depende do tipo de placa).

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.18 DestroyCallProgress (dg_DestroyCallProgress)

API

ActiveX

Finaliza o tratamento automático de supervisão de linha

Declarações:ActiveX:

```
SHORT DestroyCallProgress(SHORT Port);
```

API:

```
short dg_DestroyCallProgress(short port);
```

Descrição:

Esta função deverá ser chamada para finalizar uma *thread* de tratamento de supervisão de linha, sempre quando não for mais necessária

Parâmetros:

Port – Porta que será iniciado o controle

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.19 DestroyChatRoom (dg_DestroyChatRoom)

Remove recurso de uma sala de conferência

Declarações:ActiveX:

```
SHORT DestroyChatRoom(SHORT Card, LONG Handle);
```

API:

```
short dg_DestroyChatRoom(short card, unsigned  
long Handle)
```

Descrição:

Deve-se utilizar esse método para excluir uma conferência já existente, passando o identificado (*Handle*) que é código utilizado para identificar cada conferência.

Parâmetros:

Card - Placa onde será criada a sala de conferência

Handle - Obtido como retorno da função
CreateConferenceResource

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas

DG_ERROR_RESOURCE_UNAVAILABLE - Recurso não disponível

5.2.20 DestroyCustomCAS (dg_DestroyCustomCAS)

API

ActiveX

E1 30/60

Finaliza o tratamento automático de ramal CAS

Declarações:ActiveX:

```
SHORT DestroyCustomCAS(SHORT Port);
```

API:

```
short dg_DestroyCustomCAS(short port);
```

Descrição:

Esta função deverá ser chamada para finalizar uma *thread* de tratamento de ramal CAS, sempre quando não for mais necessária

Parâmetros:

Port – Porta que será iniciado o controle

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.21 DestroyLoggerControl



Finaliza o tratamento automático de gravação em paralelo.

Declarações:ActiveX:

```
SHORT DestroyLoggerControl(SHORT Port);
```

API:

```
short dg_DestroyLoggerControl(short port);
```

Descrição:

Esta função deverá ser chamada para finalizar uma *thread* de tratamento iniciada através da função CreateLoggerControl.

Parâmetros:

Port – Porta que será iniciado o controle

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_FEATURE_NOT_SUPPORTED - Comando enviado para uma placa FXO e não E1

5.2.22 DestroyE1Thread (dgDestroyE1Thread)



Termina o tratamento automático do protocolo R2D de troncos digitais

Declarações:

ActiveX:
`SHORT DestroyE1Thread(SHORT Port);`

API:
`short dg_DestroyE1Thread(short port)`

Descrição:

Quando a *thread* de controle não for mais necessária, é preciso removê-la para evitar desperdício de memória.

Parâmetros:

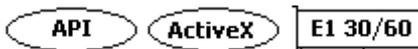
Port – Porta que será iniciado o controle

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_FEATURE_NOT_SUPPORTED - Comando enviado para uma placa FXO e não E1

Veja Também: CreateE1Thread, ConfigE1Thread

5.2.23 DisconnectBus (d_DisconnectBus)



Desconecta recursos para uso do barramento H100

Declarações:ActiveX:

```
SHORT DisconnectBus(SHORT Card, SHORT  
ConnectionType, SHORT Param1,
```

```
SHORT Param2,
```

```
SHORT Param3)
```

API:

```
short DisconnectBus(unsigned short card,unsigned  
char conn_type,
```

```
unsigned char param1,
```

```
unsigned char param2,  
unsigned char param3)
```

Descrição:

Este método é utilizado para desconectar portas que foram conectadas via barramento H100 com o método **ConnectBus**.

Não precisa nunca ser chamada diretamente pois existe o grupo de funções **dg_h100_xxxx** que permitem um controle mais eficiente e fácil destas conexões. Maiores detalhes podem ser obtidos analisando o código do exemplo de **Comutação**.

Parâmetros:

Port – Indica a porta do tronco E1

Conn_Type – Indica

LOCAL_LOCAL - Conexão local entre canais e recursos DSP

LOCAL_H100 - Conexão dos canais para conexões do barramento H100

H100_LOCAL - Conexão do barramento H100 para os canais

H100_H100_CH - Canal do barramento a ser desabilitado permitindo conexão entre dois canais H100

H100_H100_SW - Link entre dois canais H100 através do barramento local

Param1 a Param3 - Estes parâmetros variam de acordo com Conn_Type, de acordo com a tabela abaixo

LOCAL_LOCAL	_____ param1 - source (0-63 E1) or
(128-191 DSP)	_____ param2 - target (0-63 E1) or (128-
191 DSP)	_____ param1 - *source* (0-63 E1) or
LOCAL_H100	_____ param1 - *source* (0-63 E1) or
(128-191 DSP)	

31) |___ param2 - H100 frame *target* (0-
 127) |___ param3 - H100 slot *target* (0-
 |___ H100_LOCAL
 (128-191 DSP) |___ param1 - *target* (0-63 E1) or
 (0-31) |___ param2 - H100 frame *source*
 127) |___ param3 - H100 slot *source* (0-
 |___ H100_H100_CH
 |___ H100_H100_SW |___ param1 - canal local (0-255)
 (0-31) |___ param1 - H100 frame *source*
 127) |___ param2 - H100 slot *source* (0-
 31) |___ param3 - H100 frame *target* (0-
 127) |___ param4 - H100 slot *target* (0-

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
 DG_ERROR_DRIVER_CLOSED - Driver desabilitado
 DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido

5.2.24 Dial (dg_Dial)

API

ActiveX

Disca uma seqüência de números ou pausa.

Declarações:

ActiveX:

```
SHORT Dial(SHORT Port, LPCTSTR Number, LONG  
PauseAfterDial, SHORT DialType)
```

API:

```
short dg_Dial(short port, char *Number, long  
pause_after_dial, short dialtype);
```

Descrição:

O método Dial permite enviar dígitos ou pausas para que seja efetuada a discagem. Pode ser enviado os dígitos de "0" a "9", "#", "*" e os símbolos de pausa "vírgula" "ponto-e-vírgula" e "ponto".

A pausa para cada símbolo deve ser atribuída através do método **SetDialDelays**.

O parâmetro *DialType* aceita apenas o valor dtTone para discagens por tom.

Se a thread E1 estiver sendo usada, o dígito é enviado para ela, sendo gerado um MFF ou MFT, conforme o andamento do protocolo.

Parâmetros:

Port – Indica o canal da Placa

Number – String contendo os dígitos ou pausas para

discagem.

PauseAfterDial - É um tempo extra em milissegundos que é dado após a discagem. O evento OnAfterDial só será gerado após este tempo.

DialType - Pode assumir o valor dtTone, indicando se a discagem é por tom ou dtDirectMF que gera o MF mesmo com a thread E1 habilitada

Valor de Retorno:

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro Port fora do intervalo permitido

DG_ERROR_PARAM_OUTOFRANGE - Parametro PauseAfterDial maior que 60000 ms.

DG_EXIT_SUCCESS - Comando iniciado com sucesso

5.2.25 DisableAGC (dg_DisableAGC)



Desabilita o ajuste automático de ganho em um determinado canal

Declarações:

ActiveX:

```
SHORT DisableAGC(SHORT Port)
```

API:

```
short dg_DisableAgc(short port)
```

Descrição:

Este método desabilita o controle automático de ganho numa gravação ou conferência.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.26 DisableAnswerDetection (dg_DisableAnswerDetection)

APIActiveX

Desabilita a supervisão de atendimento.

Declarações:ActiveX:

```
SHORT EnableAnswerDetection(SHORT Port)
```

API:

```
short dg_EnableAnswerDetection(short port)
```

Descrição:

O método **DisableAnswerDetection** desabilita a supervisão de atendimento. A chamada desta função não tem efeito nas

placas digitais pois, devido à característica do protocolo utilizado, a detecção sempre está habilitada.

Nas placas FXO, o atendimento é detectado por áudio ou por timeout. Logo ao primeiro evento OnAnswerDetected (EV_ANSWERED) é preciso chamar este método pois, caso contrário, o evento continuará sendo gerado durante a conversação.

A VoicerLib 4 tem funções específicas para detecção de silêncio, não sendo mais necessário utilizar a detecção de atendimento para este fim.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_THREAD_NOT_RUNNING - Thread de controle de callprogress não está em funcionamento
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.27 DisableAutoFramers (dg_DisableAutoFramers)



Desabilita início automático dos framers E1

Declarações:

ActiveX:

```
BSTR dg_DisableAutoFramers(void);
```

API:

```
short dg_DisableAutoFramers(void);
```

Descrição:

Sempre quando for chamada a função de sincronismo SetCardSyncMode, a VoicerLib automaticamente reinicia os framers que fazem a comunicação E1. Caso seja necessário algum controle manual no funcionamento dos framers, esta função pode ser chamada para que, ao chamar a função SetCardSyncMode, os framers permanecerão desabilitados, sendo necessário chamar a função EnableFramer explicitamente.

Valores de Retorno:**API:**

DG_EXIT_SUCCESS - executado com sucesso

5.2.28 DisableCallProgress (dg_DisableCallProgress)

API

ActiveX

Desabilita a supervisão de linha, fax e ocupado.

Declarações:ActiveX:

```
SHORT DisableCallProgress(SHORT Port)
```

API:

```
short dg_DisableCallProgress(short port)
```

Descrição:

O método `DisableCallProgress` desabilita a supervisão de linha sem finalizar a thread de controle de callprogress. A supervisão de linha permite monitorar o sinal de chamada, ocupado, fax e tom de discagem.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso (API)
DG_ERROR_DRIVER_CLOSED - Driver Desabilitado
DG_ERROR_THREAD_NOT_RUNNING - A thread de callprogress não foi criada (pelo método `CreateCallProgress`).
DG_ERROR_PORT_OUT_OF_RANGE - Porta fora do intervalo permitido

5.2.29 DisabledDTMFFilter (`dg_DisableDTMFFilter`)

API

ActiveX

Desabilita filtro de DTMF e 425Hz

Declarações:

ActiveX:

```
SHORT DisableDTMFFilter(SHORT Port)
```

API:

```
short dg_DisableDTMFFilter(short port)
```

Descrição:

O método DisableDTMFFilter desliga as filtragens de DTMF de uma determinada porta do sistema.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.30 DisableDebug (dg_DisableDebug)

API

ActiveX

Desabilita o recurso de depuração da VoicerLib.

Declarações:ActiveX:

```
SHORT DisableDebug(VOID);
```

API:

```
short dg_DisableDebug(void);
```

Descrição:

Este método desabilita o envio de informações de debug via TCP/IP

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

5.2.31 DisableInputBuffer (dg_DisableInputBuffer)

API

ActiveX

Interrompe o envio de amostras da placa para a aplicação.

Declarações:ActiveX:

```
SHORT DisableInputBuffer(SHORT Port)
```

API:

```
short dg_DisableInputBuffer(short port)
```

Descrição:

O método **DisableInputBuffer** interrompe o envio de amostras da placa para a aplicação, iniciada pelo **EnableInputBuffer**.

Este método deve ser chamado sempre após o **StopRecordFile** caso haja uma gravação em curso.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.32 DisablePulseDetection (dg_DisablePulseDetection)

API

ActiveX

Desabilita a detecção de pulso.

Declarações:ActiveX:

```
SHORT DisablePulseDetection(SHORT Port)
```

API:

```
short dg_DisablePulseDetection(short port)
```

Descrição:

Esta função desabilita a detecção de pulso.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.33 DisableE1Thread (dg_DisableE1Thread)



Desabilita o tratamento automático do protocolo R2D de troncos digitais

Declarações:

ActiveX:
`SHORT DisableE1Thread(SHORT Port);`

API:
`short dg_DisableE1Thread(short port)`

Descrição:

O DisableE1Thread faz que a thread E1 pare de controlar o protocolo R2D de determinado canal, sem que a thread seja destruída.

Parâmetros:

Port – Porta que será iniciado o controle

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

Veja Também: CreateE1Thread, ConfigE1Thread, DestroyE1Thread, EnableE1Thread

5.2.34 DisableEchoCancelation (dg_DisableEchoCancelation)

API

ActiveX

Desabilita cancelamento de eco em recursos de conferência.

Declarações:

ActiveX:
SHORT DisableEchoCancelation(SHORT Port);

API:
short dg_DisableEchoCancelation(short port);

Descrição:

O método DisableEchoCancelation desabilita o cancelamento de eco nos recursos de conferência.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.35 DisableSilenceDetection (dg_DisableSilenceDetection)

API

ActiveX

Desabilita a detecção de silêncio

Declarações:

ActiveX:

```
SHORT DisableSilenceDetection(SHORT Port);
```

API:

```
short dg_DisableSilenceDetection(short port);
```

Descrição:

O método DisableSilenceDetection desabilita a detecção de silêncio da VoicerLib que foi habilitada pelo EnableSilenceDetection.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.36 EnableAnswerDetection (dg_EnableAnswerDetection)



Habilita a supervisão de atendimento.

Declarações:ActiveX:

```
SHORT EnableAnswerDetection(SHORT Port)
```

API:

```
short dg_EnableAnswerDetection(short port)
```

Descrição:

O método **EnableAnswerDetection** habilita a supervisão de atendimento. A supervisão de linha permite monitorar quando uma ligação for atendida. É necessário que a thread de controle de callprogress tenha sido iniciada (**CreateCallProgress**) e que o método **EnableCallProgress** também tenha sido chamado para que a detecção de atendimento possa funcionar.

A chamada desta função não tem efeito nas placas E1 pois, devido à característica do protocolo utilizado, a detecção sempre está habilitada.

Ao ser detectado o atendimento, o evento OnAnswerDetection (EV_ANSWERED) é gerado indicando o tipo de detecção (por áudio ou por timeout) e poderá ser tratado pela aplicação. Nas placas analógicas esta detecção é gerada a partir da presença de áudio na linha. Nas placas digitais, o atendimento é um dado, portanto, o evento é gerado no atendimento, independente da existência de áudio.

A VoicerLib 4 tem funções específicas para detecção de silêncio, não sendo mais necessário utilizar a detecção de atendimento para este fim.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

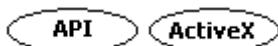
DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_THREAD_NOT_RUNNING - Thread de controle de callprogress não está em funcionamento

5.2.37 EnableAGC (dg_EnableAGC)



Habilita o ajuste automático de ganho em um determinado canal

Declarações:

ActiveX:`SHORT EnableAGC(SHORT Port)`API:`short dg_EnableAgc(short port)`**Descrição:**

O AGC (Automatic Gain Control) ou controle automático de ganho é muito útil quando existe uma diferença de ganho entre os interlocutores numa gravação ou conferência.

Ao habilitar este controle, o desenvolvedor pode melhorar substancialmente a qualidade da gravação ou a conversação em um conferência

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.38 EnableCallProgress (dg_EnableCallProgress)

API

ActiveX

Habilita a supervisão de linha, chamada, fax e ocupado para as

placas FXO

Declarações:

ActiveX:

```
SHORT EnableCallProgress(SHORT Port, SHORT  
CPTYPE)
```

API:

```
short dg_EnableCallProgress(short port, short  
cptype)
```

Descrição:

O método **EnableCallProgress** habilita a supervisão de linha. A supervisão de linha permite monitorar o tom de discagem, ocupado, chamada e fax.

Quando o sistema é iniciado, a supervisão está desabilitada como padrão, portanto é necessário criar uma thread de controle de callprogress para cada porta através do método **CreateCallProgress** e chamar o **EnableCallProgress** quando se quiser monitorar a linha.

O parâmetro CPTYPE indica o tipo de supervisão a ser iniciada. Consulte o "Guia do Programador" no tópico Supervisão de Linha para saber maiores detalhes de quando utilizar cada tipo de call progress.

Parâmetros:

Port – Indica o canal da Placa.

CPTYPE -

- **CP_ENABLE_GENERIC_TONE** - Desenvolvida para detectar a presença de um tom qualquer pré-configurado ou o atendimento de chamada, se esta facilidade estiver habilitada na VoicerLib pelo método **EnableAnswerDetection**.
-

- **CP_ENABLE_LINE_TONE_OR_BUSY** - Desenvolvida para a detecção rápida de tom de discar (tom de linha) ou ocupado antes do início de uma discagem ou após um Flash.
- **CP_ENABLE_BUSY_OR_FAX** - Desenvolvida para a detecção rápida de tom de ocupado ou sinal de FAX após o atendimento de uma chamada.
- **CP_ENABLE_ALL** - Desenvolvida para a detecção de tons de discar, de chamada, de ocupado, de FAX e atendimento entre uma discagem e o atendimento pelo assinante chamado.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro CPTtype fora do intervalo permitido
DG_ERROR_THREAD_NOT_RUNNING - Thread de controle não foi iniciada pelo **CreateCallProgress**.

5.2.39 EnableDebug (dg_EnableDebug)

API

ActiveX

Habilita o recurso de depuração da VoicerLib em ambiente Windows.

Declarações:

ActiveX:

```
SHORT EnableDebug(SHORT UdpPort);
```

API:

```
short dg_EnableDebug(short UdpPort);
```

Descrição:

A VoicerLib tem um recurso de depuração que permite que seja enviado mensagens via TCP/IP (UDP) de todos os comandos de baixo nível enviados para ou recebidos da placa. Para isso é utilizado o recurso de *broadcast* do protocolo UDP, permitindo que as mensagens sejam monitoradas de outra aplicação, mesmo que seja em um computador remoto.

Este recurso é útil para fins de localização de problemas, porém não deve ser utilizado em ambientes de produção que não necessitem de depuração pois consome recursos de CPU.

Parâmetros:

UdpPort - Informa qual a porta UDP deverá ser monitorada pela aplicação cliente

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

5.2.40 EnableDTMFFilter (dg_EnableDTMFFilter)

APIActiveX

Habilita filtro de DTMF

Declarações:

ActiveX:

```
SHORT EnableDTMFFilter(SHORT Port);
```

API:

```
short dg_EnableDTMFFilter(short port)
```

Descrição:

O método EnableDTMFFilter habilita a filtragem de tons DTMF e 425Hz. Esta filtragem é necessária principalmente em situações de conferência para evitar que os dígitos detectados por uma porta sejam também detectados por outra na mesma conferência. O mesmo vale para o tom 425hz pois o tom de ocupado também não pode se propagar em todos os canais de conferência.

Em uma aplicação de conferência típica, vários canais estão compartilhando o mesmo recurso e tudo que se ouve ou se fala em um canal é propagado pelos outros. Muitas vezes é necessário que o usuário interaja com o sistema através de dígitos do telefone. Para permitir isso sem que exista interferência dos outros canais, essa filtragem se torna necessária.

Importante: Este recurso não permite a utilização simultânea da gravação em formato GSM.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.41 EnableE1Thread (dg_EnableE1Thread)

API

ActiveX

E1 30/60

Habilita o tratamento automático do protocolo R2D de troncos digitais

Declarações:

ActiveX:

```
SHORT EnableE1Thread(SHORT Port);
```

API:

```
short dg_EnableE1Thread(short port)
```

Descrição:

Após o uso do DisableE1Thread é necessário chamar o EnableE1Thread para que a thread de controle E1 possa voltar a gerenciar o protocolo R2D.

Parâmetros:

Port – Porta que será iniciado o controle

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

Veja Também: CreateE1Thread, ConfigE1Thread, DestroyE1Thread, DisableE1Thread

5.2.42 EnableEchoCancelation (dg_EnableEchoCancelation)

API

ActiveX

Habilita cancelamento de eco em recursos de conferência.

Declarações:

ActiveX:

```
SHORT EnableEchoCancelation(SHORT Port);
```

API:

```
short dg_EnableEchoCancelation(short port);
```

Descrição:

O método EnableEchoCancelation habilita o cancelamento de eco nos recursos de conferência, evitando a degradação do sinal causada pelo efeito de eco no áudio.

ATENÇÃO: Nas placas E1, o cancelamento de eco só pode ser utilizado nas placas com 30 canais. Nas placas com 60 canais, o método retorna DG_FEATURE_NOT_SUPPORTED e o cancelamento de eco não é ativado. Pode-se forçar uma placa de 60 canais funcionar apenas com 30 e suportar cancelamento de eco chamando-se a função ForceSingleSpan antes de se iniciar a VoicerLib.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_FEATURE_NOT_SUPPORTED - Comando não suportado por este tipo de placa

5.2.43 EnableInputBuffer (dg_EnableInputBuffer)

API

ActiveX

Habilita o envio de amostras da placa para a aplicação.

Declarações:ActiveX:

```
SHORT EnableInputBuffer(SHORT Port, SHORT  
Agc_Enable)
```

API:

```
short dg_EnableInputBuffer(short port, short  
agc_enable)
```

Descrição:

O método **EnableInputBuffer** inicia o envio de amostras da placa para a VoicerLib. Com isso é possível iniciar uma gravação em arquivo ou manipular as amostras diretamente na aplicação ou ainda ambas as situações simultaneamente.

O que determina o que será feito com as amostras é a chamada do método **RecordFile** para gravá-las ou associando uma função callback com o método **SetAudioInputCallback** para tratá-la diretamente na aplicação (VOIP, por exemplo).

Recomenda-se chamar o `EnableInputBuffer` somente no início da aplicação e a cada gravação, utilizar o `PauseInputBuffer` quando necessário.

O parâmetro **agc_enable** permite habilitar o controle automático de ganho que melhora a qualidade da gravação caso a diferença de áudio entre os interlocutores seja muito grande.

Parâmetros:

Port – Indica o canal da Placa

Agc_Enable - Habilita/desabilita controle de ganho. Pode receber `DG_ENABLE` ou `DG_DISABLE`.

Valor de Retorno:

`DG_EXIT_SUCCESS` - executado com sucesso

`DG_ERROR_DRIVER_CLOSED` - Driver desabilitado

`DG_ERROR_PORT_OUT_OF_RANGE` - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

`DG_ERROR_PARAM_OUTOFRANGE` - `Agc_enable` diferente dos valores permitidos

`DG_ERROR_THREAD_ALREADY_RUNNING` - A thread de envio de amostras da placa já está em execução

5.2.44 EnablePulseDetection (dg_EnablePulseDetection)

API

ActiveX

Habilita a detecção de pulso.

Declarações:

ActiveX:

```
SHORT EnablePulseDetection(SHORT Port, SHORT  
Sensibility)
```

API:

```
short dg_EnablePulseDetection(short port, short  
sensibility)
```

Descrição:

Esta função habilita a detecção de pulso. Só é possível detectar pulso com precisão durante o silêncio.

O parâmetro sensibilidade permite alterar a sensibilidade de detecção do pulso, variando de -42dB até +12dB mas aconselha-se sempre passar zero, e variar o valor apenas caso haja algum problema na detecção de pulso

Ao habilitar a detecção de pulso, os eventuais dígitos detectados são tratados da mesma forma que na detecção de MFs, ou seja, através do evento OnDigitDetected ou OnDigitsReceived (EV_DTMF ou EV_DIGITSRECEIVED).

Parâmetros:

Port – Indica o canal da Placa

Sensibility - Sensibilidade de detecção variando de -42dB até +12dB

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido

5.2.45 EnableSilenceDetection (dg_EnableSilenceDetection)

API

ActiveX

Habilita a detecção de silêncio.

Declarações:ActiveX:

```
SHORT EnableSilenceDetection(SHORT Port, SHORT  
silence_time, SHORT audio_time)
```

API:

```
short dg_EnableSilenceDetection(short port, short  
silence_time, short audio_time)
```

Descrição:

Esta função habilita a detecção de silêncio. A VoicerLib detectará silêncio se perceber a ausência de áudio por **silence_time** milissegundos. O parâmetro **audio_time** permite modificar a sensibilidade com que a VoicerLib detectará áudio (inverso de silêncio) e é representado em milissegundos também. O padrão é passar zero no audio_time para que ao primeiro sinal de áudio, a VoicerLib já gere o evento indicando fim do silêncio.

O evento OnSilenceDetected (EV_SILENCE) ocorre quando se detecta o silêncio e quando se detecta o fim do silêncio. O parâmetro recebido SignalCode indica se foi detectado o silêncio recebendo o valor DG_SILENCE_DETECTED (1) ou fim de silêncio recebendo o valor DG_AUDIO_DETECTED (0).

Parâmetros:

Port – Indica o canal da Placa

silence_time - Valor mínimo em milisegundos para se considerar silêncio

audio_time - Valor mínimo em milisegundos para se considerar presença de áudio ou zero para considerar no primeiro sinal de áudio (recomendado)

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.46 Flash (dg_Flash)



Executa um comando de flash para a central PABX.

Declarações:

ActiveX:

```
SHORT Flash(SHORT Port, SHORT MSec, SHORT  
PauseAfterFlash)
```

API:

```
short dg_Flash(short port, long msec, long  
pauseafterflash);
```

Descrição:

Com a placa FXO, as centrais PABX sempre necessitam do flash (desligar e ligar) para poder efetuar uma transferência ou outra função qualquer. O método Flash permite que seja encaminhado para a placa um comando flash com o tempo em milissegundos especificado e também um tempo de pausa após o flash. Esta pausa é útil em algumas centrais que demoram para *comutar* os ramais.

Nas placas VB6060PCI (E1), o flash se comporta de maneira diferente e depende de correta configuração de ramais CAS (veja em CreateCustomCAS). Nestes casos, ao executar a função de flash, o sistema enviará comandos R2 ao PABX. Se não estiver configurado as threads de controle CAS, esta função não tem efeito nenhum.

Parâmetros:

Port – Indica o canal da Placa

Milisseconds – Indica o tempo em milissegundos para o Flash (consulte a documentação do PABX para maiores detalhes)

PauseAfterFlash - Indica a pausa após flash em milissegundos.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas

instaladas

DG_EXIT_FAILURE - Falha no envio do comando para a placa

DG_ERROR_COULD_NOT_CREATE_THREAD - Não foi possível iniciar a thread para executar esta operação.

5.2.47 ForceSingleSpan (dg_ForceSingleSpan)

API

ActiveX

E1 30/60

Força uma placa de 60 canais a ser de 30 canais

Declarações:

ActiveX:

```
BSTR ForceSingleSpan(VOID);
```

API:

```
short dg_ForceSingleSpan(void);
```

Descrição:

Este método deve ser chamado somente no caso de se ter placas de 60 canais e se querer utilizar o cancelamento de eco nos primeiros 30 canais. É obrigatório chamá-lo **antes** de se iniciar a VoicerLib.

Valores de Retorno:

API:

DG_EXIT_SUCCESS - executado com sucesso

5.2.48 GetE1Number (dg_GetE1Number)

API

ActiveX

E1 30/60

Lê o número recebido durante a troca de sinalização R2D em uma chamada entrante.

Declarações:

ActiveX:

```
BSTR GetE1Number(SHORT Port);
```

API:

```
short dg_GetE1Number(short port, char  
*szNumber);
```

Descrição:

Este método retorna uma *string* contendo o número recebido durante a troca de sinalização. Deve ser chamado quando chegar o evento OnE1StateChange com o *status* C_NUMBER_RECEIVED.

Este número pode chegar em partes, dependendo de como está configurado o recebimento da identificação de A através do método **ConfigE1Thread**. Se a thread for configurada para receber a identificação de A a partir do segundo dígito recebido, serão gerados dois eventos OnE1StateChange com o *status* C_NUMBER_RECEIVED: o primeiro receberá os dois primeiros dígitos e o segundo evento receberá o resto. A aplicação final deverá tratar isso.

Se a *thread* de logger estiver sendo utilizada ao invés da *thread E1*, este método poderá ser chamado quando o evento EV_LOGGEREVENT com status LOGGER_LINEREADY (Início da ligação).

Parâmetros:

Port – Porta onde deve ser lida a sinalização.

szNumber (API)– *null terminated string* que receberá o número.

Valores de Retorno:**ActiveX:**

Retorna uma string com os dígitos ou nulo caso não haja número disponível

API:

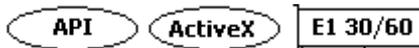
DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_THREAD_NOT_RUNNING - Thread de Logger ou E1 não está em execução

5.2.49 GetLoggerCallType (dg_GetLoggerCallType)



Lê o tipo da ligação em determinado canal de *logger*.

Declarações:ActiveX:

```
SHORT GetLoggerCallType(void);
```

API:

```
short dg_GetLoggerCallType(void);
```

Descrição:

Esta função indica se uma ligação monitorada pela *thread* de *logger* é entrante ou saínte. O sentido da ligação dependerá da conexão física dos cabos. Consulte o capítulo sobre gravação em paralelo para maiores detalhes.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

```
INCOMINGCALL = 1  
OUTGOINGCALL = 2
```

5.2.50 GenerateMF (dg_GenerateMF)

API

ActiveX

Gerador de sinais multifreqüenciais

Declarações:ActiveX:

```
SHORT GenerateMF(SHORT Port, SHORT TypeMF, SHORT  
Dig)
```

API:

```
short dg_GenerateMF(short port, short type, char  
cDig);
```

Descrição:

Esta função inicia a geração de sinais multifreqüenciais conforme indicado no parâmetro TypeMF. Qualquer valor maior que zero inicia a geração do MF e ele só será suspenso quando for chamado o valor GENERATE_OFF.

Parâmetros:

Port – Indica o canal da Placa

TypeMF – Tipo do MF a ser gerado:

GENERATE_OFF = 0

GENERATE_DTMF = 1

GENERATE_MFT = 2

GENERATE_MFF = 3

GENERATE_MF = 4

GENERATE_TONE1 = 5

GENERATE_TONE2 = 6

Dig – Dígito a ser gerado (1 a 15)

Valor de Retorno:

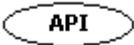
DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido. (Type ou Dig diferente dos valores acima)

5.2.51 GetAbsolutePortNumber

APIActiveX

Retorna o número absoluto da porta

Declarações:ActiveX:

```
SHORT GetAbsolutePortNumber(SHORT Card,  
                             SHORT RelativePort)
```

API:

```
short dg_GetAbsolutePortNumber(short card,  
                               short  
relativeport);
```

Descrição:

A porta absoluta é a numeração de **1 a n**, dependendo dos tipos e da quantidade de placas instaladas. Deve ser passado o número da placa e a porta relativa. Ex.: Placa 2, porta relativa 3 equivale a porta absoluta 63 em um sistema com placas digitais de 60 canais.

Parâmetros:

Card - Placa ≥ 1

RelativePort – Indica a valor da porta relativa aquela placa

Valor de Retorno:

Número de porta absoluta ≥ 1

5.2.52 GetAlarmStatus (dg_GetAlarmStatus)



Solicita estado dos alarmes

Declarações:ActiveX:

```
SHORT GetAlarmStatus(SHORT Card)
```

API:

```
short dg_GetAlarmStatus(short card)
```

Descrição:

Os alarmes dos E1s das placas digitais são gerados automaticamente pela placa quando ocorrem. Esta função permite solicitar *manualmente* o último estado dos alarmes. O evento EV_E1ALARM será gerado indicando o código dos alarmes.

Deve ser passado a placa que se deseja ter o estado.

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro CARD fora do intervalo permitido, excedendo o número de placas instaladas

5.2.53 GetCallerID (dg_GetCallerID)

API

ActiveX

Recupera o número de A quando utilizado as threads de controle E1 e Logger.

Declarações:

ActiveX:

```
BSTR GetCallerID(SHORT Port);
```

API:

```
short dg_GetCallerId(short port, char  
*szCallerID)
```

Descrição:

Nas *threads* de controle E1 e Logger, o protocolo R2D é tratado automaticamente. Para recuperar o número de quem chamou (CallerID) é necessário chamar esta função. Os eventos pertinentes às *threads* de controle sinalizarão quando a informação estiver disponível.

Nas placas FXO, quando utilizada o método IdleStart, o GetCallerID pode ser chamado no evento OnCallerID para se obter o número identificado.

Parâmetros:

Port – Porta onde deve ser lida a sinalização.

szCallerID (API)– *null terminated string* que receberá o número.

Valores de Retorno:

ActiveX:

Retorna uma string com os dígitos ou nulo caso não haja número disponível

API:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.54 GetCardBus (dg_GetCardBus)

API

ActiveX

Recupera bus pci da placa

Declarações:ActiveX:

```
LONG GetCardBus(SHORT Card)
```

API:

```
unsigned int dg_GetCardBus(short card)
```

Descrição:

Esta função devolve o bus do barramento PCI da motherboard onde a placa está instalada.

Valores de Retorno:

Valor relativo ao bus do barramento PCI

5.2.55 GetCardInterface (dg_GetCardInterface)

API

ActiveX

Recupera tipo de placa.

Declarações:ActiveX:

```
LONG GetCardInterface(SHORT Card)
```

API:

```
short dg_GetCardInterface(short card)
```

Descrição:

Esta função devolve o tipo de placa, indicando se é placa FXO ou placa digital. A função GetCardType devolve o modelo da placa, mas como existe mais de um modelo de placa E1, a utilização do GetCardInterface facilita a tarefa de saber se a placa é digital ou FXO, independente do código do modelo.

Valores de Retorno:

DG_DIGITAL_INTERFACE (1)
DG_FXO_INTERFACE (2)
DG_UNKNOWN_INTERFACE (99)

5.2.56 GetPortInterface (dg_GetPortInterface)

API

ActiveX

Recupera tipo de placa de uma determinada porta.

Declarações:

ActiveX:

```
LONG GetCardInterface(SHORT Port)
```

API:

```
short dg_GetCardInterface(short port)
```

Descrição:

Esta função devolve o tipo de placa de uma determinada porta, indicando se é placa FXO ou placa digital. A função GetCardType devolve o modelo da placa, mas como existe mais de um modelo de placa E1, a utilização do GetCardInterface facilita a tarefa de saber se a placa é digital ou FXO, independente do código do modelo.

Valores de Retorno:

DG_DIGITAL_INTERFACE (1)

DG_FXO_INTERFACE (2)

DG_UNKNOWN_INTERFACE (99)

5.2.57 GetCardType (dg_GetCardType)

API

ActiveX

Recupera o tipo da placa

Declarações:ActiveX:

```
SHORT GetCardType(SHORT Card)
```

API:

```
short dg_GetCardType(short card)
```

Descrição:

Esta função devolve o código do modelo da placa (1 a n)

Valores de Retorno:

VBE13060PCI - Placa E1 3060 PCI (Full)

VB0408PCI - Placa FXO de 4/8 canais

VBE13060PCI_R - Placa E1 3060PCI (Full R2)

VBE16060PCI - Placa E1 Slim mod. VB6060PCI

VBE16060PCI_R - Placa E1 Slim mod. VB6060PCI R2

5.2.58 GetCardSlot (dg_GetCardSlot)

APIActiveX

Recupera slot pci da placa

Declarações:ActiveX:

```
LONG GetCardSlot(SHORT Card)
```

API:

```
unsigned int dg_GetCardSlot(short card)
```

Descrição:

Esta função devolve o slot do barramento PCI da motherboard onde a placa está instalada.

Valores de Retorno:

Valor relativo ao slot do barramento PCI ou zero no caso de erro

5.2.59 GetCardNumber (dg_GetCardNumber)

API

ActiveX

Retorna a placa de um determinado canal absoluto

Declarações:ActiveX:

```
SHORT GetCardNumber(SHORT AbsolutePort)
```

API:

```
short dg_GetCardNumber(short absoluteport);
```

Descrição:

Retorna o número da placa de acordo com o canal absoluto passado no parâmetro. Por exemplo, o canal 61 pode ser o canal relativo 1 da segunda placa de 60 canais.

Parâmetros:

AbsolutePort – Indica a valor da porta

Valor de Retorno:

Número da placa ≥ 1

5.2.60 GetCardPortsCount (dg_GetCardPortsCount)

API

ActiveX

Lê a quantidade de portas de uma determinada placa

Declarações:ActiveX:

```
SHORT GetCardPortsCount (SHORT Card);
```

API:

```
short dg_GetCardPortsCount (short card);
```

Descrição:

Retorna o número de portas de uma determinada placa.

Parâmetros:

Card – Indica a placa que se deseja saber a quantidade de canais

Valor de Retorno:

Número de portas instaladas, ou zero para nenhuma.

5.2.61 GetCardsCount (dg_GetCardsCount)

API

ActiveX

Lê a quantidade de placas disponíveis.

Declarações:

ActiveX:
SHORT GetCardsCount(void);

API:
short dg_GetCardsCount(void);

Descrição:

A função **GetCardsCount** retorna o número de placas instaladas no sistema.

Valor de Retorno:

Número de placas instaladas, ou zero para nenhuma.

5.2.62 GetDigits (dg_GetDigits)

API

ActiveX

Inicia a espera de uma seqüência de dígitos.

Declarações:

ActiveX:

```
SHORT GetDigits(SHORT Port, SHORT MaxDigits,  
LPCTSTR TermDigits,  
LONG DigitsTimeOut,  
LONG InterDigitsTimeOut)
```

API:

```
short dg_GetDigits(short port, short maxdigits,  
char *termdigits, long digitstimeout,  
long interdigitstimeout);
```

Descrição:

O método **GetDigits** permite iniciar a espera de um conjunto de dígitos, por um determinado tempo ou até receber um dígito finalizador.

Como a VoicerLib tem um processamento assíncrono, após a execução de **GetDigits**, é necessário tratar o resultado no evento **OnDigitsReceived**, o que pode acontecer segundos mais tarde. Para recuperar os dígitos detectados deve-se utilizar o método **ReadDigits**.

Ao executar o **GetDigits**, o programa segue seu fluxo normal, isto é, não fica esperando a execução do **GetDigits** até o fim.

Para interromper a execução do **GetDigits**, deve ser chamado o **CancelGetDigits**. Esta é uma prática importante ao terminar uma ligação.

Caso seja necessário, o método **ClearDigits** deve ser chamado para apagar o buffer de dígitos, antes da execução do **GetDigits**.

Parâmetros:

Port – Indica o canal da Placa

MaxDigits – Número máximo de dígitos permitido. Utilize esta propriedade para limitar o número de dígitos que o

usuário poderá teclar.

TermDigits – É uma string contendo um ou mais dígitos, que ao serem detectados finaliza a execução do GetDigits e gera o evento OnDigitsReceived. Se qualquer dígito puder interromper utilize o símbolo "@" ao invés de colocar "0123456789#*", apesar de a segunda forma também estar correta. Se não houver dígito finalizador, passar "" (vazio). É importante ressaltar que este parâmetro é uma string terminada em nulo.

DigitsTimeOut – Refere-se ao tempo máximo de espera pelo primeiro dígito programado no GetDigits. Caso seja detectado o primeiro dígito, este timeout não ocorrerá mais.

InterDigitsTimeOut - É o tempo máximo que o GetDigits esperará de intervalo entre cada dígito. Após este tempo, será gerado o evento OnDigitsReceived como código correspondente ao time-out interdígito. Em milissegundos.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.63 GetDriverEnabled (dg_GetDriverEnabled)



Indica se a VoicerLib está inicializada.

Declarações:ActiveX:

```
VARIANT_BOOL GetDriverEnabled(void)
```

API:

```
short dg_DriverEnabled(void)
```

Descrição:

Esta função permite saber se a VoicerLib já foi inicializada através da função **StartVoicerLib (dg_StartVoicerLib)**.

Valores de Retorno:

Na versão ActiveX o retorno é do tipo booleano (true ou false). Já a versão API tem como retorno zero (false) e 1 (true).

5.2.64 GetE1ThreadStatus (dg_GetE1ThreadStatus)

API

ActiveX

E1 30/60

Retorna o *status* da thread E1 da porta especificada.

Declarações:ActiveX:

```
SHORT GetE1ThreadStatus(short port);
```

API:

```
short dg_GetE1ThreadStatus(short port);
```

Descrição:

Esta função indica se a thread E1 está habilitada ou desabilitada.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_E1_THREAD_ENABLE = 1
DG_E1_THREAD_DISABLE = 0

5.2.65 GetPortsCount (dg_GetPortsCount)

API

ActiveX

Lê a quantidade de portas disponíveis.

Declarações:ActiveX:

```
SHORT GetPortsCount(void);
```

API:

```
short dg_GetPortsCount(void);
```

Descrição:

A função **GetPortsCount** retorna o número de portas instaladas no sistema, somatória de todas as portas de todas as placas.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Número de portas instaladas, ou zero para nenhuma.

5.2.66 GetPortStatus (dg_GetPortStatus)

API

ActiveX

Recupera o status da porta especificada.

Declarações:

ActiveX:

```
SHORT GetPortStatus(SHORT Port);
```

API:

```
short dg_GetPortStatus(void);
```

Descrição:

Este método permite saber o que o canal especificado está executando em um determinado momento.

Parâmetros:

Port – Indica o canal da Placa

Valores de Retorno:

- spFlashing -Executando um Flash
- spDialing -Discando
- spNone -Ocioso
- spWaitingDigits -Esperando Dígitos
- spOffHook - Indicando "fora do gancho" quando utilizada a *thread E1*.

5.2.67 GetPortCardType (dg_GetPortCardType)

API

ActiveX

Recupera o tipo da placa através do número da porta

Declarações:

ActiveX:

```
SHORT GetPortCardType(SHORT Port)
```

API:

```
short dg_GetPortCardType(short port)
```

Descrição:

Esta função devolve o código do modelo da placa de uma determinada porta. Com isso é possível saber que tipo de placa tem determinada porta.

Valores de Retorno:

VBE13060PCI - Placa E1 3060 PCI (Full)

VB0408PCI - Placa FXO de 4/8 canais

VBE13060PCI_R - Placa E1 3060PCI (Full R2)

VBE16060PCI - Placa E1 Slim mod. VB6060PCI

VBE16060PCI_R - Placa E1 Slim mod. VB6060PCI R2

5.2.68 GetPlayFormat (dg_GetPlayFormat)

API

ActiveX

Recupera o formato de reprodução de uma porta.

Declarações:

ActiveX:

```
SHORT GetPlayFormat(SHORT Port)
```

API:

```
short dg_GetPlayFormat(short port)
```

Descrição:

O método GetPlayFormat retorna o formato de reprodução configurado para determinada porta.

Parâmetros:

Port – Indica o canal da Placa.

Valores de Retorno:

- 0 - ffWaveULaw (Lei *m*)
- 1 - ffSig
- 2 - ffWavePCM
- 3 - ffGsm610
- 4 - ffWaveALaw (Lei *A*) - **Suportado somente na VoicerBox E1 30/60**

5.2.69 GetRecordFormat (dg_GetRecordFormat)

API

ActiveX

Recupera o formato de gravação de uma porta.

Declarações:ActiveX:`SHORT GetRecordFormat (SHORT Port)`API:`short dg_GetRecordFormat (short port)`**Descrição:**

O método GetRecordFormat retorna o formato de gravação configurado para determinada porta.

Parâmetros:

Port – Indica o canal da Placa.

Valores de Retorno:

0 - ffWaveULaw (Lei *m*)

1 - ffSig

2 - ffWavePCM

3 - ffGsm610

4 - ffWaveALaw (Lei A) - ***Suportado somente na VoicerBox E1 30/60***

5.2.70 GetRelativeChannelNumber

APIActiveX

Converte o número de porta absoluto para o número relativo à placa

Declarações:ActiveX:

```
SHORT GetRelativeChannelNumber ( SHORT  
AbsolutePort )
```

API:

```
short dg_GetRelativeChannelNumber ( short  
absoluteport );
```

Descrição:

Converte o número de porta absoluto para o número relativo à placa. Por exemplo, o canal 61 pode ser o canal relativo 1 da segunda placa de 60 canais.

Parâmetros:

AbsolutePort – Indica a valor da porta

Valor de Retorno:

Número de porta relativa à placa >= 1

5.2.71 GetVersion (dg_GetVersion)

API

ActiveX

Pede o número de versão do firmware

Declarações:ActiveX:

```
SHORT GetVersion ( void )
```

API:

```
short dg_GetVersion(void);
```

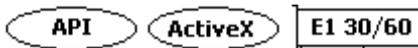
Descrição:

Este método permite saber qual a versão do firmware da placa. Pode ser útil para rastreamento de atualizações

Valores de Retorno:

Número da Versão

5.2.72 h100_AllocPort (dg_h100_AllocPort)



Aloca uma porta no barramento H.100

Declarações:ActiveX:

```
SHORT h100_AllocPort(SHORT Port);
```

API:

```
short dg_h100_AllocPort(short port);
```

Descrição:

Esta função não precisa ser chamada diretamente pelo programador, já que as funções de conexão o fazem automaticamente, quando necessário.

Em um primeiro momento, é necessário alocar um recurso do

barramento H100 para uma determinada porta. No barramento deverá haver um **frame/slot** disponível para uma conexão full-duplex de duas portas. Isso é feito pela função **h100_AllocPort** (**h100_AllocPort**) passando a porta como parâmetro. Neste momento, o recurso do barramento foi alocado, porém nenhuma conexão ainda foi feita.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.73 h100_DisconnectAll (dg_h100_DisconnectAll)



Desconecta todas as portas de uma placa do barramento H.100

Declarações:

ActiveX:
`SHORT h100_DisconnectAll(SHORT Card);`

API:
`short dg_h100_DisconnectAll(short card);`

Descrição:

Esta função desconecta todas as portas da placa determinada do barramento H100. Ao ter as portas desconectadas, será possível criar novas conexões manualmente.

Para utilizar as outras funções h100_xxx **não** é necessário chamar esta função antes.

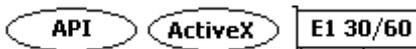
Parâmetros:

Card – Indica a placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.74 h100_EnablePort (dg_h100_EnablePort)



Habilita uma porta com conexão padrão no recurso de DSP

Declarações:ActiveX:

```
SHORT h100_EnablePort(SHORT Port)
```

API:

```
short dg_h100_EnablePort(short port)
```

Descrição:

Antes de utilizar qualquer porta no dispositivo H100, é necessário habilitá-la no barramento através dessa função.

Em uma situação padrão, as portas estão sempre conectadas à recursos de DSP da placa, permitindo funções como: reprodução de mensagens, reconhecimento de dígitos, etc...

Na prática, ao habilitar uma porta no barramento não modificará seu comportamento porém, internamente, o canal passa a estar conectado no barramento e dali, conectado ao recurso de DSP da placa.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.75 h100_HalfDuplexDisconnect



Efetua uma desconexão unidirecional entre duas portas

Declarações:

ActiveX:
`SHORT h100_HalfDuplexDisconnect(SHORT PortSource,`

```
SHORT PortTarget);
```

API:

```
short dg_h100_HalfDuplexDisconnect(short  
port_source, short port_target);
```

Descrição:

Esta função desconecta Port do barramento H.100, devolvendo o seu controle para um recurso DSP. É necessário passar a porta desejada e a porta a qual esta está conectada. O controle de saber que porta está conectada aonde é responsabilidade da aplicação final.

Parâmetros:

PortSource – Indica a porta a ser desconectada

PortTarget – Indica a porta a qual PortSource está conectada

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.76 h100_HalfDuplexConnect (dg_h100_HalfDuplexConnect)

API

ActiveX

E1 30/60

Efetua uma conexão unidirecional entre duas portas

Declarações:ActiveX:

```
SHORT dg_h100_HalfDuplexConnect(SHORT PortSource,  
SHORT PortTarget);
```

API:

```
short dg_h100_HalfDuplexConnect(short  
port_source, short port_target);
```

Descrição:

Ao chamar esta função, a porta *PortSource* é conectada em *PortTarget* em um único sentido, ou seja, o áudio de *PortSource* será enviado para *PortTarget* mas o inverso não.

Parâmetros:

PortSource – Indica a porta de origem

PortTarget – Indica a porta de destino

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.77 h100_FullDuplexConnect (dg_h100_FullDuplexConnect)

API

ActiveX

E1 30/60

Efetua uma conexão bi-direcional entre duas portas

Declarações:

ActiveX:

```
SHORT h100_FullDuplexConnect(SHORT Port1, SHORT  
Port2);
```

API:

```
short dg_h100_FullDuplexConnect(short port1,  
short port2);
```

Descrição:

Ao chamar esta função, a porta *Port1* é conectada em *Port2* e vice-versa, permitindo que o áudio seja enviado nos dois sentidos. Esta é a forma indicada para que duas portas possam conversar normalmente (falar e ouvir).

Parâmetros:

Port1 – Indica a primeira porta

Port2 – Indica a segunda porta

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.78 h100_FullDuplexDisconnect

API

ActiveX

E1 30/60

Efetua uma desconexão bi-direcional entre duas portas

Declarações:

ActiveX:

```
SHORT h100_FullDuplexDisconnect(SHORT Port1,  
SHORT Port2);
```

API:

```
short dg_h100_FullDuplexDisconnect(short port1,  
short port2);
```

Descrição:

Esta função desconecta Port1 e Port2 do barramento H.100, devolvendo o seu controle para um recurso DSP.

Parâmetros:

Port1 – Indica a primeira porta

Port2 – Indica a segunda porta

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.79 h100_GetFreePortByRange

API

ActiveX

E1 30/60

Retorna uma porta livre dentro de um intervalo

Declarações:

ActiveX:

```
SHORT h100_GetFreePortByRange(SHORT PortStart,  
SHORT PortEnd);
```

API:

```
short dg_h100_GetFreePortByRange(short  
port_start, short port_end);
```

Descrição:

Uma função muito útil ao se utilizar os recursos de conexão do H100 é a **h100_GetFreePortByRange** (**dg_h100_GetFreePortByRange**). Ela serve para que o programador saiba qual porta, dentro de um determinado intervalo, está livre para uso. Isso facilita os processos de transferência onde é necessário alocar uma porta do E1 para discagem, etc...

Parâmetros:

PortStart – Indica a porta inicial do intervalo

PortEnd – Indica a porta final do intervalo

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta

fora do intervalo permitido, excedendo o número de portas instaladas

5.2.80 h100_SetDefault (dg_h100_SetDefault)



Retorna a configuração padrão da placa indicada

Declarações:

ActiveX:

```
SHORT h100_SetDefault(SHORT Card);
```

API:

```
short dg_h100_SetDefault(short card);
```

Descrição:

Esta função faz com que a placa indicada retorne ao estado inicial de configuração, onde cada canal do E1 está conectado ao seu correspondente DSP.

Parâmetros:

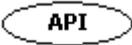
Card – Indica a placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.81 HangUp (dg_HangUp)

APIActiveX

Libera a linha conectada a placa (desliga).

Declarações:

ActiveX:

```
SHORT HangUp(SHORT Port);
```

API:

```
short dg_HangUp(short port);
```

Descrição:

Ao chamar este método, a linha é desconectada e a porta liberada, equivalendo ao desligar do telefone. Na placa E1 o hangup é interpretado como um *Idle* (*R2 valor 0x9*) o que faz com que a linha seja desconectada e liberada.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_EXIT_FAILURE - Ocorre caso não seja possível inserir

comando na fila das threads de controle.
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Porta especificada fora do intervalo de portas configuradas

5.2.82 IsCallInProgress



Indica se existe uma discagem em curso.

Declarações:

ActiveX:
`VARIANT_BOOL IsCallInProgress(SHORT Port);`

Descrição:

Esta função permite saber se existe uma discagem em curso, iniciada pelo **MakeCall**.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

True ou False

5.2.83 IdleAbort (dg_IdleAbort)

API

ActiveX

Interrompe a execução de uma função de Idle.

Declarações:ActiveX:`SHORT IdleAbort(SHORT Port)`API:`short dg_IdleAbort(short port)`**Descrição:**

Ao chamar este método a monitoração de linha para atendimento será cancelada.

Parâmetros:

Port – Porta onde será feita o cancelamento da monitoração

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.84 IdleStart (dg_IdleStart)

API

ActiveX

Inicia a execução de uma função de Idle.

Declarações:

ActiveX:

```
SHORT IdleStart(SHORT Port)
```

API:

```
short dg_IdleStart(short port)
```

Descrição:

Ao chamar este método a função de monitoração de estado de espera é iniciada. O método **IdleSettings** deve ser chamado com todas as configurações necessárias antes de se iniciar a monitoração.

Parâmetros:

Port – Porta onde será feita a monitoração

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.85 IdleSettings (dg_IdleSettings)

API

ActiveX

Configura a monitoração do estado de espera.

Declarações:

ActiveX:

```
SHORT IdleSettings(SHORT Port, VARIANT_BOOL
AutoPickUp, SHORT RingCount,
                    SHORT PauseAfterPickUp,
VARIANT_BOOL WatchTrunkBefore,
                    VARIANT_BOOL
WatchTrunkAfter, SHORT Format, SHORT
TimeOut,
                    SHORT Max, LPCTSTR
TermDigits);
```

API:

```
short WCDECL dg_IdleSettings(short port, u8
AutoPickUp, short RingCount,
                               short
PauseAfterPickUp, u8 WatchTrunkBefore,
                               u8
WatchTrunkAfter, short Format,
                               short TimeOut,
short Max, char *TermDigits)
```

Descrição:

Esta função configura as opções a serem monitoradas durante o estado de espera. Permite configurar detecção automática de BINA (OnCallerID), atendimento automático e integração com o PABX

Parâmetros:

- **Port** – Indica o canal da Placa que será monitorado

- **AutoPickUp** – Indica se atende automático ou não. (*0/1 na API e TRUE/FALSE no ActiveX*)
- **RingCount** – Número de rings para o atendimento automático
- **PauseAfterPickUp** – Pausa após o pickup
- **WatchTrunkBefore** – Monitora a linha antes do atendimento. (*0/1 na API e TRUE/FALSE no ActiveX*)
- **WatchTrunkAfter** - Monitora a linha depois do atendimento. (*0/1 na API e TRUE/FALSE no ActiveX*)
- **Format** - wtDTMF, wtMFP, wtCustom – Ver detalhes no capítulo **Funções Especiais**.
- **Timeout** – Timeout interdigito
- **Max** – Número máximo de dígitos da sinalização
- **TermDigits** – Indica um ou mais dígitos como finalizadores.

5.2.86 IsPlaying (dg_IsPlaying)

API

ActiveX

Indica se a placa está reproduzindo alguma mensagem

Declarações:

ActiveX:

```
VARIANT_BOOL IsPlaying(SHORT Port);
```

API:

```
short IsPlaying(short port)
```

Descrição:

Este método permite ao desenvolvedor saber se o canal especificado está reproduzindo uma mensagem

Parâmetros:

Port – Indica o canal da Placa

Valores de Retorno:

ActiveX - TRUE ou FALSE

API - 1 - Reproduzindo , 0 - Não reproduzindo

5.2.87 IsRecording (dg_IsRecording)



Indica se a placa está gravando alguma mensagem

Declarações:ActiveX:

```
VARIANT_BOOL IsRecording(SHORT Port);
```

API:

```
short IsRecording(short port)
```

Descrição:

Este método permite ao desenvolvedor saber se o canal especificado está gravando uma mensagem

Parâmetros:

Port – Indica o canal da Placa

Valores de Retorno:

ActiveX - TRUE ou FALSE

API - 1 - Reproduzindo , 0 - Não reproduzindo

5.2.88 LocalBridgeConnect (dg_LocalBridgeConnect)



Efetua uma conexão bi-direcional entre duas portas na placa E1 sem barramento H100

Declarações:ActiveX:

```
SHORT LocalBridgeConnect(SHORT Port1, SHORT  
Port2);
```

API:

```
short dg_LocalBridgeConnect(short port1, short  
port2);
```

Descrição:

Ao chamar esta função, a porta *Port1* é conectada em *Port2* e vice-versa, permitindo que o áudio seja enviado nos dois sentidos. Ambas as portas precisam pertencer à mesma placa, obrigatoriamente.

Parâmetros:

Port1 – Indica a primeira porta

Port2 – Indica a segunda porta

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.89 LocalBridgeDisconnect (dg_LocalBridgeDisconnect)

Efetua uma desconexão de duas portas conectadas, em placas E1 sem H100

Declarações:

ActiveX:
SHORT LocalBridgeDisconnect(SHORT Port1, SHORT Port2);

API:
short dg_LocalBridgeDisconnect(short port1, short port2);

Descrição:

Ao chamar esta função, as duas portas voltam a ter o comportamento inicial, de antes de serem conectadas pelo LocalBridgeConnect

Parâmetros:

Port1 – Indica a primeira porta

Port2 – Indica a segunda porta

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.90 MakeCall



Inicia a discagem com supervisão.

Declarações:

ActiveX:

```
SHORT MakeCall(SHORT Port, SHORT CallType,  
LPCTSTR Number, LPCTSTR InitialPhrase,
```

VARIAN

```
T_BOOL WithAnalysis, SHORT DialType);
```

Descrição:

Este método inicia a discagem com supervisão conforme configurado pelos métodos **SetCallxxxx**. O término será tratado no evento OnAfterMakeCall.

Parâmetros:

Port – Indica o canal da Placa.

CallType – *ctExternal* ou *ctWithFlash*

Number – String do Número que será discado

InitialPhrase – String da frase a ser reproduzida no início do processo

WithAnalysis – True/False que indicará se a discagem será com supervisão (monitorar ocupado, etc...) ou sem (ligação entregue após a discagem).

DialType– Define o tipo de discagem :pulso/tom.

Retorno:

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_EXIT_SUCCESS - executado com sucesso

5.2.91 MenuAbort



Interrompe a execução de uma função de Menu.

Declarações:

ActiveX:

```
SHORT MenuAbort (SHORT Port) ;
```

Descrição:

Ao chamar este método a função iniciado pelo MenuStart será interrompida para o canal específico.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Um dos parâmetros fora do intervalo permitido

5.2.92 MenuErrorSettings



Configura as opções de menu

Declarações:

ActiveX:

```
SHORT MenuErrorSettings(SHORT Port, LPCTSTR  
InvalidDigitPhrase, SHORT InvalidDigitRetries,  
LPCTSTR TimeOutPhrase);
```

Descrição:

Tem a finalidade de configurar as frases e situações de erro de entrada de dados das funções especiais de menu

Parâmetros:

Port – Indica o canal da Placa.

InvalidDigitPhrase – Frase utilizada em caso de opção inválida

InvalidDigitRetries – Número de tentativas

TimeOutPhrase – Frase a ser reproduzida caso o usuário não digite nada no tempo especificado no MenuStart.

Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Um dos parâmetros fora do intervalo permitido

5.2.93 MenuStart



Inicia a função de menu.

Declarações:

ActiveX:

```
SHORT MenuStart(SHORT Port, LPCTSTR PlayMessage,  
LPCTSTR ValidDigits, SHORT TimeOut,  
VARIANT_BOOL  
EnablePulseDetection, SHORT  
PulseSensibility);
```

Descrição:

Inicia a execução de um menu. Após sua execução o método

OnMenu será chamado devolvendo o dígito escolhido e o status. O parâmetro **PlayMessage** pode receber um arquivo de áudio literal ou "@" para executar uma lista de mensagens configurada pelas funções **PlayListXXX**.

Parâmetros:

Port – Indica o canal da Placa.

Message – Frase do menu ou "@"

ValidDigits – Dígitos considerados válidos. Devem ser colocados todos os dígitos de opções válidas sem separadores. Ex.: "235"

TimeOut – Tempo máximo para digitação da opção após a frase definida em **Message**.

EnablePulseDetection - Habilita/Desabilita detecção de pulse durante o funcionamento do Menu

PulseSensibility - Sensibilidade de detecção - passar zero para assumir o padrão

Retorno:

DG_EXIT_SUCCESS - executado com sucesso

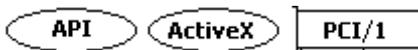
DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Um dos parâmetros fora do intervalo permitido

DG_ERROR_PLAY_EMPTYLIST - Lista de mensagens vazia, no caso de passar "@" no parâmetro **PlayMessage**.

5.2.94 MicOff (dg_MicOff)



Desconecta o áudio do microfone ao HeadSet. Somente para placas de 1 canal.

Declarações:

ActiveX:
`SHORT MicOff(SHORT Port);`

API:
`short dg_MicOff(short port);`

Descrição:

Existem na placa 3 conectores. Num deles pode ser conectado um HeadSet, por onde o usuário do sistema poderá estabelecer conversação, dispensando o uso do aparelho telefônico. Quando o componente é inicializado, o microfone do headset está desconectado. Porém, o programador tem a flexibilidade de a qualquer momento conectá-lo e desconectá-lo. Ao desconectar o áudio, o interlocutor não poderá ouvir o que é falado através do headset, equivalente a função MUTE dos aparelhos telefônicos.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

5.2.95 MicOn (dg_MicOn)



Conecta o áudio do microfone ao HeadSet.

Declarações:

ActiveX:
`SHORT MicOn(SHORT Port);`

API:
`short dg_MicOn(short port);`

Descrição:

Existem na placa 3 conectores. Num deles pode ser conectado um HeadSet, por onde o usuário do sistema poderá estabelecer conversação, dispensando o uso do aparelho telefônico. Quando o componente é inicializado, o microfone do headset está desconectado. Porém, o programador tem a flexibilidade de a qualquer momento conectá-lo e desconectá-lo. Ao conectar o áudio, o interlocutor poderá ouvir o que é falado através do headset.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

5.2.96 PauseInputBuffer (dg_PauseInputBuffer)

API

ActiveX

Interrompe temporariamente o envio de amostras entre a placa e a voicerlib

Declarações:

ActiveX:

```
SHORT PauseInputBuffer(SHORT Port, VARIANT_BOOL  
Paused);
```

API:

```
short dg_PauseInputBuffer(short port, short  
paused);
```

Descrição:

O método `PauseInputBuffer` permite que o envio de amostras da placa para a aplicação fique em pausa. É útil quando o `EnableInputBuffer` é chamado apenas no início da aplicação e não a cada gravação, para evitar utilização de processador desnecessária, principalmente quando se utiliza as *callbacks* para tratar as amostras diretamente para a aplicação.

Parâmetros:

Port – Indica o canal da Placa.

Paused – `DG_PAUSE` (1) coloca em pausa e `DG_RELEASE` (0) retira.

Valor de Retorno:

`DG_EXIT_SUCCESS` - executado com sucesso

`DG_ERROR_PORT_OUT_OF_RANGE` - Parâmetro fora

do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_REC_NOT_RECORDING - Não está gravando

5.2.97 PhoneOff (dg_PhoneOff)



Desconecta o áudio do fone ao HeadSet.

Declarações:

ActiveX:

```
SHORT PhoneOff(SHORT Port);
```

API:

```
short dg_PhoneOn(short port);
```

Descrição:

Existem na placa 3 conectores. Num deles pode ser conectado um HeadSet, por onde o usuário do sistema poderá estabelecer conversação, dispensando o uso do aparelho telefônico. Quando o componente é inicializado, o fone do headset estará desconectado. Porém, o programador tem a flexibilidade de a qualquer momento conectar e desconectar de forma independente, o áudio do fone, fazendo uso dos métodos PhoneOn() e PhoneOff(). Enquanto o fone não for conectado, o usuário não ouvirá nada, mas isso não implica que a linha está desligada. Uma ligação pode estar atendida e a placa poderá gravar tanto a fala da pessoa que estiver na outra ponta da ligação quanto a do usuário do sistema, que está com o headset, desde que o microfone do headset esteja ligado,

porém este não estará ouvindo nada.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

5.2.98 PhoneOn (dg_PhoneOn)



Conecta o áudio do fone ao HeadSet.

Declarações:

ActiveX:
`SHORT HangUp(SHORT Port);`

API:
`short dg_HangUp(short port);`

Descrição:

Existem na placa 3 conectores. Num deles pode ser conectado um HeadSet, por onde o usuário do sistema poderá estabelecer conversação, dispensando o uso do aparelho telefônico.

Quando o componente é inicializado, o fone do headset estará desconectado. Porém, o programador tem a flexibilidade de a qualquer momento conectar e desconectar de forma

independente, o áudio do fone, fazendo uso dos métodos `PhoneOn()` e `PhoneOff()`.

Enquanto o fone não for conectado, o usuário não ouvirá nada, mas isso não implica que a linha está desligada. Uma ligação pode estar atendida e a placa poderá gravar tanto a fala da pessoa que estiver na outra ponta da ligação quanto a do usuário do sistema, que está com o headset, desde que o microfone do headset esteja ligado, porém este não estará ouvindo nada.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

5.2.99 PlayBuffer (dg_PlayBuffer)

API

ActiveX

Insere um vetor de amostras diretamente na placa

Declarações:

ActiveX:

```
SHORT PlayBuffer(SHORT Port, LONG Samples,  
SHORT Samplesize);
```

API:

```
short dg_PlayBuffer(short port, void
```

```
*Samples, short samples_size, int  
*remaining_size)
```

Descrição:

O PlayBuffer permite inserir diretamente na porta indicada um array de amostras de áudio a ser reproduzida pela placa. Este array pode ter até 8192 bytes, portanto o valor de `samples_size` não poderá ultrapassar este limite.

Na API é possível passar um ponteiro para um inteiro e se este ponteiro for fornecido, a quantidade de bytes livres será retornada na parâmetro `remaining_size`.

A forma de utilização e as técnicas de programação deste método são discutidas no **Guia do Programador**, no tópico **Streaming de Áudio**.

() Não é possível enviar amostras para a placa enquanto um arquivo estiver sendo reproduzido nesta mesma porta.*

Parâmetros:

Port – Indica a porta para onde as amostras serão enviadas
Samples – Vetor contendo as amostras a serem enviadas
SamplesSize - Tamanho do vetor **Samples**
remaining_size - Retorna a quantidade de bytes livres no buffer

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_ALREADY_PLAYING - Uma reprodução já está

em andamento
DG_ERROR_PARAM_OUTOFRANGE - *samples_size* maior
que o tamanho do buffer (8Kbytes)

5.2.100 PlayCardinal



Permite reproduzir numerais cardinais inteiros ou fracionários por extenso.

Declarações:

ActiveX:
SHORT PlayCardinal(SHORT Port, LPCTSTR Value,
LPCTSTR TermDigits, LONG PauseBefore);

Parâmetros:

Port – Indica o canal da Placa.

Value – Uma string ou variável contendo o número a ser reproduzido. Não se deve utilizar ponto como separador de milhares e **deve-se** utilizar a vírgula como separador decimal

TermDigits - É uma string contendo um ou mais dígitos, que ao serem detectados finaliza a reprodução das mensagens e gera os eventos OnPlayStop e OnDigitsReceived.

PauseBefore – Indica uma pausa de *n* milissegundos antes de iniciar a reprodução

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas
DG_ERROR_PLAY_EMPTYLIST - Lista de mensagens vazia

5.2.101 PlayCurrency



Permite reproduzir valores monetários por extenso.

Declarações:

ActiveX:
SHORT PlayCurrency(SHORT Port, LPCTSTR Value,
LPCTSTR TermDigits, LONG PauseBefore);

Parâmetros:

Port – Indica o canal da Placa.

Value – Uma string ou variável contendo o valor a ser reproduzido. Não se deve utilizar ponto como separador de milhares e **deve-se** utilizar a vírgula como separador decimal
TermDigits - É uma string contendo um ou mais dígitos, que ao serem detectados finaliza a reprodução das mensagens e gera os eventos OnPlayStop e OnDigitsReceived.

PauseBefore – Indica uma pausa de *n* milissegundos antes de iniciar a reprodução

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas
DG_ERROR_PLAY_EMPTYLIST - Lista de mensagens vazia

5.2.102 PlayDate



Permite reproduzir data por extenso.

Declarações:

ActiveX:

```
SHORT PlayDate(SHORT Port, LPCTSTR Value, LPCTSTR  
Mask, LPCTSTR TermDigits, LONG PauseBefore);
```

Parâmetros:

Port – Indica o canal da Placa.

Value – Uma string ou variável contendo a data a ser reproduzida. Obrigatoriamente deve ser utilizado a barra "/" como separador.

Mask – Máscara utilizada que indica o formato da data.

Pode assumir os seguintes valores:

- **d/m/y** – Ex.: "25 de setembro de 2001"
- **d/m** – Ex.: "25 de setembro"
- **m/d** – Ex.: "Setembro 25"
- **m/d/y** – Ex.: "Setembro 25 2001"

TermDigits - É uma string contendo um ou mais dígitos, que ao serem detectados finaliza a reprodução das mensagens e gera os eventos OnPlayStop e

OnDigitsReceived.

PauseBefore – Indica uma pausa de n milissegundos antes de iniciar a reprodução

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas

DG_ERROR_PLAY_EMPTYLIST - Lista de mensagens vazia

5.2.103 PlayFile (dg_PlayFile)

API

ActiveX

Inicia a reprodução de um arquivo através de uma porta da placa.

Declarações:

ActiveX:

```
SHORT PlayFile(SHORT Port, LPCTSTR FileName,  
LPCTSTR TermDigits, LONG Origin);
```

API:

```
short dg_PlayFile(short port, char *FileName,  
char *TermDigits, long Origin);
```

Descrição:

O método PlayFile inicia a reprodução de um arquivo através da placa. É possível programar a interrupção através de um ou

mais dígitos recebidos através do parâmetro TermDigits. Ao iniciar a reprodução o evento OnPlayStart (EV_PLAYSTART) é gerado e o evento OnPlayStrop (EV_PLAYSTOP) ao término, indicando o motivo da interrupção. É possível interromper também *manualmente* através do método StopRecordFile.

O PlayFile detecta automaticamente o tipo de arquivo, baseado na extensão para .sig, .gsm e se for wave, detecta se é PCM ou LeiMi. Isso permite uso de diversos formatos para reprodução sem preocupação para o programador. Se for utilizado uma extensão desconhecida, fica valendo o que está especificado no SetPlayFormat.

O método **GetPlayFormat** permite saber qual o formato configurado.

Parâmetros:

File – String contendo o nome e caminho completo do arquivo SIG a ser reproduzido.

TermDigits – É uma string contendo um ou mais dígitos, que ao serem detectados finaliza a reprodução das mensagens e gera os eventos OnPlayStop e OnDigitsReceived. Se qualquer dígito puder interromper utilize o símbolo "@". Se não houver dígito finalizador, passar "" (vazio).

Origin - Este parâmetro permite reproduzir a mensagem a partir de um determinado ponto. Se for passado 0 (zero) a mensagem é reproduzida do início. Se for passado **-1** a mensagem é reproduzida a partir do final menos 2 segundos (ideal para ouvir em *real-time*). Qualquer número diferente de **0** e **-1** significa a partir de quantos segundos a mensagem será reproduzida.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso (API)
DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_PLAY_OPENFILE - Arquivo não encontrado ou erro ao tentar abri-lo.

DG_ERROR_AUDIO_FORMAT_UNSUPPORTED - Formato de áudio não suportado pela VoicerLib

DG_ERROR_COULD_NOT_CREATE_THREAD - Não foi possível inicializar *thread* de reprodução

5.2.104 PlayList



Inicia a reprodução de uma lista de mensagens de um determinado canal.

Declarações:

ActiveX:
`SHORT PlayList(SHORT Port, LPCTSTR TermDigits);`

Descrição:

Inicia a reprodução da lista de mensagens associada ao canal indicado por `Port` que foi criada a partir do método `PlayListAdd`. O funcionamento é idêntico ao do `PlayFile`, sendo gerado apenas um evento `OnPlayStart` no início e um `OnPlayStop` no final da última mensagem da lista.

Parâmetros:

Port – Indica o canal da Placa.

TermDigits – É uma string contendo um ou mais dígitos,

que ao serem detectados finaliza a reprodução das mensagens e gera os eventos OnPlayStop e OnDigitsReceived.

Valor de Retorno:

DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_PLAY_EMPTYLIST - Lista de mensagens vazia
DG_ERROR_INVALIDPARAM - Especificação do tipo de reprodução inválida na lista

5.2.105 PlayListAdd



Permite adicionar itens a serem reproduzidos na lista do canal.

Declarações:

ActiveX:
VARIANT_BOOL PlayListAdd(SHORT Port, SHORT
ItemType, LPCTSTR StringValue, LPCTSTR
Mask, LONG PauseBefore);

Parâmetros:

Port – Indica o canal da Placa.
ItemType – Configura o tipo de mensagem a ser reproduzida (ptFile, ptCurrency, ptCardinal, ptDate e ptTime)

Value – É a string que contém o valor a ser reproduzido, respeitando a sintaxe determinada por ItemType.

Mask – Máscara utilizada somente para o tipo ptDate.

PauseBefore – Indica uma pausa de *n* milissegundos antes de iniciar a reprodução

Valor de Retorno:

True ou False

5.2.106 PlayListClear



Elimina todos os itens na lista do canal.

Declarações:

ActiveX:
`void PlayListClear(SHORT Port);`

Descrição:

Este método deve ser chamado antes do método PlayListAdd para eliminar todos os elementos que por ventura estejam lá.

Parâmetros:

Port – Indica o canal da Placa.

5.2.107 PlayListGetCount



Retorna a quantidade de elementos na lista daquele canal.

Declarações:

ActiveX:
`SHORT PlayListGetCount (SHORT Port);`

Descrição:

Esta função devolve a quantidade de elementos da lista de mensagens.

Parâmetros:

Port – Indica o canal da Placa.

Valor de Retorno:

Um inteiro com a quantidade de elementos

5.2.108 PlayListRemoveItem



Remove o item especificado da lista de mensagens do canal

Declarações:

ActiveX:
`VARIANT_BOOL PlayListRemoveItem (SHORT Port, SHORT`

Index);

Descrição:

Remove o item especificado pelo parâmetro Index. Deve ser utilizado um valor entre 0 e $n-1$.

Parâmetros:

Port – Indica o canal da Placa.

Index – Índice do elemento a ser removido

Valor de Retorno:

True se conseguiu remover e False no caso de erro.

5.2.109 PlayNumber



Permite reproduzir números dígito a dígito.

Declarações:ActiveX:

```
SHORT PlayNumber(SHORT Port, LPCTSTR Value,  
LPCTSTR TermDigits, LONG PauseBefore);
```

Parâmetros:

Port – Indica o canal da Placa.

Value – Uma string ou variável contendo o número a ser reproduzido. Não se deve utilizar ponto como separador de milhares e **deve-se** utilizar a vírgula como separador decimal

TermDigits - É uma string contendo um ou mais dígitos,

que ao serem detectados finaliza a reprodução das mensagens e gera os eventos OnPlayStop e OnDigitsReceived. Pode falar também: *barra, traço, vírgula e ponto*

PauseBefore – Indica uma pausa de *n* milissegundos antes de iniciar a reprodução

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas

DG_ERROR_PLAY_EMPTYLIST - Lista de mensagens vazia

5.2.110 PlayTime



Permite reproduzir hora por extenso.

Declarações:

ActiveX:

```
SHORT PlayTime(SHORT Port, LPCTSTR Value, LPCTSTR  
TermDigits, LONG PauseBefore);
```

Parâmetros:

Port – Indica o canal da Placa.

Value – Uma string ou variável contendo a hora a ser reproduzida. Obrigatoriamente a hora deve estar representada no formato **hh:mm:ss** ou **hh:mm**

TermDigits - É uma string contendo um ou mais dígitos, que ao serem detectados finaliza a reprodução das mensagens e gera os eventos OnPlayStop e OnDigitsReceived.

PauseBefore – Indica uma pausa de *n* milissegundos antes de iniciar a reprodução

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas

DG_ERROR_PLAY_EMPTYLIST - Lista de mensagens vazia

5.2.111 PickUp (dg_PickUp)



Atende a linha conectada a placa.

Declarações:

ActiveX:

```
SHORT PickUp(SHORT Port, LONG PauseAfterPickup);
```

API:

```
short dg_PickUp(short port, long  
pause_after_pickup);
```

Descrição:

Sempre que se desejar tomar a linha conectada à placa para

originar uma ligação ou para atender uma ligação entrante, deve-se fazer uso do método `PickUp`. Na placa E1 sem a thread de controle E1 habilitada, o `PickUp` é interpretado com um pedido de ocupação do canal E1 e se a thread de controle estiver habilitada o comportamento é idêntico ao da placa FXO.

A pausa após o atendimento permite ao desenvolvedor continuar os procedimentos de atendimento após um tempo especificado por este parâmetro. Neste caso o evento `OnAfterPickUp` (`EV_AFTERPICKUP`) é gerado após decorrido este tempo.

Parâmetros:

Port – Indica o canal da Placa

PauseAfterPickup - Pausa em milissegundos para gerar o evento **OnAfterPickUp**

Valor de Retorno:

`DG_EXIT_SUCCESS` - executado com sucesso

`DG_EXIT_FAILURE` - Ocorre caso não seja possível inserir comando na fila das threads de controle.

`DG_ERROR_DRIVER_CLOSED` - Driver desabilitado

`DG_ERROR_PORT_OUT_OF_RANGE` - Porta especificada fora do intervalo de portas configuradas

`DG_ERROR_PARAM_OUTOFRANGE` - `PauseAfterPickUp` menor que zero ou maior que 60000 ms

5.2.112 PromptAbort



Interrompe a execução de uma função de `PromptStart`.

Declarações:

ActiveX:
SHORT PromptAbort (SHORT Port);

Descrição:

Ao chamar este método a função iniciada pelo PromptStart será interrompida para o canal específico.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.113 PromptSettings



Configura as opções de prompt

Declarações:

ActiveX:
SHORT PromptSettings (SHORT Port, LPCTSTR
PlaybackPhrase,
LPCTSTR
ConfirmationPhrase, LPCTSTR RetryDigit,

```
LPCTSTR  
ConfirmationDigit, LPCTSTR CancelDigit);
```

Descrição:

Tem a finalidade de configurar as condições de confirmação e conferência dos dados digitados.

Parâmetros:

Port – Indica o canal da Placa.

PlaybackPhrase – Frase utilizada para conferência – ex.:
"Você digitou"

ConfirmationPhrase – Frase para confirmação – ex.:
"Tecla * para confirmar, # para cancelar ou 9 para digitar de novo"

RetryDigit – Dígito que indicará redigitação dos dados

ConfirmationDigit - Dígito que indicará aceitação dos dados

CancelDigit - Dígito que indicará cancelamento da entrada de dados

Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.114 PromptStart



Inicia a função especial de entrada de dados

Declarações:

```
ActiveX:  
SHORT PromptStart(SHORT Port, LPCTSTR  
PromptMessage, SHORT MinDigit,  
SHORT MaxDigit, SHORT  
TimeOut, SHORT InterdigitTimeOut,  
LPCTSTR TermDigits,  
VARIANT_BOOL WithConfirmation,  
VARIANT_BOOL WithPlayback,  
SHORT Retries);
```

Descrição:

Ao chamar este método a função de prompt é iniciada. O processo terminará gerando o evento OnPrompt que indicará o dado digitado e o Status.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Message – Mensagem ou lista de mensagens ("@") a serem reproduzidas para indicar a entrada de dados. Ex.: *"Digite sua senha..."*

MinDigit – Número mínimo de dígitos a serem esperados pela função. Após o timeout ou dígito terminador, se o número mínimo não for alcançado, a função avisará no evento OnPrompt

MaxDigit – Número máximo de dígitos a serem esperados pela função.

TimeOut – Timeout global de espera de dígitos a ser contado após o término da mensagem.

InterdigitTimeOut – Timeout interdígito a ser considerado após a digitação do primeiro dígito.

TermDigits – Dígitos terminadores de entrada de dados. Ex.: *"Disque sua senha e # para terminar"* neste caso a # deve ser passada como parâmetro aqui.

WithConfirmation – Se *true*, Executa as funções de confirmação

WithPlayback – Se *true*, executa as funções de conferência, reproduzindo o que foi digitado.

Retries – Número de repetições do prompt em caso do usuário não digitar nada.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Um dos parâmetros fora do intervalo permitido

DG_ERROR_PLAY_EMPTYLIST - Lista de mensagens vazia, no caso de passar "@" no parâmetro **PlayMessage**.

5.2.115 R2AskForID (dg_R2AskForId)



Envia um pedido da identificação do assinante para a thread E1.

Declarações:

ActiveX:
`SHORT R2AskForID(SHORT Port);`

API:
`short dg_R2AskForId(short port);`

Descrição:

Caso seja necessário enviar um comando de pedido de

identificação (C_ASK_FOR_ID) manualmente, diretamente para a *thread* de controle E1, deve ser utilizada esta função. A *thread* de controle já deverá ter sido inicializada através da função **CreateE1Thread (dg_CreateE1Thread)**.

Normalmente não será necessário utilizar esta função, pois o método **ConfigE1Thread (dg_CreateE1Thread)** permite que este e outros parâmetros sejam configurados previamente.

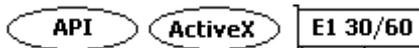
Parâmetros:

Port – Porta para onde será enviado o comando.

Valores de Retorno:

DG_EXIT_SUCCESS - Função executada com sucesso
DG_EXIT_FAILURE - Função executada com erro
DG_ERROR_DRIVER_CLOSED - Driver não inicializado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro passado está fora do intervalo permitido
DG_ERROR_THREAD_NOT_RUNNING - Thread E1 não foi inicializada

5.2.116 R2AskForGroupII (dg_R2AskForGroupII)



Envia um pedido do grupo II para a *thread* E1.

Declarações:

ActiveX:

```
SHORT R2AskForGroupII(SHORT Port);
```

API:

```
short dg_R2AskForGroupII(short port);
```

Descrição:

Caso seja necessário enviar um comando de pedido de grupo II (C_PREP_RX_GRUPO_B) manualmente, diretamente para a *thread* de controle E1, deve ser utilizada esta função. A *thread* de controle já deverá ter sido inicializada através da função **CreateE1Thread (dg_CreateE1Thread)**.

Normalmente não será necessário utilizar esta função, pois o método **ConfigE1Thread (dg_CreateE1Thread)** permite que este e outros parâmetros sejam configurados previamente.

Parâmetros:

Port – Porta para onde será enviado o comando.

Valores de Retorno:

DG_EXIT_SUCCESS - Função executada com sucesso
DG_EXIT_FAILURE - Função executada com erro
DG_ERROR_DRIVER_CLOSED - Driver não
inicializado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro
passado está fora do intervalo permitido
DG_ERROR_THREAD_NOT_RUNNING - Thread E1
não foi inicializada

5.2.117 R2SendGroupB (dg_SendGroupB)

API

ActiveX

E1 30/60

Envia o grupo B para a thread E1.

Declarações:

ActiveX:

```
SHORT R2SendGroupB(SHORT Port);
```

API:

```
short dg_R2SendGroupB(short port, short  
groupb_type)
```

Descrição:

Esta função envia para a thread E1, o grupo B da sinalização E1, que pode assumir os valores abaixo.

Normalmente não será necessário utilizar esta função, pois o método **ConfigE1Thread (dg_CreateE1Thread)** permite que este e outros parâmetros sejam configurados previamente.

Parâmetros:

Port – Porta para onde será enviado o comando.

GroupB_Type - Tipo de grupo B a ser enviado:

B_FREE_CALLING = 1

B_BUSY = 2

B_NUMBER_CHANGED = 3

B_CONGESTION = 4

B_FREE_WITHOUTBILLING = 5

B_COLLECTCALL = 6

B_NUMBER_UNKNOWN = 7

B_OUT_OF_SERVICE = 8

Valores de Retorno:

DG_EXIT_SUCCESS - Função executada com sucesso
DG_EXIT_FAILURE - Função executada com erro
DG_ERROR_DRIVER_CLOSED - Driver não
inicializado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro
passado está fora do intervalo permitido
DG_ERROR_THREAD_NOT_RUNNING - Thread E1
não foi inicializada

5.2.118 ReadDigits (dg_ReadDigits)

API

ActiveX

Lê o conteúdo do buffer de dígitos do canal especificado

Declarações:ActiveX:

```
BSTR ReadDigits(SHORT Port)
```

API:

```
short dg_ReadDigits(short port, char *szDigits)
```

Descrição:

O método ReadDigits permite ler o conteúdo do buffer de dígitos do canal informado. Este buffer é preenchido pelos métodos PlayFile, RecordFile ou GetDigits.

Parâmetros:

Port – Indica o canal da Placa

szDigits - *null-terminated string* que receberá os dígitos (somente API)

Valor de Retorno:**ActiveX:**

Retorna uma string com os dígitos ou nulo caso não haja dígitos no buffer interno da biblioteca

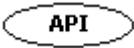
API:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.119 RecordPause (dg_RecordPause)

APIActiveX

Interrompe temporariamente uma gravação, permitindo sua continuação no mesmo arquivo

Declarações:ActiveX:

```
SHORT RecordPause(SHORT Port, VARIANT_BOOL  
Paused);
```

API:

```
short dg_RecordPause(short port, short paused);
```

Descrição:

O método RecordPause permite que a gravação fique em pausa. Isto facilita a programação principalmente quando o usuário quiser colocar o cliente em espera e não desejar que este período seja gravado.

Parâmetros:

Port – Indica o canal da Placa.

Paused – *DG_PAUSE* (1) coloca em pausa e *DG_RELEASE* (0) retira.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_REC_NOT_RECORDING - Não está gravando

5.2.120 RecordFile (dg_RecordFile)

APIActiveX

Inicia a gravação de um arquivo de áudio através de uma porta da placa.

Declarações:

ActiveX:

```
SHORT RecordFile(SHORT Port, LPCTSTR FileName,  
LPCTSTR TermDigits);
```

API:

```
short dg_RecordFile(short port, char *FileName,  
char *TermDigits);
```

Descrição:

O método **RecordFile** inicia a gravação de um arquivo através da placa. É possível programar a interrupção da gravação através de um ou mais dígitos recebidos através do parâmetro **TermDigits**. Ao iniciar a gravação o evento **OnRecordStart** é gerado e o evento **OnRecordStop** ao término, indicando o motivo da interrupção. É possível interromper também *manualmente* através do método **StopRecordFile**. Também é possível colocar a gravação em pause, sem fechar o arquivo através do método **RecordPause**.

O **RecordFile** só poderá ser chamado após a chamada do método **EnableInputBuffer**, que inicia o envio de amostras de áudio da placa para a VoicerLib.

O método **SetRecordFormat** deve ser chamado para configurar o formato de gravação a ser utilizado. A VoicerLib não assume nenhum formato baseado apenas na extensão do arquivo. O método **GetRecordFormat** permite saber qual o formato configurado.

Parâmetros:

Port – Indica o canal da Placa.

File – String contendo o nome e caminho completo do arquivo.

TermDigits – É uma string contendo um ou mais dígitos, que ao serem detectados finaliza a execução do **GetDigits** e gera o evento **OnDigitsReceived**. Se qualquer dígito puder interromper utilize o símbolo "@" ao invés de colocar "0123456789#*", apesar de a segunda forma também estar

correta. Se não houver dígito finalizador, passar "" (vazio).

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso (API)
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_THREAD_NOT_RUNNING - O streaming de áudio não foi habilitado através do método **EnableInputBuffer**.

5.2.121 ResetError (dg_ResetError)



Envia um pedido para verificação de código de erro e zera código na placa.

Declarações:ActiveX:

SHORT ResetError(SHORT Card, SHORT Flag)

API:

short dg_ResetError(short card, short flag)

Descrição:

O firmware do hardware mantém um endereçamento de memória para arquivar os códigos de erro de hardware. Esses erros não deverão ocorrer nunca, porém a monitoração dos mesmos permitirá diagnosticar com mais facilidades problemas de hardware ou mesmo conflitos com o sistema operacional.

Parâmetros:

Card – Indica a placa que será enviado o comando.

Flag – Este parâmetro pode assumir dois valores.

- **DONTSEND_ERROR** (0): O flag com este valor simplesmente *reseta* a informação de erro da placa
- **SEND_ERRORCODE** (1): Passando-se este valor, será gerado um evento OnErrorDetected (EV_ERROR) com o código de erro existente, além de *reseta* a informação de erro da placa. **IMPORTANTE:** Nunca use este parâmetro dentro do próprio tratamento do evento para evitar *reentrância* infinita.

Valores de Retorno:

DG_EXIT_SUCCESS - Função executada com sucesso

DG_ERROR_DRIVER_CLOSED - Driver não inicializado

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro

Flag não é DONTSEND_ERROR ou SEND_ERRORCODE

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro Card fora do número de placas instalado

5.2.122 ResetPortResource (dg_ResetPortResource)

API

ActiveX

Recupera a configuração original de portas virtuais

Declarações:

ActiveX:

```
SHORT ResetPortResource();
```

API:

```
short dg_ResetPortResource()
```

Descrição:

A qualquer momento é possível voltar a numeração das portas para o reconhecimento automático, bastando para isso chamar o método **ResetPortResource**. Este método não exige nenhum parâmetro e fará com que a VoicerLib efetue o reconhecimento de portas feito na inicialização.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

5.2.123 dg_SetEventCallback

**Declarações:**API:

```
void dg_SetEventCallback(void *ptrFunc, void  
*context_data);
```

Descrição:

Esta função, exclusiva da API, indica à VoicerLib qual a função que fará o tratamento dos eventos da placa. Nela deve ser passado o ponteiro da função e a estrutura que guardará os

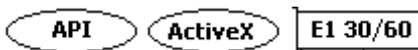
dados dos eventos.

Para maiores detalhes e um exemplo de funcionamento, consulte o tópico **Conceitos Básicos - API do Guia de Programação** deste manual.

Parâmetros:

***ptrFunc**: Ponteiro da função que tratará os eventos.
context_data: ponteiro para a estrutura que conterà os dados do evento

5.2.124 SendR2Command (dg_SendR2Command)



Envia um comando R2 diretamente para a porta de um tronco E1.

Declarações:

ActiveX:
`SHORT SendR2Command(SHORT Port, LONG Command)`

API:
`short dg_SendR2Command(short port, int r2)`

Descrição:

Esta função permite enviar comandos R2 diretamente para o canal da placa. Além disso, serve para habilitar ou desabilitar a

detecção dos comandos R2 de um determinado canal. Caso se esteja utilizando a *thread de controle E1*, não é necessário chamar este método diretamente, pois isto é feito pela própria *thread*.

Ao habilitar a detecção, o evento OnR2Received (EV_R2) será gerado toda vez que for detectado um sinal R2.

Parâmetros:

Port – Indica a porta do tronco E1

Command – Indica o comando R2 a ser passado, sendo:

- **R2_IDLE** (0x9)
- **R2_CLEAR_FOWARD** (0x9)
- **R2_SEIZURE** (0x1)
- **R2_BACKWARD_DISCONNECTION** (0x1)
- **R2_SEIZURE_ACK** (0xd)
- **R2_BILLING** (0xd)
- **R2_CLEAR_BACK** (0xd)
- **R2_ANSWERED** (0x5)
- **R2_BLOCKED** (0xd)
- **R2_FAILURE** (0xd)
- **R2_ENABLEDETECTION** (0x10) - Habilita a detecção
- **R2_DISABLEDETECTION** (0x20) - Desabilita a detecção

Valores de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro
passado está fora do intervalo permitido (Port)
DG_ERROR_DRIVER_CLOSED - Driver não
inicializado

5.2.125 SetAlarmMode (dg_SetAlarmMode)

API

ActiveX

E1 30/60

Configura o modo de notificação dos alarmes dos framers E1

Declarações:

ActiveX:

```
SHORT SetAlarmMode(SHORT Card, SHORT Mode);
```

API:

```
short WCDECL dg_SetAlarmMode(short card,  
short mode);
```

Descrição:

A placa VoicerBox E1 30/60 gera eventos de alarmes indicando qualquer tipo de problema nos troncos E1. Por padrão inicial a notificação dos alarmes é manual, ou seja, a placa não envia estes alarmes para a VoicerLib e esta conseqüentemente não gera o evento OnE1Alarm. Se for necessário saber o *status* do alarme, é necessário chamar a função GetAlarmStatus.

Chamando esta função com o parâmetro Mode com valor ALARM_AUTOMATIC_NOTIFY, os eventos de alarme ocorrerão automaticamente logo que ocorrerem.

Recomenda-se utilizar o modo automático mas somente depois de se configurar o sincronismo das placas.

(Os tipos de alarmes são explicados no evento OnE1Alarm).

Parâmetros:

Card – Indica a placa que será enviado o comando.

Mode – Modo de notificação

- ALARM_MANUAL_NOTIFY - O evento OnE1Alarm só é gerado após a chamada da função GetAlarmStatus
- ALARM_AUTOMATIC_NOTIFY - O evento OnE1Alarm ocorre sempre quando um alarme for detectado.

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro CARD fora do intervalo permitido, excedendo o número de placas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido. (Mode diferente de

ALARM_MANUAL_NOTIFY e

ALARM_AUTOMATIC_NOTIFY)

5.2.126 SetAnswerSensitivity (dg_SetAnswerSensitivity)



Permite aumentar ou diminuir a sensibilidade da detecção de atendimento.

Declarações:

ActiveX:

```
SHORT SetAnswerSensitivity(short Factor);
```

API:

```
short dg_SetAnswerSensitivity(short factor);
```

Descrição:

Este método configura o nível de sensibilidade da detecção de atendimento. Deve ser chamado no início da aplicação e afeta todos os canais ao mesmo tempo. Não tem efeito nas placas digitais.

Se for passado Factor fora do intervalo permitido, o sistema assume o padrão **6**.

A chamada dessa função deve ser feita **antes** do EnableAnswerDetection para ter efeito.

Parâmetros:

Factor – Fator de sensibilidade que varia de 1 (menos sensível) até 10 (mais sensível). No início, o valor padrão é 6

Valores de Retorno:

EXIT_SUCCESS (0) - executado com sucesso (API)
ERROR_DRIVER_CLOSED (-3) - Driver desabilitado

5.2.127 SetAnswerThreshold (dg_SetAnswerThreshold)



Permite alterar o limiar para detecção de atendimento.

Declarações:ActiveX:

```
SHORT SetAnswerThreshold(SHORT Port, SHORT  
Threshold)
```

API:

```
short dg_SetAnswerThreshold(short port, short  
threshold)
```

Descrição:

O limiar de atendimento deve ser utilizado em conjunto com o fator de sensibilidade ([SetAnswerSensitivity](#)) para melhorar a detecção de atendimento em casos onde ela é dificultada pelas condições da rede pública.

Parâmetros:

Port - Canal

Threshold - Pode variar de 1 a 30. O valor padrão é 4. Quanto maior o valor, maior é a sensibilidade, mais fácil de detectar o atendimento. Se o valor estiver muito alto, é possível que o atendimento seja detectado antes da hora.

Valores de Retorno:

EXIT_SUCCESS (0) - executado com sucesso (API)
ERROR_DRIVER_CLOSED (-3) - Driver desabilitado

5.2.128 SetAudioInputCallback (dg_SetAudioInputCallback)

APIActiveX

Configura a função CallBack que poderá receber as amostras de áudio da placa

Declarações:

ActiveX:

```
SHORT SetAudioInputCallback(LONG FunctionPointer)
```

API:

```
void dg_SetAudioInputCallback(void *ptrFunc)
```

Descrição:

Este método passa para a VoicerLib o ponteiro da função Callback que tratará as amostras de áudio recebidas da placa. Para receber estas amostras é necessário habilitar o seu envio através do método **EnableInputBuffer**.

Parâmetros:

Ponteiro da função Callback

5.2.129 SetCardDetections (dg_SetCardDetections)

APIActiveX

Configura o valor a ser utilizado nas detecções de tons.

Declarações:ActiveX:

```
SHORT SetCardDetections(SHORT Card, SHORT  
DetectionType, FLOAT Value1, FLOAT Value2);
```

API:

```
short dg_SetCardDetections(short card, short  
type, float value1, float value2);
```

Descrição:

Esta função permite configurar os valores das freqüências que serão monitoradas pelo sistema. A VoicerLib já vem com valores pré-configurados que servirão para a maioria dos casos. A utilização desta função permite alterar estes valores, caso seja necessário.

É importante ressaltar que esta função altera as configurações de freqüências de tons para determinada placa, e não por porta.

Parâmetros:

Card – Indica a placa que será enviado o comando.

Type – Este parâmetro indica qual dos *presets* será alterado

- **CFG_DETECT_FREQTONE1** - Pré-configurado em 425Hz
- **CFG_DETECT_FREQTONE2** - Pré-configurado em 1100Hz (Fax)
- **CFG_DETECT_FREQTONE3** - Pré-configurado em 2100Hz (Fax)
- **CFG_DETECT_FREQTONE4** - Livre
- **CFG_DETECT_FREQTONE5** - Livre
- **CFG_DETECT_FREQTONE6** - Livre
- **CFG_DETECT_FREQTONE7** - Livre
- **CFG_DETECT_FREQTONE8** - Livre

Value1 – Indica o valor da primeira freqüência a ser configurado (em Hz)

Value2 – Indica o valor da segundo freqüência a ser configurado (em Hz). Se não existir segunda freqüência, utilize o valor zero.

Valores de Retorno:

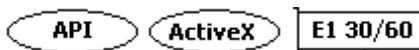
DG_EXIT_SUCCESS - Thread iniciada com sucesso
DG_ERROR_DRIVER_CLOSED - Driver não

inicializado

DG_ERROR_CARD_OUT_OF_RANGE - Número da placa fora do intervalo permitido

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro **Type** diferente dos valores mostrados acima.

5.2.130 SetCardSyncMode (dg_SetCardSyncMode)



Configura o tipo de sincronismo que a placa funcionará.

Declarações:

ActiveX:

```
SHORT SetCardSyncMode(SHORT Card, SHORT SyncMode);
```

API:

```
short dg_SetCardSyncMode(unsigned short card, unsigned short SyncMode);
```

Descrição:

Devido à características existentes somente nas linhas digitais (E1) e também devido à existência do conector H100 utilizado para interconexão de placas, o desenvolvedor deve prestar atenção às configurações de sincronismo após a inicialização. Sempre após inicializar o driver (StartVoicerLib) é necessário configurar o modo de operação e o sincronismo das placas. Caso esta configuração seja omitida, a aplicação poderá apresentar erros de sinalização, etc...

Para maiores informações, leia o tópico **Configurações de Sincronismo** referentes à placa E1 no **Guia de Programação**

deste manual.

Parâmetros:

Card – Indica a placa que será enviado o comando.

SyncMode – Este parâmetro indica o tipo de sincronismo que a placa funcionará

- **SYNC_INTERNAL** - Placa MASTER com sincronismo interno
- **SYNC_LINE_A** - Placa MASTER com sincronismo externo no E1 A (Primeiro E1)
- **SYNC_LINE_B** - Placa MASTER com sincronismo externo no E1 B (Segundo E1, em placas de 60 canais)
- **SYNC_H100_CT_A** - Placa operando em modo SLAVE

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_CARD_OUT_OF_RANGE - Card informado fora do intervalo de placas instaladas.

DG_FEATURE_NOT_SUPPORTED - A placa não precisa de configuração de sincronismo

DG_ERROR_PARAM_OUTOFRANGE - SyncMode inválido

5.2.131 SetCallAfterAnswer



Configura as opções após a detecção de atendimento do método

MakeCall

Declarações:

ActiveX:

```
SHORT SetCallAfterAnswer(SHORT Port, LPCTSTR  
FileName,  
SHORT Pause,  
VARIANT_BOOL AutoHangUp);
```

Descrição:

Este método configura uma frase para ser reproduzida após a detecção de atendimento

Parâmetros:

Port – Indica o canal da Placa.

FileName – Indica o nome do arquivo de voz a ser reproduzido

Pause – Em milisegundos, indica a pausa após o atendimento **antes** de reproduzir a mensagem.

AutoHangUp – Indica se, após uma discagem do tipo Flash, desliga automaticamente quando detectado o atendimento.

Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.132 SetCallAfterPickup



Configura as opções após o pickup do método MakeCall

Declarações:

Delphi:

```
procedure SetCallAfterPickup(Port: Smallint; Digiti:
string;
                                PauseAfter: SmallInt);
```

Visual Basic:

```
Function SetCallAfterPickup(Port As Integer, Digit as
String,
                                PauseAfter as Integer) As Integer
```

Descrição:

Este método configura os dígitos a serem discados após o atendimento (ex. Para pegar linha externa) e uma pausa.

Parâmetros:

Port – Indica o canal da Placa.

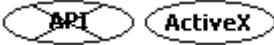
Digit – Indica o(s) dígito(s) a serem discados

PauseAfter – Em milisegundos, indica a pausa após o Pickup dado pelo método MakeCall.

Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.133 SetCallBusyPhrase



Configura a frase a ser reproduzida em caso de ocupado.

Declarações:

Delphi:

```
procedure SetCallBusyPhrase(Port: Smallint; FileName: string);
```

Visual Basic:

```
Function SetCallBusyPhrase (Port As Integer,Filename as String) As Integer
```

Descrição:

Este método configura uma frase para ser reproduzida em caso de ocupado.

Parâmetros:

Port – Indica o canal da Placa.

FileName – Indica o nome do arquivo de voz a ser reproduzido

Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.134 SetCallBusyReturnFlash



Configura o flash em caso de ocupado.

Declarações:

Delphi:

```
procedure SetCallBusyReturnFlash(Port: Smallint; Count:
smallint; Digit: String; PauseAfter: smallint):
smallint;
```

Visual Basic:

```
Function SetCallBusyReturnFlash (Port As Integer, Count
as Integer, Digit as String, PauseAfter as Integer) as
integer
```

Descrição:

Este método configura as opções de flash de retomada em caso de ocupado.

Parâmetros:

Port – Indica o canal da Placa.

Count – Número de flashes

Digit – Dígito após o flash (se existir)

PauseAfter – Pausa após o flash

Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.135 SetCallFlashTime



Configura o flash geral do método MakeCall

Declarações:

Delphi:

```
procedure SetCallFlashTime(Port: Smallint;  
                           FlashTime:smallint): smallint;
```

Visual Basic:

```
Function SetCallFlashTime(Port As Integer, FlashTime as  
                           Integer) as integer
```

Descrição:

Este método configura o tempo de flash a ser utilizado em todas as situações executadas pelo método MakeCall.

Parâmetros:

Port – Indica o canal da Placa.

FlashTime – Tempo de flash em milissegundos.

Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta
fora do intervalo permitido, excedendo o número de portas
instaladas

5.2.136 SetCallNoAnswerPhrase



Configura a frase a ser reproduzida em caso de não atendimento.

Declarações:

Delphi:

```
procedure SetCallNoAnswerPhrase(Port: Smallint;  
                                FileName:  
                                string);
```

Visual Basic:

```
Function SetCallNoAnswerPhrase(Port As Integer,Filename  
as  
                                String) As Integer
```

Descrição:

Este método configura uma frase para ser reproduzida em caso de não atendimento.

Parâmetros:

Port – Indica o canal da Placa.

FileName – Indica o nome do arquivo de voz a ser reproduzido

Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.137 SetCallNoAnswerRingCount



Configura a quantidade de rings a serem considerados como não atendimento

Declarações:

ActiveX:

```
SHORT SetCallNoAnswerRingCount(SHORT Port, SHORT RingCount);
```

Descrição:

Este método configura o tempo de flash a ser utilizado em todas as situações executadas pelo método MakeCall.

Parâmetros:

Port – Indica o canal da Placa.

RingCount – Número de rings.

Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.138 SetCallNoAnswerReturnFlash



Configura o flash em caso de não atendimento.

Declarações:

Delphi:

```
procedure SetCallNoAnswerReturnFlash(Port: Smallint;  
Count: smallint; Digit: String; PauseAfter: smallint):  
smallint;
```

Visual Basic:

```
Function SetCallNoAnswerReturnFlash(Port As Integer,  
Count as Integer, Digit as String, PauseAfter as  
Integer) as integer
```

Descrição:

Este método configura as opções de flash de retomada em caso de não atendimento.

Parâmetros:

- Port** – Indica o canal da Placa.
- Count** – Número de flashes
- Digit** – Dígito após o flash (se existir)
- PauseAfter** – Pausa após o flash

Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.139 SetCallPauseBeforeAnalysis



Configura o tempo de pausa antes de iniciar a supervisão do método MakeCall

Declarações:

Delphi:

```
procedure SetCallPauseBeforeAnalysis(Port: Smallint;  
    PauseBefore:smallint):  
    smallint;
```

Visual Basic:

```
Function SetCallPauseBeforeAnalysis(Port As Integer,  
    PauseBefore as Integer) as  
    integer
```

Descrição:

Este método configura o tempo de pausa antes de se iniciar a supervisão. Pode ser utilizado para evitar false atendimento em algumas situações.

Parâmetros:

Port – Indica o canal da Placa.

PauseBefore – Tempo de flash em milisegundos.

Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.140 SetCallStartFlash



Configura o flash inicial em discagens com flash.

Declarações:

Delphi:

```
procedure SetCallStartFlash(Port: Smallint; Count:
smallint;
Digit: String; PauseAfterFlash:
smallint; ; PauseAfterDigit: smallint): smallint;
```

Visual Basic:

```
Function SetCallStartFlash(Port As Integer, Count as
Integer, Digit as String,
PauseAfterFlash as Integer,
PauseAfterDigit as Integer) as integer
```

Descrição:

Este método configura as opções de flash inicial. É utilizado quando o método MakeCall é chamado com o formato *ctWithFlash*.

Parâmetros:

- Port** – Indica o canal da Placa.
- Count** – Número de flashes
- Digit** – Dígito após o flash (se existir)
- PauseAfterFlash** – Pausa após o flash
- PauseAfterDigit** - Pausa após o dígito

Retorno:

- DG_EXIT_SUCCESS - executado com sucesso
- DG_ERROR_DRIVER_CLOSED - Driver desabilitado
- DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta

fora do intervalo permitido, excedendo o número de portas instaladas

5.2.141 SetCallWaitForDialTone



Indica se o tom de discagem deverá ser aguardado.

Declarações:

ActiveX:

```
SHORT SetCallWaitForDialTone(SHORT Port,  
    VARIANT_BOOL WaitDialTone);
```

Descrição:

Este método indica se o método MakeCall deverá esperar pelo tom de discagem antes de disar. Se não receber este tom, gera o evento OnAfterMakeCall indicando a situação.

Parâmetros:

Port – Indica o canal da Placa.

WaitForDialTone – True/False

Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta
fora do intervalo permitido, excedendo o número de portas
instaladas

5.2.142 SetDetectionType (dg_SetDetectionType)

API

ActiveX

Habilita ou desabilita diversos tipos de detecções (antigo **EnableDetections**)

Declarações:

ActiveX:

```
SHORT SetDetectionType(SHORT Port, SHORT Command,  
SHORT Enable)
```

API:

```
short dg_SetDetectionType(short port, short  
command, short enable);
```

Descrição:

Este é o método que permite habilitar (*Enable=DG_ENABLE*) ou desabilitar (*Enable=DG_DISABLE*) as seguintes detecções, definidas pelo parâmetro **Command**:

- DETECT_OFF - Desabilita todas as detecções independente do parâmetro **Enable**
- DETECT_DTMF - Habilita/Desabilita detecção de DTMF
- DETECT_MFT - Habilita/Desabilita detecção de MFT (MF para "trás")
- DETECT_MFF - Habilita/Desabilita detecção de MFF (MF para "frente")
- DETECT_MF - Habilita/Desabilita detecção de MF definido pelo usuário
- DETECT_TONE1 - Habilita/Desabilita detecção de tom 425Hz puro
- DETECT_TONE2 - Habilita/Desabilita detecção de tom 1100Hz puro

- DETECT_TONE3 - Habilita/Desabilita detecção de tom 2100Hz puro
- DETECT_TONE4 - Habilita/Desabilita detecção de tom puro definido pelo usuário
- DETECT_AUDIO - Habilita/Desabilita detecção de áudio (qualquer coisa diferente de silêncio)
- DETECT_TONE5 - Habilita/Desabilita detecção de tom definido pelo usuário
- DETECT_TONE6 - Habilita/Desabilita detecção de tom definido pelo usuário
- DETECT_TONE7 - Habilita/Desabilita detecção de tom definido pelo usuário
- DETECT_TONE8 - Habilita/Desabilita detecção de tom definido pelo usuário
- DETECT_ALL_MF - Habilita/Desabilita todos os tipos de MF
- DETECT_ALL_TONE - Habilita/Desabilita todos os tipos de tons

Se for passado o comando DETECT_OFF todas as detecções desta porta são desabilitadas. Chamadas consecutivas deste método acumula as detecções, ou seja, é possível habilitar a detecção do TONE1 e TONE3 simultaneamente, por exemplo.

Para fins de supervisão de linha, recomenda-se o uso das funções de CallProgress (consulte o tópico **Supervisão de Linha**) por ser de implementação mais simples.

Os tons definidos pelo usuário devem ser configurados pelo método **SetCardDetections**.

Parâmetros:

Port – Indica o canal da Placa

Command - Valores que indicam qual detecção habilitar, conforme lista acima.

Enable - DG_ENABLE ou DG_DISABLE

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido

5.2.143 SetDialDelays (dg_SetDialDelays)

API

ActiveX

Configura os tempos de pausa dos símbolos de discagem

Declarações:

ActiveX:

```
SHORT SetDialDelays(USHORT CommaDelay, USHORT  
DotDelay, USHORT SemicolonDelay);
```

API:

```
short dg_SetDialDelays(unsigned short  
CommaDelay, unsigned  
short DotDelay, unsigned  
short SemicolonDelay);
```

Descrição:

Quando a função Dial ou MakeCall é chamada, um dos parâmetros passados é uma string contendo uma seqüência de dígitos a serem discados pela placa. Além dos dígitos, é possível passar os caracteres *vírgula*, *ponto* e *ponto-e-vírgula*, cada um representando um determinado tempo de pausa em milissegundos a ser respeitada durante a discagem.

Ex.: Se for discado "123,4", o sistema discará "123", dará uma pausa de **x** milissegundos e depois discará o "4".

Essa função unifica para a API e o ActiveX o acesso às antigas propriedades do ActiveX (*DelayComma*, *DelayDot* e *DelaySemicolon*) que foram retiradas desta versão.

Parâmetros:

CommaDelay – Determina o tempo em milissegundos para a pausa relativa ao caracter "," (vírgula) .

DotDelay – Determina o tempo em milissegundos para a pausa relativa ao caracter "." (ponto) .

SemiColonDelay – Determina o tempo em milissegundos para a pausa relativa ao caracter ";" (ponto-e-vírgula) .

Valores de Retorno:

DG_EXIT_SUCCESS- Executado com sucesso

5.2.144 SetDigitGain(dg_SetDigitGain)

API

ActiveX

Configura ganho/atenuação dos dígitos ou tons gerados

Declarações:

ActiveX:

```
SHORT SetDigitGain(SHORT Card, SHORT Cmd, FLOAT  
GainValue);
```

API:

```
short dg_SetDigitGain(short card, short command,
float gain_value)
```

Descrição:

É possível dar ganho ou atenuar o sinal dos dígitos ou tons gerados pela placa. Esse tipo de configuração afeta todos as portas da placa informada em *card*.

O parâmetro *Cmd/command* determina qual tipo de MF ou tom terá seu ganho alterado. Já o parâmetro *GainValue* fornece o valor em dBm do ganho a ser aplicado.

Parâmetros:

Card – Indica a placa a alterar

Cmd/GainValue - O comando e os valores permitidos de ganho são associados:

Cmd/Command	Descrição	GainValue
DIAL_CFG_GAINDTMF	Ganho dos dígitos DTMFs	-3dBm ate -42 dBm
DIAL_CFG_GAINMFT	Ganho dos MF "pra trás"	-3dBm ate -42 dBm
DIAL_CFG_GAINMFF	Ganho dos MF "pra frente"	-3dBm ate -42 dBm
DIAL_CFG_GAINMF	Ganho dos MF definidos pelo usuário	-3dBm ate -42 dBm
DIAL_CFG_GAINTONE1	Ganho dos tons definidos pelo usuário	+3.17dBm até -42dBm
DIAL_CFG_GAINTONE2	Ganho dos tons definidos pelo usuário	+3.17dBm até -42dBm

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido (Cmd ou GainValue)

5.2.145 SetDigitFrequency(dg_SetDigitFrequency)

API

ActiveX

Configura as frequências que compõem os sinais MF

Declarações:

ActiveX:

```
SHORT SetDigitFrequency(SHORT Card, SHORT Cmd,  
SHORT Frequency,  
CHAR Digit, LONG  
FrequencyValue);
```

API:

```
short dg_SetDigitFrequency(short card, short  
command,  
short freqindex,  
char cDigit, int frequency_value)
```

Descrição:

Os sinais multifrequenciais (MF) têm este nome por serem compostos de mais de uma frequência. Esta função permite configurar as frequências de cada dígito separadamente, dando grande versatilidade. A mudança dessa configuração afeta todas as portas de determinada placa.

Parâmetros:

Card – Placa a ser configurada

Cmd - Indica qual o tipo de MF a ser configurado e pode ser:

DIAL_CFG_FREQDTMF: frequência do sinal de DTMF

DIAL_CFG_FREQMFT: frequência do sinal de MFT

DIAL_CFG_FREQMFF: frequência do sinal de MFF

DIAL_CFG_FREQMF: frequência do sinal de MF (= DTMF) ou configurado pelo usuário

DIAL_CFG_FREQTOM1: frequencia do sinal do TOM1 (425 Hz)

DIAL_CFG_FREQTOM2: frequencia do sinal do TOM2 (425 Hz)

Frequency - Indica qual das duas freqüências será configurada (FREQ1, FREQ2 ou NONE). No caso dos xxxFREQTOM1 e xxxFREQTOM2 este parâmetro é ignorado.

Digit - Dígito a ser configurado (0-9, #, *, A B C D)

FrequencyValue - de 200Hz a 3500Hz

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas

5.2.146 SetDTMFAttenuating (dg_SetDTMFAttenuating)



Indicar o fator de atenuação para as freqüências na geração de tons DTMF.

Declarações:

ActiveX:

```
SHORT SetDtmfAttenuating(LONG HighValue, long  
LowValue)
```

API:

```
short dg_SetDtmfAttenuating(long highvalue, long  
lowvalue)
```

Descrição:

Este método pode receber valores entre 0 e 60 dB (*decibéis*) para atenuação das frequências de DTMF. Normalmente, valores acima de 20 dB de atenuação já tornam o tom inaudível.

Quando um novo valor é atribuído, todos os canais de todas as placas são afetados.

Parâmetros:

HighValue – Valor da atenuação e dB da frequência alta

LowValue - Valor da atenuação e dB da frequência baixa

Valor de Retorno:

EXIT_SUCCESS (0) - executado com sucesso (API)

5.2.147 SetDTMFConfig (dg_SetDTMFConfig)

API

ActiveX

Determina a duração do tom DTMF gerado e da pausa entre dígitos.

Declarações:

ActiveX:

```
SHORT SetDTMFConfig(LONG Duration, LONG Pause)
```

API:

```
short dg_SetDtmfConfig(long duration, long  
pause)
```

Descrição:

O tom tem a duração de 90ms como padrão mas poderá ser alterado conforme a necessidade. A pausa também tem valor padrão em 90ms.

Quando um novo valor é atribuído, todos os canais de todas as placas são afetados.

Parâmetros:

Duration – Duração do dtmf em milissegundos

Pause - Duração da pausa entre dígitos, em milissegundos

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_PARAM_OUTOFRANGE - Duration ou Pause maior que 60000ms

5.2.148 SetE1CRC4Option (dg_SetE1CRC4Option)



Habilita ou desabilita opção de CRC4 no framer E1

Declarações:ActiveX:

```
SHORT SetE1CRC4Option(SHORT Card, SHORT  
E1Device, SHORT Enable)
```

API:

```
short dg_SetE1CRC4Option(short card, short e1,  
short enable)
```

Descrição:

Esta opção permite habilitar ou desabilitar a opção de CRC4 do framer E1, para eventualmente compatibilizar com a central pública. Não é comum ter que utilizar esta opção.

Parâmetros:

Card - Indica qual a placa na qual será configurado este parâmetro

E1Device - Pode assumir os valores **CFG_FRAMER_E1_B** ou **CFG_FRAMER_E1_A**

Enable – **CFG_FRAMER_CRC4_ON** ou **CFG_FRAMER_CRC4_OFF**

Valor de Retorno:

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas

DG_EXIT_SUCCESS - executado com sucesso

5.2.149 SetFastDetection (dg_SetFastDetection)

API

ActiveX

Habilita ou desabilita o algoritmo de detecção rápida de MF.

Declarações:

ActiveX:

```
SHORT SetFastDetection(SHORT Port, SHORT  
Enable);
```

API:

```
short WCDECL dg_SetFastDetection(short port,  
short enable)
```

Descrição:

Este método habilita ou desabilita o algoritmo de detecção rápida de MF, permitindo sua identificação em 20ms. Normalmente, o modo de detecção rápida **não** deve ser utilizado pois em aplicações normais a duração dos MFs é bem maior que 20ms e este algoritmo acarreta num alto consumo de processamento do DSP da placa.

Devido a este alto consumo, não é possível utilizar o modo rápido em conjunto com gravação gsm, por exemplo.

A detecção propriamente dita continua a ser habilitada pelo método **SetDetectionType**. O método **SetFastDetection**, caso seja necessário, deverá ser chamado antes de se habilitar as detecções.

```
short WCDECL dg_SetDetectionType(short port, short  
command, short enable )
```

Parâmetros:

Port - Porta da placa que será aplicada a configuração
Enable - DG_ENABLE ou DG_DISABLE

Valor de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro **Enable** diferente dos valores mostrados acima.
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.150 SetFaxFrequencies (dg_SetFaxFrequencies)

API

ActiveX

Modificar a frequência padrão para detecção de fax

Declarações:ActiveX:

```
SHORT SetFaxFrequencies(SHORT First, SHORT  
Second)
```

API:

```
short dg_SetFaxFrequencies(short first, short  
second)
```

Descrição:

Esta frequência deve ser alterada caso seja necessário detectar alguma frequência fora de padrão. Neste caso o fax não será mais detectado. As frequências padrão de fax são 1100Hz e 2100hz.

Esta função chama, na verdade, os seguintes comandos:

```
dg_SetCardDetections(card,  
CFG_DETECT_FREQTONE2, first,0);  
dg_SetCardDetections(card,  
CFG_DETECT_FREQTONE3, second,0);
```

para todas as placas instaladas.

Caso seja necessário tratar essas frequências de maneira diferente por placa, utilizar o **dg_SetCardDetections** diretamente.

Parâmetros:

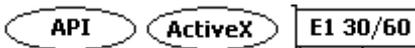
Card - Indica qual a placa na qual será configurado este parâmetro

Frequency – Valor da frequência em Hz.

Valor de Retorno:

DG_EXIT_SUCCESS - Thread iniciada com sucesso
DG_ERROR_DRIVER_CLOSED - Driver não inicializado
DG_ERROR_CARD_OUT_OF_RANGE - Número da placa fora do intervalo permitido
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro **Type** diferente dos valores mostrados acima.

5.2.151 SetFramerLoop (dg_SetFramerLoop)



Coloca os *framers* em loop para fins de testes ou monitoração

Declarações:

ActiveX:

```
SHORT SetFramerLoop(short card, short device,  
short loop_type)
```

API:

```
short dg_SetFramerLoop(short card, short device,  
short loop_type)
```

Descrição:

Os framers são os troncos E1s das placas digitais. Esta função permite *fechar* um looping local ou remoto entre o TX e o RX de cada E1 para fins de testes ou mesmo de monitoração.

Em operação normal, os framers estão configurados como FRAMER_LOOP_OFF. Caso seja necessário efetuar um teste , por exemplo, é possível modificá-lo para FRAMER_LOOP_REMOTE.

OBS: A opção LOOP_REMOTE também é útil na operação de gravação em paralelo quando não se quiser utilizar o conector "T".

Parâmetros:

Card - Indica qual a placa na qual será configurado este parâmetro

Device - Indica qual dos troncos E1 será configurado. Pode assumir **E1_A** ou **E1_B**.

Loop_Type – Tipo de looping:

FRAMER_LOOP_OFF (0) - Sem looping - Opção padrão e utilizada em operações normais

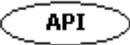
FRAMER_LOOP_REMOTE (1) - Looping remoto - Testes de Linha ou gravação em paralelo

FRAMER_LOOP_LOCAL (2) - Looping local - Utilizado somente em testes de assistência técnica

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card fora do intervalo de placas instaladas
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido (device ou loop_type)

5.2.152 SetFrequency (dg_SetFrequency)

APIActiveX

Modificar a frequência padrão de supervisão de tons do tom CFG_DETECT_FREQTONE1.

Declarações:

ActiveX:

```
SHORT SetFrequency(SHORT Card, SHORT Frequency)
```

API:

```
short dg_SetFrequency(short card, short  
Frequency)
```

Descrição:

A frequência base para detecção dos tons de chamada, ocupado, etc... é 425hz. Algumas centrais trabalham com valores diferentes do padrão. Alterando esta frequência as placas Digivoice passarão a monitorar outra faixa de tons. **Ao atribuir este valor, todos os canais de todas as placas são afetados.**

Este método é mantido para fins de compatibilidade e para facilitar a mudança dessa frequência, que é a mais utilizada. Internamente ele faz chamada ao método

dg_SetCardDetections(card, CFG_DETECT_FREQTONE1, Frequency, 0);

Parâmetros:

Card - Indica qual a placa na qual será configurado este parâmetro

Frequency – Valor da frequência. O padrão é 425hz.

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro **card** fora do intervalo permitido, excedendo o número de placas instaladas

DG_ERROR_PARAM_OUTOFRANGE -Valor da frequência fora do intervalo entre **0** e **3000hz**

5.2.153 SetGSMMode (dg_SetGSMMode)

API

ActiveX

Modificar o formato padrão dos arquivos GSM

Declarações:

ActiveX:

SHORT SetGSMMode (SHORT Mode)

API:

short dg_SetFrequency(short mode)

Descrição:

Este método configura a VoicerLib se deverá ser utilizado o GSM compatível com o Asterisk(c) (GSM_RAW) ou o GSM padrão da Digivoice (GSM_DIGIVOICE). A diferença entre os dois formatos consiste apenas na existência de um cabeçalho no padrão Digivoice. A codificação em si é a mesma e por isso não foi criado um novo formato de gravação. Este método afeta todas as portas de todas as placas e o padrão assumido atualmente é o GSM_RAW que não contém cabeçalho no arquivo.

Este método, se for necessário, pode ser chamado logo após iniciar a VoicerLib.

Parâmetros:

Mode - GSW_RAW ou GSM_DIGIVOICE

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

5.2.154 SetH100 (dg_SetH100)



Altera configurações de sincronismo das placas E1

Declarações:

ActiveX:
SHORT SetH100Sync (SHORT Card,

```
SHORT H100Type,
SHORT Param1,
SHORT Param2)
```

API:

```
short dg_SetH100Sync(unsigned short card,
                    unsigned char h100type,
                    unsigned char param2,
                    unsigned char param3)
```

Descrição:

Esta função permite configurar o funcionamento de sincronismo dos *Framers* E1. Como seu funcionamento é bastante complexo, recomenda-se utilizar a função **SetCardSyncMode** para configurar os sincronismos das placas.

Parâmetros:

Card – Indica a placa (1 a *n*)

H100Type - Indica o tipo de configuração:

- H100_MASTER
- H100_SLAVE
- H100_CT_NETREF
- H100_SCBUS
- H100_MVIP90
- H100_HMVIP
- H100_TERMINAL
- H100_STREAM_RATE

Os parâmetros são relativos à H100Type e seguem a seguinte tabela:

H100type / param1 / param2

```
-> H100_MASTER
    |_____ H100_MS_SEL
        |_____ H100_MS_DISABLE=0
        |_____ H100_MS_CT_A=1
        |_____ H100_MS_CT_B=3
```

```
|_____ H100_MS_REF
|_____ H100_MS_NONE=0
|_____ H100_MS_CT_NETREF1=6
|_____ H100_MS_CT_NETREF2=7
|_____ H100_MS_L_NETREF0=8
|_____ H100_MS_L_NETREF1=,
|_____ H100_MS_L_NETREF2=10
```

```
|_____ H100_MS_REFFR
|_____ H100_MS_8KHz=0
|_____ H100_MS_1536MHz=1
|_____ H100_MS_1544MHz=2
|_____ H100_MS_2048MHz=3
```

-> H100_SLAVE (param2 not used)

```
|_____ H100_SL_CT_A
|_____ H100_SL_CT_B
|_____ H100_SL_SCBUS
|_____ H100_SL_MVIP
|_____ H100_SL_LOCAL0
|_____ H100_SL_LOCAL1
```

-> H100_CT_NETREF

```
|_____ H100_REF_SOURCE1
|_____ H100_REF_NONE=0
|_____ H100_REF_L_NETREF0=8
|_____ H100_REF_L_NETREF1=9
|_____ H100_REF_L_NETREF2=10
|_____ H100_REF_SOURCE2
|_____ H100_REF_NONE=0
|_____ H100_REF_L_NETREF0=8
|_____ H100_REF_L_NETREF1=9
|_____ H100_REF_L_NETREF2=10
|_____ H100_REF_DIV1
|_____ H100_REF_D1
|_____ H100_REF_D192
|_____ H100_REF_D193
|_____ H100_REF_D256
|_____ H100_REF_DIV2
```

```
|_____ H100_REF_D1
|_____ H100_REF_D192
|_____ H100_REF_D193
|_____ H100_REF_D256
```

-> H100_SCBUS

```
|_____ H100_SC_SEL
|_____ H100_SC_ENABLE=1
|_____ H100_SC_DISABLE=0
|_____ H100_SC_FREQ
|_____ H100_SC_2048
|_____ H100_SC_4096
|_____ H100_SC_8192
```

-

> H100_MVIP90 (param
2 not used)

```
|_____ H100_MV_ENABLE=1
|_____ H100_MV_DISABLE=0
```

-

> H100_HMVIP (param
2 not used)

```
|_____ H100_HMV_ENABLE=1
|_____ H100_HMV_DISABLE=0
```

-

> H100_TERMINAL (param
2 not used)

```
|_____ H100_TERM_ENABLE=1
|_____ H100_TERM_DISABLE=0
```

-> H100_STREAM_RATE

```
|_____ H100_ST_0_3=0
|_____ H100_ST_2048
|_____ H100_ST_4096
|_____ H100_ST_8192
|_____ H100_ST_4_7=2
|_____ H100_ST_2048
```

```
|_____ H100_ST_4096
|_____ H100_ST_8192
|_____ H100_ST_8_11=4
|_____ H100_ST_2048
|_____ H100_ST_4096
|_____ H100_ST_8192
|_____ H100_ST_12_15=6
|_____ H100_ST_2048
|_____ H100_ST_4096
|_____ H100_ST_8192
```

Valores de Retorno:

DG_EXIT_SUCCESS - Executado com sucesso
DG_ERROR_CARD_OUT_OF_RANGE - Parâmetro card
fora do intervalo de placas instaladas
DG_FEATURE_NOT_SUPPORTED - Comando não
suportado por este tipo de placa

5.2.155 SetPlayFormat (dg_SetPlayFormat)

API

ActiveX

Especifica o formato de reprodução de uma porta.

Declarações:ActiveX:

```
SHORT SetPlayFormat(SHORT Port, SHORT FileFormat)
```

API:

```
short dg_SetPlayFormat(short port, enum  
EnumFileFormat)
```

Descrição:

O método SetPlayFormat permite indicar formatos de reprodução diferentes por canal, os outros canais assumirão o formato especificado na propriedade FileFormat, que determina o formato para todos os canais. Como o novo formato GSM oferece uma qualidade de áudio inferior ao Wave pode ser necessário gravar em GSM e reproduzir mensagens no formato Wave. Com este método é possível manter formatos de reprodução diferentes no mesmo canal ou em canais distintos.

O PlayFile detecta automaticamente o tipo de arquivo, baseado na extensão para .sig, .gsm e se for wave, detecta se é PCM ou LeiMi. Isso permite uso de diversos formatos para reprodução sem preocupação para o programador. Se for utilizado uma extensão desconhecida, fica valendo o que está especificado no SetPlayFormat.

Parâmetros:

Port – Indica o canal da Placa.

FileFormat – Formato para reprodução.

0 - ffWaveULaw (Lei *mi*)

1 - ffSig - obsoleto - é forçado formato
ffWaveULaw

2 - ffWavePCM

3 - ffGsm610

4 - ffWaveALaw (Lei A)

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro fora do
intervalo permitido, excedendo o número de portas instaladas

5.2.156 SetPortGain (dg_SetPortGain)

API

ActiveX

Especifica o ganho para a uma determinada porta.

Declarações:

ActiveX:

```
SHORT SetPortGain(SHORT Port, SHORT txrx, SHORT Value)
```

API:

```
short dg_SetPortGain(short port, short rx_tx, short value)
```

Descrição:

Esta função permite dar ganho/atenuação em uma determinada porta, no RX e TX separadamente. Com isso, é possível controlar a qualidade do áudio que se "escuta" ou o que se "fala". O padrão é ganho zero. Cada 6dB a menos equivale à metade de ganho e a cada 6dB a mais equivale ao dobro.

Parâmetros:

Port – Indica o canal da Placa.

rx_tx – Indica se o valor será alterado no RX ou no TX. Para isso pode-se utilizar as constantes TX_GAIN (0) ou RX_GAIN (1).

Value – Valor de ganho a ser passado na função, podendo variar de -42 dBm até +12 dBm.

Valores de Retorno:

EXIT_SUCCESS - Executado com sucesso

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetros **port**

fora do intervalo especificado.

DG_ERROR_PARAM_OUTOFRANGE - Este erro pode ocorrer caso se especifique **Value** maior que 12 ou menor que -42, ou parâmetro **txrx** diferente do previsto

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

5.2.157 SetPortID (dg_SetPortID)



Especifica a identificação de uma porta no tronco digital quando se utiliza uma das threads de controle.

Declarações:

ActiveX:

```
SHORT SetPortID(SHORT Port, LPCTSTR ID);
```

API:

```
short dg_SetPortId(short port, char *szID);
```

Descrição:

Utilizando-se a *thread* de controle E1, esta função permite identificar cada canal com um número de identificação diferente a ser fornecido ao destino como BINA.

Parâmetros:

Port – Indica o canal da Placa.

ID – String contendo o número de identificação do canal

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

5.2.158 SetPortChatLog(dg_SetPortChatLog)

API

ActiveX

E1 30/60

Conecta uma conferência a uma porta específica para fins de gravação.

Declarações:

ActiveX:

```
SHORT SetPortChatLog(SHORT Port, SHORT ChatRoom,  
SHORT Enable);
```

API:

```
short dg_SetPortChatLog(short port , long  
ChatRoom, short enable);
```

Descrição:

Esta função deverá ser utilizada quando se deseja gravar uma sala de conferência (chat room) que pode ser uma conferência de apenas 2 canais (utilizada na gravação em paralelo) ou uma com várias portas conectadas.

Quando uma conferência (chat room) é criada através da função **CreateChatRoom (dg_CreateChatRoom)** ela recebe um *handle* que é um identificador único para esta sala. Este handle é o que deverá passado no parâmetro *ChatRoom*. A

eventual gravação da conferência indicada por *ChatRoom* deverá ser iniciada na porta *port* que é a porta que *ouve* todas as outras da sala de conferência. *Port* não pode fazer parte da conferência.

O parâmetro *Enable* pode receber *DG_ENABLE*, para ligar a porta à sala ou *DG_DISABLE* para desligá-la

Parâmetros:

Port – Indica o canal da Placa que gravará a conferência.

ChatRoom – Indicador da sala de conferência criada por

CreateChatRoom

Enable – *DG_ENABLE* ou *DG_DISABLE*

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_EXIT_FAILURE - Erro ao tentar executar o comando

5.2.159 SetRecordGain (dg_SetRecordGain)

API

ActiveX

Especifica o ganho para a gravação.

Declarações:ActiveX:

```
SHORT SetRecordGain(SHORT Port, SHORT Gain)
```

API:

```
short dg_SetRecordGain(short port, short gain)
```

Descrição:

Especifica o ganho para a gravação de uma porta independente. O ganho de gravação aqui poder variar de -42dB até +12dB. Nesta opção, a função é apenas uma interface para a função **SetPortGain**, aplicando ganho apenas no RX. O padrão do ganho é zero.

Como o ganho interferirá na qualidade do audio, a checagem de intervalo faz com que um **Gain** menor que -40dB seja utilizado - 40dB. E **Gain** maior que +12dB é fixado em +12dB.

Parâmetros:

Port – Indica o canal da Placa.

Ganho – Valores conforme o tipo da placa (-40 -> +12)

Valores de Retorno:

DG_EXIT_SUCCESS - Executado com sucesso

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetros fora do intervalo especificado.

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

5.2.160 SetRecordFormat (dg_SetRecordFormat)

API

ActiveX

Especifica o formato de gravação de uma porta

Declarações:

ActiveX:

```
SHORT SetRecordFormat(SHORT Port, SHORT  
FileFormat)
```

API:

```
short dg_SetRecordFormat(short port, enum  
EnumFileFormat)
```

Descrição:

O método SetRecordFormat permite indicar formatos de gravação diferentes por canal, os outros canais assumirão o formato especificado na propriedade FileFormat, que determina o formato para todos os canais. Como o novo formato GSM oferece uma qualidade de áudio inferior ao Wave pode ser necessário gravar em GSM e reproduzir mensagens no formato Wave. Com este método é possível manter formatos de gravação diferentes no mesmo canal ou em canais distintos.

É possível alterar o formato de gravação sem a necessidade de chamar o método DisableInputBuffer porém não é possível fazê-lo com uma gravação em curso.

Parâmetros:

Port – Indica o canal da Placa.

FileFormat – Formato para gravação.

0 - ffWave

1 - ffSig obsoleto - é forçado formato ffWaveULaw

2 - ffWavePCM

3 - ffGsm610

4 - ffWaveALaw (Lei A)

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_REC_ALREADY_RECORDING - Operação não permitida pois a porta está gravando

5.2.161 SetSilenceThreshold (dg_SetSilenceThreshold)

API

ActiveX

Permite definir o limiar de silêncio relacionado às detecções de dígitos.

Declarações:

ActiveX:

```
SHORT SetSilenceThreshold(SHORT Port, SHORT Value)
```

API:

```
short dg_SetSilenceThreshold(short port, short value)
```

Descrição:

O limiar de silêncio define, de modo simplificado, como a VoicerLib tratará os dígitos a serem detectados (DTMF, MFF, MFT, etc...). O parâmetro *value* pode variar de **0** até **-42** dBm. O valor padrão é -30 dBm mas este valor pode ser alterado. Quanto mais próximo de zero, menos sensível será a detecção.

Este limiar de silêncio tem resolução de 3dB, portanto, -42dB e -39dB tem o mesmo efeito na placa.

Parâmetros:

Port – Indica o canal da Placa

Value - Valor em dBm, variando de 0 até **-42**.

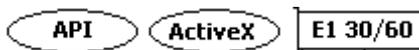
Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro fora do intervalo permitido ($-42 < \text{Value} < 0$)

5.2.162 SetStartE1RxCount (dg_SetStartE1RxCount)

Configura o número inicial de dígitos a serem recebidos na thread E1

Declarações:ActiveX:

```
SHORT SetStartE1RxCount(SHORT Port, SHORT Count);
```

API:

```
short dg_SetStartE1RxCount(short port, short count);
```

Descrição:

Esta função é destinada a permitir o controle "manual" da troca

de sinalização R2D acontecendo em uma determinada porta. Muitas vezes uma aplicação precisa ir analisando os dígitos recebidos um a um, permitindo algum tipo de encaminhamento específico, análise de rotas, etc... Ela configura o número inicial de dígitos que a *thread* E1 receberá antes de gerar o evento EV_E1CHANGESTATUS com status C_NUMBER_RECEIVED.

Utilize apenas esta função caso seja necessário o tipo controle explicado. Caso seja utilizada uma sinalização tradicional, dê preferências às configurações da função **ConfigE1Thread**.

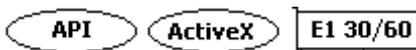
Parâmetros:

Port – Indica o canal da Placa
Count - Número de dígitos a serem esperados

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro count fora do intervalo permitido ($0 < \text{count} < 30$)

5.2.163 SetNextE1RxCount (dg_SetNextE1RxCount)



Configura os próximos dígitos a serem recebidos na thread E1

Declarações:

ActiveX:

```
SHORT SetNextE1RxCount(SHORT Port, SHORT Count);
```

API:

```
short dg_SetNextE1RxCount(short port, short  
count);
```

Descrição:

Complementar à **SetStartE1RxCount**, esta função é destinada a permitir o controle "manual" da troca de sinalização R2D acontecendo em uma determinada porta. Esta função configura os próximos dígitos que a *thread* E1 receberá antes de gerar o evento EV_E1CHANGESTATUS com status C_NUMBER_RECEIVED.

Utilize apenas esta função caso seja necessário o tipo controle explicado. Caso seja utilizada uma sinalização tradicional, dê preferências às configurações da função **ConfigE1Thread**.

Parâmetros:

Port – Indica o canal da Placa

Count - Número de dígitos a serem esperados

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas

DG_ERROR_PARAM_OUTOFRANGE - Parâmetro count fora do intervalo permitido ($0 < \text{count} < 30$)

DG_ERROR_THREAD_NOT_RUNNING - Thread E1 não foi iniciada pelo CreateE1Thread

5.2.164 SetTwist (dg_SetTwist)

API

ActiveX

Configura os níveis de rigor para a detecção de dígitos

Declarações:

ActiveX:

```
SHORT SetTwist(SHORT Port, SHORT Twist1, SHORT  
Twist2 );
```

API:

```
short dg_SetTwist(short port, short twist1,  
short twist2);
```

Descrição:

A VoicerLib procura, como padrão, manter uma boa relação entre uma boa detecção de dígitos sem, no entanto, pegar talkoffs (falsos dígitos). Isso é feito através de cálculos entre as frequências encontradas nos DTMFs.

O **Twist1** é a diferença máxima aceita entre a primeira e segunda frequências tem valor padrão de 9dB. Já o **Twist2** é a diferença mínima das duas primeiras frequências com uma eventual terceira frequência e tem valor padrão de 15dB.

No Twist1, quanto **maior** o valor, **menor** é o rigor na detecção. No Twist2, quanto **maior** o valor, **maior** é o rigor.

Parâmetros:

Port – Indica o canal da Placa

Twist1 - Diferença em dB entre a primeira e segunda frequência de DTMF -> maior valor, *menor* rigor

Twist2 - Diferença entre as primeiras duas frequências e uma

terceira -> maior valor, *mais* rigor

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro porta fora do intervalo permitido, excedendo o número de portas instaladas
DG_ERROR_PARAM_OUTOFRANGE - Parâmetro count fora do intervalo permitido ($0 < \text{count} < 30$)
DG_ERROR_THREAD_NOT_RUNNING - Thread E1 não foi iniciada pelo CreateE1Thread

5.2.165 ShutdownVoicerLib (dg_ShutdownVoicerLib)



Finaliza a comunicação da placa com a aplicação.

Declarações:

ActiveX:
`SHORT ShutdownVoicerLib(void);`

API:
`short dg_ShutdownVoicerLib(void);`

Descrição:

Este método faz com que se finalize a comunicação da placa

com a aplicação. Logo, a sua chamada deverá ser a última coisa que o programador deverá fazer antes de sair da aplicação. Caso a aplicação seja encerrada sem que o este método tenha sido chamado, o micro corre o risco de "travar" a qualquer momento pois recursos de memória e interrupção ficam ativos. Então, **NUNCA FECHER A APLICAÇÃO ANTES DE CHAMAR ESTE MÉTODO.**

Valores de Retorno:

Retornos:

DG_EXIT_SUCCESS - se executado com sucesso
DG_ERROR_DRIVER_CLOSED - A VoicerLib já está finalizada (não é necessariamente um erro, pode ser simplesmente ignorado).

5.2.166 StartVoicerLib (dg_StartVoicerLib)



Inicializa a comunicação da placa com a aplicação.

Declarações:

ActiveX:
SHORT StartVoicerLib(void)

API:
short dg_StartVoicerlib(char *szConfigPath)

Descrição:

Este método faz com que se inicie a comunicação da placa com a aplicação. A sua chamada deverá ser a primeira coisa que o programador deverá fazer na aplicação.

No ActiveX, pode-se preencher a propriedade **ConfigPath** com o caminho dos arquivos de configuração com extensão **.i00**. Caso esta propriedade esteja vazia a VoicerLib assumirá como padrão o diretório **Arquivos de ProgramasVoicerLib4**. A mesma regra se aplica ao parâmetro **szConfigPath** na API.

Sempre deverá ser testado os valores de retorno para poder diagnosticar algum problema no caso do valor ser diferente de **DG_EXIT_SUCCESS**.

Valores de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_MEMORY_ALLOCATION - Memória insuficiente para iniciar os serviços
DG_ERROR_DRIVER_ALREADY_OPEN - Driver já está inicializado
DG_ERROR_READING_PORTCOUNT - O número de portas lido é inconsistente, muito provavelmente causado por problemas de I/O no barramento PCI
DG_ERROR_LOADING_DEVICEDRIVER - Erro ao iniciar o device driver. Só acontece no caso de uma instalação incompleta ou corrompida.
DG_ERROR_CREATING_EVENT - Erro ao criar eventos de controle da VoicerLib
DG_ERROR_COULD_NOT_CREATE_THREAD - Erro ao criar threads de controle.
DG_ERROR_FIRMWARE_NOT_FOUND - Arquivo de firmware não foi encontrado na pasta indicada por **ConfigPath**.
DG_ERROR_FIRMWARE_IO_TIMEOUT - Não foi possível carregar o firmware corretamente na placa. Isto pode ser causado por problemas de hardware na placa ou mesmo

no slot PCI do computador.

Veja Também: [ShutdownVoicerLib](#)

5.2.167 StopPlayBuffer (dg_StopPlayBuffer)

API

ActiveX

Interrompe a reprodução iniciada pelo PlayBuffer.

Declarações:

ActiveX:

SHORT StopPlayBuffer(SHORT Port)

API:

short dg_StopPlayBuffer(short port)

Descrição:

Este método envia para a placa um comando dizendo para que a reprodução dos seus buffers internos seja interrompida. Deve ser chamado sempre quando não há mais amostras a serem enviadas pelo **PlayBuffer**.

Parâmetros:

Port – Indica a porta da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_ALREADY_PLAYING - A porta está ocupada por uma reprodução de arquivo.

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro fora do intervalo permitido

5.2.168 StopPlayFile (dg_StopPlayFile)

API

ActiveX

Interrompe a reprodução de um arquivo de áudio.

Declarações:

ActiveX:

```
SHORT StopPlayFile(SHORT Port)
```

API:

```
short dg_StopPlayFile(short port)
```

Descrição:

Este método interrompe a reprodução de um arquivo de áudio. Neste caso, o evento OnPlayStop é gerado e o parâmetro Status receberá o valor ssStopped.

Parâmetros:

Port – Indica a porta da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso

DG_ERROR_DRIVER_CLOSED - Driver desabilitado

DG_ERROR_NOT_PLAYING - A porta não está reproduzindo nada

DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro fora do intervalo permitido

5.2.169 StopRecordFile (dg_StopRecordFile)

Interrompe a gravação de um arquivo.

Declarações:

ActiveX:

```
SHORT StopRecordFile(SHORT Port);
```

API:

```
short dg_StopRecordFile(short port);
```

Descrição:

Este método interrompe a gravação de um arquivo sem cancelar o envio de amostras para a VoicerLib.

Chamando este método o evento **OnRecordStop** é gerado e o parâmetro **Status** receberá o valor **ssStopped**.

Para cancelar o envio de amostras, o método **DisableInputBuffer** deve ser chamado.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

DG_EXIT_SUCCESS - executado com sucesso
DG_ERROR_DRIVER_CLOSED - Driver desabilitado
DG_ERROR_PORT_OUT_OF_RANGE - Parâmetro fora do intervalo permitido
DG_ERROR_THREAD_NOT_RUNNING - O envio de amostras da placa não foi habilitado e, portanto não há gravação para finalizar

5.3 Eventos

Conceitualmente, **eventos** e **mensagens** se referem à mesma coisa, ou seja, são *avisos* que a VoicerLib repassa para a aplicação, indicando o acontecimento de algo ou resposta de alguma ação iniciada através de comando do programa.

No ActiveX, estes *avisos* são chamados de **eventos**, termo usual aos programadores que utilizam ferramentas visuais, como Delphi ou Visual Basic. Já o termo **mensagem** é mais genérico e conhecido dos programadores habituados a utilizar C/C++.

Normalmente, quando um evento é tratado no programa, a ferramenta visual gera uma rotina que será responsável por tratar este evento. Todas estas rotinas assumirão o nome padrão Onxxxx onde o xxxx indicará o tipo do evento. Estas rotinas receberão parâmetros padrões, de acordo com a funcionalidade de cada evento. Alguns recebem apenas a porta, outros além da porta, recebem um *estado* ou qualquer outro tipo de identificação. Neste guia de referência, estes parâmetros serão relacionados no item **Parâmetros Recebidos** de cada evento.

Nas funções de API, a VoicerLib repassa os eventos que

ocorrem na placa através das mensagens que devem ser tratadas em uma rotina específica, que foi chamada **ReceiveEvents** (leia o capítulo "*Conceitos Basicos - API*"). Neste **Guia de Referência** os dados referentes ao *context_data* são relacionados no item **Context_Data (API)**.

5.3.1 OnAfterDial (EV_AFTERDIAL)

Ocorre ao término de uma discagem feita por um único comando Dial.

Descrição:

Para efetuar uma discagem, é necessário utilizar o método Dial, que pode discar de um a 28 números. Visando facilitar o controle de fluxo do programa, o evento AfterDial ocorrerá tão logo todos os dígitos passados como parâmetro no método Dial sejam discados pela placa. Se for utilizado o parâmetro PauseAfterDial do método Dial, o evento OnAfterDial só será gerado após a discagem mais a pausa.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

Context_Data (API):

command - EV_AFTERDIAL (0x2b)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.2 OnAfterFlash (EV_AFTERFLASH)

Ocorre ao término de um comando Flash

Descrição:

De maneira semelhante ao evento OnAfterDial, o evento OnAfterFlash ocorrerá ao término do comando executado pelo método Flash.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

Context_Data (API):

command - EV_AFTERFLASH (0x2c)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.3 OnAfterMakeCall

Ocorre ao término da função de discagem MakeCall.

Descrição:

Ocorre ao término da discagem com supervisão quando utilizado o MakeCall, disponível somente no ActiveX.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Status – Pode assumir:

- **mNoDialTone** – Sem tom de discagem
- **mkDelivered** – Ligação entregue sem supervisão
- **mkNoAnswer** – Ligação não foi atendida
- **mkBusy** – Ligação ocupada
- **mkAnswered** - Ligação atendida
- **mkAborted** – ligação cancelada pelo AbortCall
- **mkDialToneAfterDial** - Indica recebimento de tom de linha depois da discagem. Normalmente esta situação indica algum problema com o ambiente (PABX, linha, etc...)
- **mkFaxDetected** - Indica detecção de fax.

5.3.4 OnAfterPickUp (EV_AFTERPICKUP)

Ocorre ao término de um comando PickUp

Descrição:

De maneira semelhante ao evento OnAfterDial, o evento OnAfterPickup ocorrerá após a pausa determinada na chamada do método PickUp.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

Context_Data (API):

command - EV_AFTERPICKUP (0x2d)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.5 OnAnswerDetected (EV_ANSWERED)

Ocorre quando a placa detecta atendimento da ligação originada.

Descrição:

A placa possui recursos de monitoração do status da linha à ela conectada, e de repassá-los para a aplicação. Para tanto, ela deve ser habilitada a utilizar estes recursos.

Nas placas FXO, o programador tem a flexibilidade de a qualquer ponto da aplicação habilitar e desabilitar tais recursos, fazendo uso dos métodos **EnableAnswerDetection** para habilitar e **DisableAnswerDetection** para desabilitar.

Quando uma ligação é originada pela placa, e os recursos citados estiverem habilitados, o evento **OnAnswerDetected** ocorrerá no momento em que essa ligação for atendida. Isto é válido também para a placa digital, pois ao receber o R2D referente à atendimento, a VoicerLib também gera o evento.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

AnswerType - Indica se o atendimento foi por áudio (AUDIO_DETECTED) ou por timeout (TIMEOUT_DETECTED).

Context_Data (API):

command - EV_ANSWERED (0x1f)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.6 OnAudioSignalDetected (EV_AUDIO_SIGNAL)

Ocorre quando a porta recebe um sinal de áudio, MF, tom, etc...

Descrição:

Este evento tem por finalidade facilitar a análise dos sinais recebidos e suas respectivas cadências pois repassa para a aplicação todos os sinais recebidos, sem tratamento nenhum.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

SignalCode: Indica o sinal de áudio recebido:

CP_SILENCE=20H
CP_AUDIO=21H
CP_TONE1=22H
CP_TONE2=23H
CP_TONE3=24H
CP_TONE4=25H
CP_TONE5=26H
CP_TONE6=27H
CP_TONE7=28H
CP_TONE8=29H
CP_UNDEFINED = 0x40

Context_Data (API):

command - EV_AUDIO_SIGNAL (0x3e)

port/card - Indica a porta que gerou o evento

data: mesmos valores do parâmetro **Value** explicado acima

data_aux: não utilizado

card: Indica a Placa que gerou o evento

5.3.7 OnBusyDetected (EV_BUSY)

Ocorre quando a placa detecta tom de ocupado na linha.

Descrição:

Nas placas analógicas, o programador tem a flexibilidade de a qualquer ponto da aplicação habilitar e desabilitar tais recursos, fazendo uso dos métodos EnableCallProgress para habilitar e DisableCallProgress para desabilitar.

Sempre quando a placa tomar a linha conectada à ela, para originar uma ligação, e receber tom de ocupado, o evento OnBusyDetected ocorrerá, desde que os recursos de monitoração estejam habilitados. Isto é válido também para a placa digital, pois ao receber o R2D referente à bloqueio, não disponibilidade do canal, etc... a VoicerLib também gera o evento.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Context_Data (API):

command - EV_BUSY (0x21)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.8 OnCallerID (EV_CALLERID)

Ocorre quando a placa detecta o identificador de A.

Descrição:

Este evento é gerado quando o identificador de A é recebido pelo tronco.

Na API é possível ler o número através da função **dg_GetCallerID**.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Número– Número identificado exatamente como enviado pela companhia telefônica.

Context_Data (API):

command - EV_CALLERID (0x32)

port - Indica o canal da Placa que gerou o evento

data: Número de dígitos disponíveis para leitura

data_aux: não utilizado

card: não utilizado

5.3.9 OnCalling (EV_CALLING)

Ocorre quando a placa detecta tom chamada na linha.

Descrição:

Nas placas analógicas, o programador tem a flexibilidade de a

qualquer ponto da aplicação habilitar e desabilitar tais recursos, fazendo uso dos métodos `EnableCallProgress` para habilitar e `DisableCallProgress` para desabilitar.

Sempre quando a placa tomar a linha conectada à ela, originar uma ligação e receber tom de chamada, o evento `OnCalling` ocorrerá, desde que os recursos de monitoração estejam habilitados. Isto é válido também para a placa digital, pois ao finalizar a troca de sinalização R2D MFC, a VoicerLib também gera o evento na cadência 1x4 (1 segundo de tom para 4 de intervalo).

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Context_Data (API):

command - EV_CALLING (0x1d)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.10 OnCallStateChange

Ocorre em várias situações durante o `MakeCall`.

Descrição:

O método `MakeCall` inicia uma discagem completa, com supervisão, frase inicial, discagem, etc... O evento `OnCallStateChange` indica, por canal, em qual etapa o `MakeCall` está no momento. Este evento deve ser utilizado para fins de

monitoramento e depuração de programas.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

Status - Indica o estado atual do MakeCall, podendo assumir os seguintes valores:

- csPlayingStartPhrase - Falando frase inicial
- csPickingUp - Iniciando discagem sem transf
- csInitialFlash - Dando flash inicial
- csDialingPrefix - Prefixo apos o flash
- csStartAnalysis - Iniciando supervisão
- csNoAnswerReturn - Retomada em caso de não atendimento
- csPlayingBusyPhrase - Falando frase de ret. caso ocupado
- csPlayingNoAnswerPhrase - Falando frase de ret. caso ocupado
- csWaitingDialTone - Esperando tom de discagem
- csDialing - Discando
- csCalling - Chamando

5.3.11 OnDialToneDetected (EV_DIALTONE)

Ocorre quando a placa detecta tom de discagem na linha.

Descrição:

Nas placas analógicas, o programador tem a flexibilidade de a qualquer ponto da aplicação habilitar e desabilitar tais recursos, fazendo uso dos métodos EnableCallProgress para habilitar e DisableCallProgress para desabilitar. Também é necessário configurar a frequência de tons de acordo com a Central PABX utilizada (**SetFrequency**).

Sempre quando a placa tomar a linha conectada à ela, para originar uma ligação, e receber um tom de discagem, o evento OnDialToneDetected ocorrerá.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Context_Data (API):

command - EV_DIALTONE (0x1e)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.12 OnDigitDetected (EV_DTMF)

Ocorre sempre que a placa detecta um dígito.

Descrição:

Toda vez que a placa detectar um dígito (DTMF, MFF, MFT, etc), o evento OnDigitDetected ocorrerá, e o código ASCII dígito será passado através do parâmetro Digit.

Um dígito DTMF é um par de frequências pré-definidas e na faixa da voz. O programador deve estar ciente que a placa sempre está habilitada a detectar os dígitos, logo, durante uma conversação é comum a placa detectar alguns dígitos indevidamente, pois ela analisa todo o conteúdo de frequência da voz dos locutores e toda vez em que ela encontrar na voz um par de frequências que coincidam com um dígito DTMF, o

evento OnDigitDetected ocorrerá. Quando o programador quiser validar os dígitos apenas em certos momentos, deve-se fazer uso do método GetDigits e do evento OnDigitReceived ao invés de tratar dígito a dígito.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento
Digit – Código ASCII do dígito recebido

Context_Data (API):

command - EV_DTMF (0x1c)
port - Indica o canal da Placa que gerou o evento
data: Código ASCII do dígito recebido
data_aux: não utilizado
card: não utilizado

5.3.13 OnDigitsReceived (EV_DIGITSRECEIVED)

Ocorre sempre em resposta ao método GetDigits(dg_GetDigits) é chamado.

Descrição:

Este evento ocorre em resposta ao método GetDigits. Basicamente este método espera um determinado número de dígitos e/ou finalizador por um intervalo de tempo. Os dígitos detectados devem ser recuperados através da função **ReadDigits (dg_ReadDigits)**.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Status – Indica qual das situações gerou o evento

OnDigitsReceived, a saber:

EdMaxDigits(2) – Recebeu o número máximo de dígitos estipulado.

EdTermDigit(5) – Recebeu o dígito finalizador

EdDigitTimeout(3) – Estourou o tempo total para entrada de todos os dígitos.

EdInterDigitTimeOut(4) – Estourou o tempo inter-dígito.

EdDigitOverMessage – Detectou dígito durante a reprodução.

Context_Data (API):

command - EV_DIGITSRECEIVED (0x31)

port - Indica o canal da Placa que gerou o evento

data: Código ASCII do dígito recebido

- **EdMaxDigits(2)** – Recebeu o número máximo de dígitos estipulado.
- **EdTermDigit(5)** – Recebeu o dígito finalizador
- **EdDigitTimeout(3)** – Estourou o tempo total para entrada de todos os dígitos.
- **EdInterDigitTimeOut(4)** – Estourou o tempo inter-dígito.
- **EdDigitOverMessage** – Detectou dígito durante a reprodução.

data_aux: não utilizado

- **card**: não utilizado

5.3.14 OnErrorDetected (EV_ERRORDETECTED)

Ocorre quando acontece algum erro de hardware

Descrição:

Este erro só deverá acontecer se houver alguma falha de

hardware, que pode ser causada por falhas de instalação ou conflitos com outras placas instaladas.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

ErrorType: Indica o tipo de erro, podendo assumir:

- ERROR_FIFO_FULL (0x79)
- ERROR_READ_FAILURE (0x80)
- ERROR_CARD_RESET (0x99)
- ERROR_CMD_OVERFLOW (0x98)

Context_Data (API):

command - EV_ERRORDETECTED (0x39)

port/card - Indica a Placa que gerou o evento

data: mesmos valores do parâmetro ErrorType explicado acima

data_aux: não utilizado

card: Indica a Placa que gerou o evento

5.3.15 OnE1Alarm (EV_E1_ALARM)

Indica a ocorrência de um alarme nos framers E1

Descrição:

A monitoração dos alarmes é essencial para o funcionamento das aplicações que utilizam a placa VoicerBox E1 30/60.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

E1ID - Indica qual dos 2 framers E1 gerou o alarme

AlarmCode - Mostra o alarme gerado:

- ALARM_RSLIP (0x01) - Escorregamento (problema de sincronismo)
- ALARM_RAIS (0x02) - Alarme remoto
- ALARM_AISS (0x04) - Indicação de alarme
- ALARM_AIS16S (0x08) - Indicação de alarme canal 16
- ALARM_LOSS (0x10) - Perda de sinal
- ALARM_RESERVED (0x20) - Reservado
- ALARM_MFSYNC (0x40) - Sincronismo de multiquadro
- ALARM_SYNC (0x80) - sincronismo de quadro

AlarmData - Se for ALARM_RSLIP, indica o contador de vezes que detectou o *escorregamento*. Nos outros alarmes, pode receber ON (1) ou OFF (0).

Context_Data (API):

command - EV_E1_ALARM (0x38)

port - Indica a Placa que gerou o evento

data: código do alarme (ver **AlarmCode** acima)

data_aux: estado do alarme (ON/OFF) ou contador no caso do SLIP

card: Indica a Placa que gerou o evento

5.3.16 OnE1GroupB (EV_GROUP_B)

Ocorre quando a *thread E1* recebe um grupo B

Descrição:

Este evento ocorrerá sempre quando a thread E1 estiver em uso.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Value: Indica o tipo de dado

- B_FREE_CALLING (1)
- B_BUSY (2)
- B_NUMBER_CHANGED (3)
- B_CONGESTION (4)
- B_FREE_WITHOUTBILLING (5)
- B_COLLECTCALL (6)
- B_NUMBER_UNKNOWN (7)
- B_OUT_OF_SERVICE (8)

Context_Data (API):

command - EV_GROUP_B (0x34)

port/card - Indica a porta que gerou o evento

data: mesmos valores do parâmetro **Value** explicado acima

data_aux: não utilizado

card: Indica a Placa que gerou o evento

5.3.17 OnE1FramerResponse (EV_FRAMER)

Ocorre sempre que for solicitado um comando para o Framer E1

Descrição:

Este evento permite analisar a situação dos framers E1 e deverá somente ser utilizado quando houver algum problema de comunicação com uma outra central, por exemplo.

Parâmetros Recebidos:

Card – Indica a placa que gerou o evento

Data1 - Dado indicativo do primeiro framer

Data2 - Dado indicativo do segundo framer

Context_Data (API):

command - EV_FRAMER (0x3b)

data: Dado indicativo do primeiro framer

data_aux: Dado indicativo do segundo framer

card: não utilizado

5.3.18 OnE1StateChange (EV_E1CHANGESTATUS)

Ocorre sempre quando algum estado da sinalização R2D é alterado.

Descrição:

Durante a troca de sinalização R2D, ocorrem diversos eventos indicando o ponto onde o protocolo se situa. Basicamente este evento deve ser utilizado para monitoração, não sendo necessário nenhum tratamento específico já que a *Thread E1* cuida de todos os detalhes para o desenvolvedor.

Caso o programador esteja tratando toda a sinalização sem utilizar a *thread E1* (*não recomendado*), este evento deverá ser utilizado para dar as informações necessárias para a troca de sinalização.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

State - Indica o estado atual e pode receber os seguintes valores:

- **C_NOTCOMPLETED** (0x1015) - Ligação não
-

completada

- **C_B_ENDCALL** (0x1014) - Assinante B desligou
- **C_ANSWERED** (0x1012) - Assinante B atendeu ou retornou depois de **C_B_ENDCALL**
- **C_CONGESTION** (0x1013) - Congestionamento
- **C_SEIZURE** (0x100b) - Ocupação
- **C_GROUP_II** (0x1017) - Grupo II recebido
- **C_NUMBER_RECEIVED** (0x100d) - Número recebido durante ligação entrante
- **C_UNAVAILABLE** (0x1011) - Canal não disponível
- **C_GROUP_I** (0x101c) - Recebeu grupo I
- **C_ID_RECEIVED** (0x100a) - Recebeu identificação de assinante A

Context_Data (API):

command - EV_E1CHANGESTATUS (0x33)

port - Indica o canal da Placa que gerou o evento

data: mesmos valores do **State** (ActiveX) acima

data_aux: não utilizado

card: não utilizado

5.3.19 OnFaxDetected (EV_FAX)

Ocorre quando um sinal de fax é detectado pela placa.

Descrição:

O programador tem a flexibilidade de a qualquer ponto da aplicação habilitar e desabilitar tais recursos, fazendo uso dos métodos EnableCallProgress() para habilitar e DisableCallProgress() para desabilitar.

Sempre quando for detectado um sinal de fax na linha, o evento OnFaxDetected será chamado, permitindo tratamentos especiais, como por exemplo, desviar para o ramal do aparelho de fax.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Context_Data (API):

command - EV_FAX (0x22)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.20 OnLineOff (EV_LINEOFF)

Ocorre quando o usuário desliga o aparelho telefônico conectado à placa em paralelo.

Descrição:

O Evento **OnLineOff** ocorre sempre que o aparelho telefônico for desligado.

Nas placas digitais, este evento também é gerado de acordo com a sinalização R2D indicando fim de ligação.

Exclusivamente para a placa FXO, nas versões anteriores da VoicerLib este evento era gerado apenas quando havia uma conexão em paralelo com a placa. Agora este evento também ocorre quando a placa executa um comando **HangUp** indicando

que está "no gancho".

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

Context_Data (API):

command - EV_LINEOFF (0x27)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.21 OnLineReady (EV_LINEReadY)

Ocorre quando um aparelho telefônico conectado à placa em paralelo toma a linha para originar uma ligação ou após executar o comando PickUp.

Descrição:

Nas placas FXO, quando a conexão é feita de acordo com o explicado no tópico **Gravação em Paralelo** deste manual, o evento OnLineReady também é gerado quando se detecta *Loop* na linha.

Nas placas digitais, este evento também é gerado de acordo com a sinalização R2D indicando início de ligação.

Exclusivamente para a placa FXO, nas versões anteriores da VoicerLib este evento era gerado apenas quando havia uma

conexão em paralelo com a placa. Agora este evento também ocorre quando a placa executa um comando Pickup indicando que está "fora do gancho".

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Context_Data (API):

command - EV_LINEREADY (0x25)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.22 OnLoggerEvent (EV_LOGGEREVENT)

Ocorre em diversas situações quando se utiliza a *thread de Logger*

Descrição:

Este evento só ocorrerá quando se estiver utilizando a *thread de controle de logger*, criada através da função CreateLoggerControl. O parâmetro LoggerStatus indicará diversas situações para que o aplicativo possa controlar as várias etapas da sinalização R2, o início e o fim das ligações, disparando ou finalizando as gravações.

Para maiores informações, leia o capítulo **Gravação em Paralelo - Placas E1 30/60**.

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

LoggerStatus - Pode assumir os seguintes valores:

- **LOGGER_FREE_WITH_BILLING (1)** - Assinante livre com tarificação
- **LOGGER_BUSY (2)** - Ocupado
- **LOGGER_NUMBER_CHANGED (3)** - Número de destino mudou
- **LOGGER_CONGESTION (4)** - Congestionamento
- **LOGGER_FREE_WITHOUT_BILLING (5)** - Livre sem tarificação
- **LOGGER_FREE_RETENTION (6)** - Livre com retenção
- **LOGGER_LEVEL_NUMBER_AVAILABLE (7)** - Número disponível
- **LOGGER_B_ENDCALL (20)** - Assinante B desligou durante conversação
- **LOGGER_B_RETURN (21)** - Assinante B retornou à ligação.
- **LOGGER_LINEREADY (22)** - Início da conversação
- **LOGGER_LINEOFF (23)** - Fim da conversação

Context_Data (API):

command - EV_LOGGEREVENT (0x37)

port - Indica o canal da Placa que gerou o evento

data: mesmos valores do parâmetro LoggerStatus mostrado acima

data_aux: não utilizado

card: não utilizado

5.3.23 OnMenu

Ocorre ao término de uma função de menu iniciada pelo MenuStart

Descrição:

O evento OnMenu é gerado após o término da função especial de menu

Parâmetros Recebidos (ActiveX):

Port – Indica o canal da Placa que gerou o evento

OptionSelected – Opção digitada pelo usuário

Status – Indica o que aconteceu:

- **msAborted** – Cancelado pelo método MenuAbort
- **msTimeOut** – Usuário não digitou nada
- **msRetriesExceeded** – Número de tentativas excedido
- **msValidOptionDetected** – Foi digitada uma opção válida

5.3.24 OnPlayStart (EV_PLAYSTART)

Ocorre sempre quando for iniciado o playback de qualquer mensagem.

Descrição:

Sempre quando se desejar que a placa fale uma mensagem, deve-se fazer uso do método PlayFile, e toda vez que este método é chamado, o evento OnPlayStart ocorrerá.

Este evento é indicado para atualizações na interface, como por exemplo, desabilitar botões, mostrar mensagens, etc...

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Context_Data (API):

command - EV_PLAYSTART (0x2a)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.25 OnPlayStop (EV_PLAYSTOP)

Ocorre sempre quando for finalizado o playback de qualquer mensagem.

Descrição:

O evento OnPlayStop ocorrerá quando o playback for interrompido. Este evento retorna na variável StopStatus o motivo da interrupção.

Parâmetros Recebidos:

- **Port** – Indica o canal da Placa que gerou o evento
- **StopStatus** – Indica o motivo da interrupção. Os códigos são:
 - **ssNormal** – Significa a mensagem foi falada por

completo (terminou normalmente);

- **ssDigitReceived** – Significa que a mensagem foi interrompida pelo recebimento de um dos dígitos passados como parâmetro para interrupção da mensagem. O dígito deve ser recuperado na propriedade Digits.
- **ssStopped** – Significa que a mensagem foi interrompida pela chamada do método StopPlayFile().
- **ssAbnormal** – Significa que a mensagem foi interrompida por causa de algum erro indeterminado.

Context_Data (API):

command - EV_PLAYSTOP (0x29)

port - Indica o canal da Placa que gerou o evento

data: mesmos valores do parâmetro **StopStatus** explicado acima

data_aux: não utilizado

card: não utilizado

5.3.26 OnPrompt

Ocorre quando a função de prompt chega ao fim.

Descrição:

O Evento ocorre ao final do prompt, que foi iniciado pelo método PromptStart.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Value – Valor digitado pelo usuário

Status – Pode assumir os valores:

- **mpAborted** – Cancelado pelo método PromptAbort
-

- **mpRetriesExceeded** – Número de tentativa excedido
- **mpOk** – Usuário confirmou entrada de dados
- **mpCanceled** – Usuário cancelou

5.3.27 OnR2Received (EV_R2)

Ocorre sempre quando a sinalização R2 é recebida pela placa.

Descrição:

Este evento ocorrerá sempre quando alguma sinalização R2 é recebida pela placa. Antes de receber qualquer sinalização é necessário habilitar a recepção de R2 através da função `SendR2Command` (`dg_SendR2Command`) passando como parâmetro **R2_ENABLE**.

Quando a *thread E1* estiver sendo utilizada, o controle de receber ou não estas informações fica sob responsabilidade da biblioteca.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Signal - Indica o sinal R2 recebido:

- **R2_IDLE** (0x9)
- **R2_CLEAR_FOWARD** (0x9)
- **R2_SEIZURE** (0x1)
- **R2_BACKWARD_DISCONNECTION** (0x1)
- **R2_SEIZURE_ACK** (0xd)
- **R2_BILLING** (0xd)
- **R2_CLEAR_BACK** (0xd)
- **R2_ANSWERED** (0x5)
- **R2_BLOCKED**(0xd)
- **R2_FAILURE** (0xd)
- **R2_ENABLE** (0x10)

- **R2_DISABLE** (0x20)

Context_Data (API):

command - EV_R2 (0x35)

port - Indica o canal da Placa que gerou o evento

data: mesmos valores do parâmetro **Signal** explicado acima

data_aux: não utilizado

card: não utilizado

5.3.28 OnRecording (EV_RECORDING)

Ocorre durante a gravação de uma mensagem.

Descrição:

O evento OnRecording é ideal para monitorar o andamento de uma gravação. A variável *Duration* contém a duração em segundos da gravação até aquele momento.

Através deste evento é possível limitar o tamanho das mensagens ou mesmo exibir informações sobre o andamento da gravação.

Parâmetros Recebidos:

- **Port** – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.
- **Duration** – Indica a duração da gravação em segundos naquele instante.

Context_Data (API):

command - EV_RECORDING (0x30)

port - Indica o canal da Placa que gerou o evento

data: Indica a duração da gravação em segundos naquele instante.

data_aux: não utilizado

card: não utilizado

5.3.29 OnRecordStart (EV_RECORDSTART)

Ocorre sempre quando for iniciado a gravação de qualquer mensagem.

Descrição:

Sempre quando se desejar gravar uma conversa, deve-se fazer uso do método RecordFile, e toda vez que este método é chamado, o evento OnRecordStart ocorrerá.

Este evento é indicado para atualizações na interface, como por exemplo, desabilitar botões, mostrar mensagens, etc...

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Context_Data (API):

command - EV_RECORDSTART (0x2e)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.30 OnRecordStop (EV_RECORDSTOP)

Ocorre sempre quando for finalizado a gravação de qualquer mensagem.

Descrição:

O evento OnRecordStop ocorrerá quando a gravação for interrompida. Este evento retorna na variável Status o motivo da interrupção.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

StopStatus – Indica o motivo da interrupção. Os códigos são:

- **ssDigitReceived** – Significa que a mensagem foi interrompida pelo recebimento de um dos dígitos passados como parâmetro para interrupção da mensagem. O dígito deve ser recuperado na propriedade Digits.
- **ssStopped** – Significa que a mensagem foi interrompida pela chamada do método StopPlayFile().
- **ssAbnormal** – Significa que a mensagem foi interrompida por causa de algum erro indeterminado.

Context_Data (API):

command - EV_RECORDSTOP (0x2f)

port - Indica o canal da Placa que gerou o evento

data: mesmos valores do parâmetro **StopStatus** explicado acima

data_aux: não utilizado

card: não utilizado

5.3.31 OnRingDetected (EV_RINGS)

Ocorre sempre quando for detectado o sinal de Ring na linha.

Descrição:

O evento Ring ocorre sempre quando o sinal de ring for detectado pela placa, portanto, se o usuário demorar 3 toques para atender o telefone, o evento OnRingDetected ocorrerá 3 vezes.

Isto é válido também para a placa digital, pois ao finalizar a troca de sinalização R2D MFC, a VoicerLib também gera o evento na cadência 1x4 (1 Ring para 4s de intervalo).

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Context_Data (API):

command - EV_RINGS (0x1b)

port - Indica o canal da Placa que gerou o evento

data: não utilizado

data_aux: não utilizado

card: não utilizado

5.3.32 OnSilenceDetected (EV_SILENCE)

Ocorre sempre quando for detectado o silêncio ou áudio em determinada porta

Descrição:

Para habilitar a detecção de silêncio utiliza-se o `EnableSilenceDetection` passando os valores mínimos de silêncio ou áudio. Cada vez que o estado se altera entre silêncio e áudio, o evento **OnSilenceDetected** é gerado, passando no parâmetro **State** o que foi detectado.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento
State - Indica se foi detectado um silêncio
(`DG_SILENCE_DETECTED`) ou áudio
(`DG_AUDIO_DETECTED`)

Context_Data (API):

command - `EV_SILENCE` (0x3f)
port - Indica o canal da Placa que gerou o evento
data: Indica se foi detectado um silêncio
(`DG_SILENCE_DETECTED`) ou áudio
(`DG_AUDIO_DETECTED`)
data_aux: não utilizado
card: não utilizado

5.4 Propriedades exclusivas do ActiveX

Neste capítulo, são mostradas as propriedades que são de uso exclusivo do ActiveX. A versão de API tem funções que desempenham as mesmas funções.

5.4.1 ConfigPath

Tipo: String

Função: Determina o arquivo e o caminho de configuração para as placas VoicerBox E1 30/60

Descrição: Esta propriedade permite ao programador customizar o nome e o caminho do arquivo de configuração para as placas VoicerBox E1 30/60. Se nenhum valor for informado será utilizado **c:\arquivos de programas\voicerlib3**.

NA API o caminho de configuração é passado diretamente pela função StartVoicerLib

5.4.2 StockSigPath

Tipo: String

Função: Determina o caminho onde serão encontradas os arquivos de áudio (wave) utilizados para reproduzir valores por extenso, data, etc...

Descrição: Ao distribuir uma aplicação onde são utilizadas as funções PlayCurrency, PlayDate, PlayTime, PlayNumber ou PlayList, o desenvolvedor necessita também enviar as mensagens a serem utilizadas. Esta propriedade permite configurar o caminho para estas mensagens.

A seguir apresentamos os nomes dos arquivos fornecidos com a biblioteca e a respectiva frase que reproduz.

<u>Arquivo SIG</u>	<u>Frase</u>	<u>Arquivo SIG</u>	<u>Frase</u>
Hora.wav	Hora	Horas.wav	Horas
Minuto.wav	Minuto	Minutos.wav	Minutos
Segundo.wav	Segundos	Real.wav	Real
Reais.wav	Reais	DeReais.wav	De Reais
Centavo.wav	Centavo	Centavos.wav	Centavos
Cento.wav	Cento	100.wav	Cem
200.wav	Duzentos	300.wav	Trezentos
400.wav	Quatrocentos	500.wav	Quinhentos
600.wav	Seiscentos	700.wav	Setecentos
800.wav	Oitocentos	900.wav	Novecentos
Mil.wav	Mil	milhão.wav	Milhão
milhoes.wav	Milhões	bilhão.wav	Bilhão
bilhoes.wav	Bilhões	trilhão.wav	Trilhão
trilhoes.wav	Trilhões	Virgula.wav	Virgula
Ponto.wav	Ponto	De.wav	De

<u>Arquivo SIG</u>	<u>Frase</u>	<u>Arquivo SIG</u>	<u>Frase</u>
e.wav	E	Barra.wav	Barra
traco.wav	Traço	0.wav	Zero
1.wav	Um	2.wav	Dois
3.wav	Três	4.wav	Quatro
5.wav	Cinco	6.wav	Seis
7.wav	Sete	8.wav	Oito
9.wav	Nove	10.wav	Dez
11.wav	Onze	12.wav	Doze
13.wav	Treze	14.wav	Quatorze
15.wav	Quinze	16.wav	Dezesseis
17.wav	Dezessete	18.wav	Dezoito
19.wav	Dezenove	20.wav	Vinte

<u>Arquivo SIG</u>	<u>Frase</u>	<u>Arquivo SIG</u>	<u>Frase</u>
30.wav	Trinta	40.wav	Quarenta
50.wav	Cinquenta	60.wav	Sessenta
70.wav	Setenta	80.wav	Oitenta
90.wav	Noventa	Janeiro.wav	Janeiro
Fevereiro.wav	Fevereiro	Marco.wav	Março
Abril.wav	Abril	Maiο.wav	Maiο
Junho.wav	Junho	Julho.wav	Julho
Agosto.wav	Agosto	Setembro.wav	Setembro
Outubro.wav	Outubro	Novembro.wav	Novembro
Dezembro.wav	Dezembro		

Caso o desenvolvedor queira fazer sua própria locução ou mesmo em outra língua basta criar as frases acima, respeitando-se sempre o nome do arquivo e o formato utilizado **(wave lei mi, 8Khz, 8 bits, mono)**.

Parte



VI

Utilizando o Configurador da placa E1

6 Utilizando o Configurador da placa E1

Ao instalar a VoicerLib ou Kit Integrador, também é instalado o Configurador da Placa E1, também chamado de **VBoxConfig**. Este utilitário é muito importante pois permite configurar, monitorar eventos e testar as placas instaladas no computador.

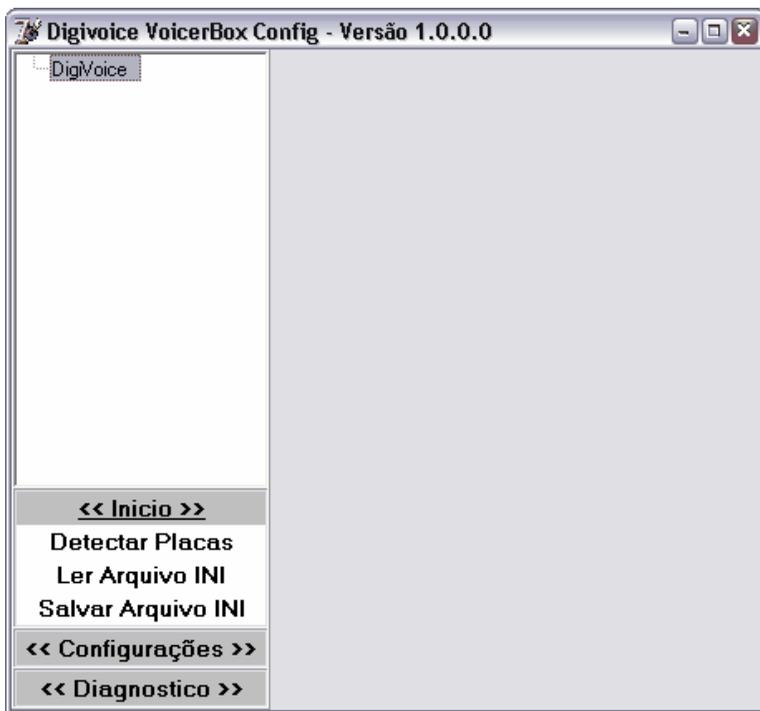
Além disso, também permite que toda a configuração seja salva e posteriormente utilizada diretamente pela aplicação, através da chamada do método **ReadConfigurationFile**.

A tela principal do VBoxConfig é apresentada abaixo:

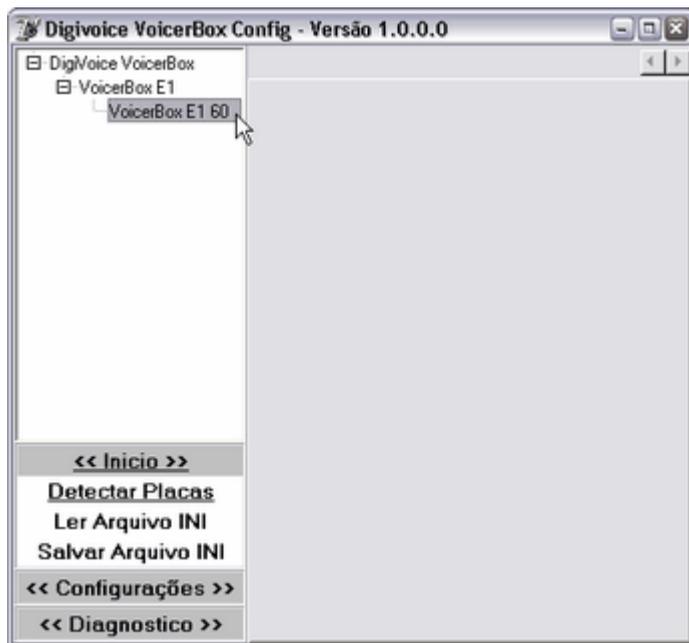


6.1 Iniciando o Programa

Para detectar as Placas E1 é necessário entrar no Menu << **Início** >> e depois clicar em **Detectar Placas**. Feito isso o sistema automaticamente irá iniciar a VoicerLib e junto todas as placas E1 que estiverem conectadas no PC.



Após a detecção, serão listadas todas as placas em uma lista acima dos menus.



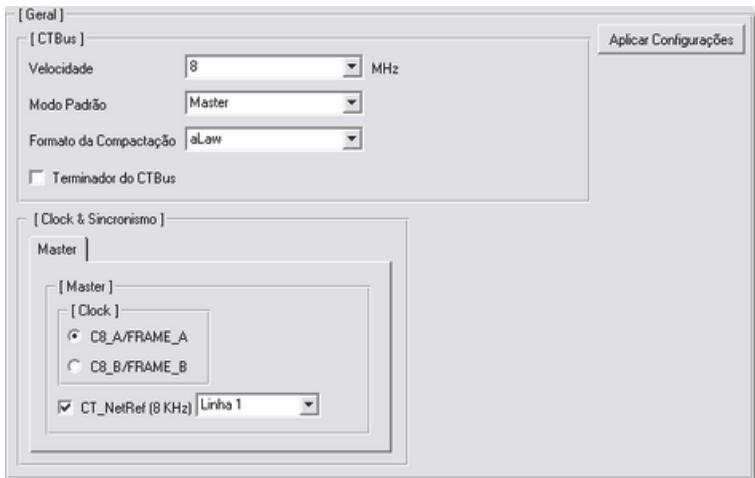
Caso haja algum erro, uma mensagem será exibida na tela e as placas não serão listadas na parte superior do menu.

6.2 Configurando as Placas

As configurações apresentadas aqui são um *"mapa rápido"* para ter as placas configuradas em um ambiente de teste/desenvolvimento. Para maiores detalhes sobre cada opção, consulte o Help do **VBoxConfig**.

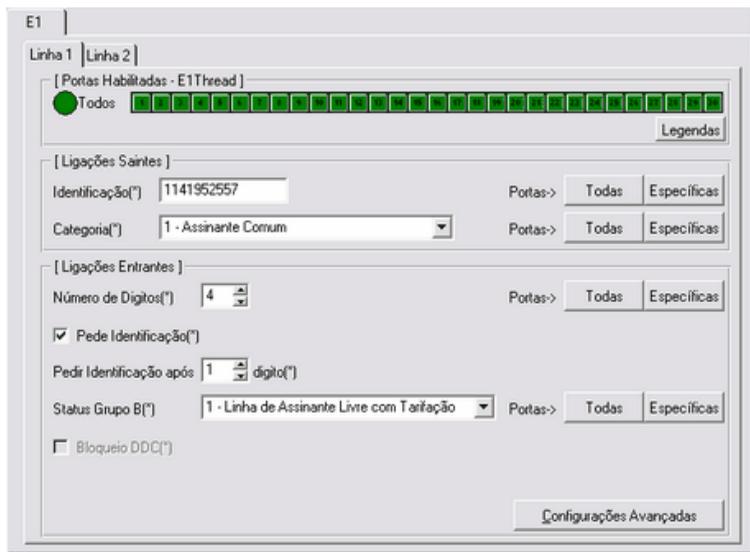
Para configurar uma placa VoicerBox E1, primeiramente deve-se seleccionar uma placa na lista de placas detectadas. Após a

seleção entre no Menu << **Configurações** >>. Ao fazer isso aparecerão as opções: **Hardware**, **Geral** e **Protocolo**. Clique em **Geral** e será possível visualizar as seguintes opções:



Nesta janela configure o campo **Modo Padrão** como **Master**. Para testar a placa em *loop* (ler mais adiante) defina o campo **CT_NetRef** como **Interno**. Caso a placa seja conectada ao DigiVoice InterPro E1 ou a outro equipamento, preencha o campo **CT_NetRef** como **Linha 1**. Após configurar clique no botão **Aplicar Configurações**.

Após alterar as configurações gerais iremos para as configurações de protocolo. Para acessá-las entre no menu **Configurações** e selecione **Protocolo** o que abrirá uma janela com varias opções.



Abaixo a descrição de como configurar os principais campos:

- **Portas Habilitadas** – E1Thread (*Configuração individual por Linha E1*):
 - **Todas**: Para habilitar todas as portas individualmente por Linha E1 clique no título "Todos" ou clique no "Circulo" ao lado;
 - **Individual**: Para habilitar as portas individualmente clique nos painéis existentes identificados com o número da porta;
- **Ligações Saindes** (*Configurações individuais por Linha E1*):
 - **Identificação** (BINA, independente por porta):
 - **Todas**: Após preencher o campo de "Identificação" clique no botão "Todas" ao lado do campo de "Identificação";
 - **Individual**: Após preencher o campo de "Identificação" clique no botão "Específicas", após

clica-lo será aberta uma janela onde você poderá preencher a "Identificação" individual por Porta;

- **Categoria** (Grupo II, independente por porta):
 - **Todas:** Após preencher o campo de "Categoria" clique no botão "Todas" ao lado do campo de "Categoria";
 - **Individual:** Após preencher o campo de "Categoria" clique no botão "Específicas", após clica-lo será aberta uma janela onde você poderá preencher a "Categoria" individual por Porta;
-
- **Ligações Entrantes** (*Configurações individuais por Linha*):
 - **Número de dígitos** (independente por porta):
 - **Todas:** Após preencher o campo de "Número de dígitos" clique no botão "Todas" ao lado do campo de "Número de dígitos";
 - **Individual:** Após preencher o campo de "Número de dígitos" clique no botão "Específicas", após clica-lo será aberta uma janela onde você poderá preencher a "Número de dígitos" individual por Porta;

 - **Pede Identificação** (BINA, geral por Linha E1);
 - **Pedir Identificação após dígito** (geral por Linha E1);
 - **Status Grupo B** (Grupo B, independente por porta):
 - **Todas:** Após preencher o campo de "Status Grupo B" clique no botão "Todas" ao lado do campo de "Status Grupo B";
 - **Individual:** Após preencher o campo de "Status Grupo B" clique no botão "Específicas", após clica-lo será aberta uma janela onde você poderá preencher a "Status Grupo B" individual por Porta;
-



Ao lado de cada nome de campo a ser configurado temos um identificador que mostra em que Status se encontra o campo, são eles:

1. (?): Configuração ainda não foi aplicada para todas as portas ou para portas específicas;
2. (*): Configuração válida para todas as portas;
3. (1): Configuração individual por porta;

6.3 Salvando as configurações

O **VBoxConfig** permite que todas as configurações selecionadas sejam salvas em um arquivo para posterior utilização. Desta forma, não é necessário repetir as mesmas configurações todas as vezes que se executar o utilitário. Também é possível ler estas configurações diretamente na aplicação através do método **ReadConfigurationFile**.

Para salvar todas as configurações selecionadas, deve-se clicar na opção **Salvar Arquivo INI**.



Aparecerá as seguintes opções:



Se for escolhida a opção **Local Padrão**, um arquivo chamado DV_VLIB3.ini na pasta C:\Arquivos de programas\voicerlib3. Já a opção **Local Selecionado** dá ao usuário de escolher o nome do arquivo e sua localização. Com isso é possível manter vários arquivos de configuração para placas e aplicações distintas.



Salvando no Local Padrão fará com que toda vez que o VBoxConfig for iniciado, será dada a opção de se ler este arquivo automaticamente.

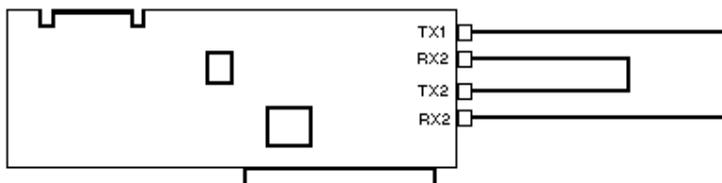
6.4 Testando a Placa em Loop

O teste local ou teste em loop permite testar a placa VoicerBox E1 30/60 sem que haja a necessidade de outro equipamento. Os testes ficam limitados devido ao fato de não se poder interagir com a placa através de, por exemplo, um aparelho telefônico. Entretanto este teste permite verificar diversas funcionalidades de troca de sinalização, discagem, atendimento

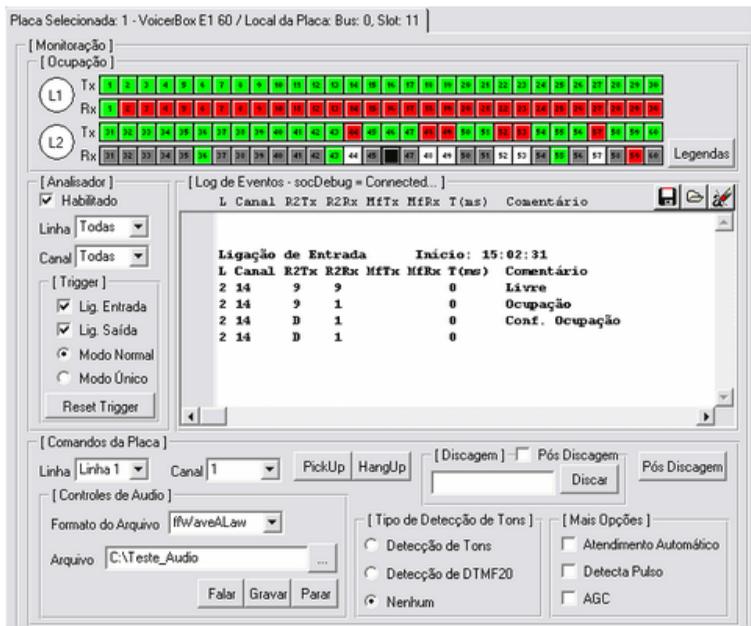
etc...

Para efetuar o teste local é necessário que a placa tenha 60 canais pois ao fechar o loop local o canal **1** estará conectado diretamente ao **31**, o **2** ao **32** e assim por diante.

Para montar o cabo para teste da placa em Loop, você deverá pegar 2 cabos coaxiais e ligar o cabo 1 do TX1 no RX2 e o cabo 2 do RX1 no TX2. Observe o diagrama abaixo:



Para testar a sua placa, clique no menu << **Diagnostico** >> e depois em **Monitoração**. Ao abrir a janela de monitoração serão mostrados todos os canais referentes a placa selecionada:



Para efetuar uma ligação de teste entre o canal 1 e o 31, por exemplo, selecione a **Linha 1** e o **Canal 1** (na área da tela chamada **Comandos da Placa**), clique no botão **PickUp** para atender o canal, coloque na caixa **Discagem** o número a ser discado (Ex.: 1234). Por fim clique no botão **Discar**.

Durante a discagem, os canais **1** da **Linha1** (*porta 1*) e **1** da **Linha 2** (*porta 31*) irão trocar as sinalizações que poderão ser observadas no **Log de Eventos** na mesma tela. Após o processo de troca de sinalização o **Canal 1** da **Linha 2** (*porta 31*) já estará pronto para ser atendido

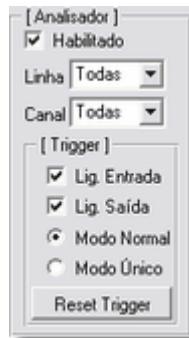
Para atender o canal, selecione a **Linha 2** e o **Canal 1** e clique no botão **PickUp**. Poderá ser notado que o Painéis **Rx1** e **Tx31** estarão na cor Azul, significando que estão atendidos. Para desligar clique no botão **HangUp**.



As ligações de teste poderão ser feitas entre quaisquer canais, sempre tendo em mente que existe uma correspondência física entre 1-31, 2-32, etc...

6.5 Verificando o Log

Ao iniciar a aplicação de teste como padrão o Trigger de Log vem pré-configurado para a Linha 1 e Canal 1, isso significa que a aplicação irá monitorar somente o a Linha 1 e o Canal 1, para poder monitorar outro canal você deverá selecionar outra Linha/Canal e depois clicar no botão "Reset Trigger".



Caso hajam vários canais conectados, recomenda-se configurar o trigger para monitorar uma porta específica para evitar excesso de informações na tela de *log*.



[Log de Eventos - socDebug = Connected...]

L Canal	R2Tx	R2Rx	MfTx	MfRx	T(ms)	Comentário
Ligação de Entrada		Início: 15:02:31				
2 14	9	9			0	Livre
2 14	9	1			0	Ocupação
2 14	D	1			0	Conf. Ocupação
2 14	D	1			0	
2 14	D	1			0	
2 14	9	1			0	Livre