Digi
Voice

Digivoice Tecnologia em Eletrônica

Digivoice VoicerLib2

Sistema de Desenvolvimento para placas Digivoice

Guia do Programador

MAN0024 - Versão 4.2
Impressão: 28/07/2006, 16:26

Conteúdo

Parte I

Introdução

1 Introdução

A Digivoice Tecnologia em Eletrônica disponibilizou para todos os desenvolvedores interessados em trabalhar com sua plataforma de hardware uma biblioteca de desenvolvimento.

Esta biblioteca é distribuída na forma de componentes ActiveX para plataforma Windows 32 bits (a partir do Windows 98SE). Recomenda-se fortemente a utilização do Windows XP.

A característica aberta do ActiveX permitirá ao desenvolvedor grande versatilidade, já que será possível utilizar os componentes em qualquer plataforma de desenvolvimento que tenha suporte a ActiveX. Eis alguns exemplos:

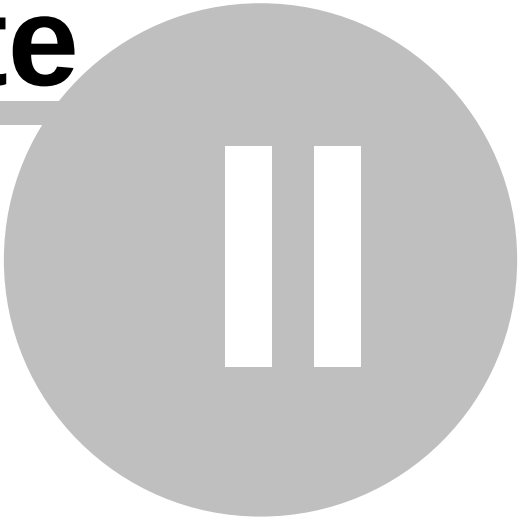
- Borland Delphi (a partir da versão 5.0)
- Borland C++ Builder (a partir da versão 3.0)
- Microsoft Visual Basic (a partir da versão 6.0)
- Microsoft Visual C++ (a partir da versão 5.0)

Nesta versão é possível trabalhar tanto com a placa VoicerPhone como com a VoicerBox e também é possível trabalhar com mais de uma placa por micro.

Este manual visa dar ao desenvolvedor uma visão completa das funções da biblioteca, mostrando o modo de programação e todas as funções disponíveis.

Além disso, conta com um guia de referência de todas as propriedades, eventos e métodos existentes.

Parte



Histórico de Versões

2 Histórico de Versões

2.88 28/07/2006

- Em casos de erro de escrita de gravação a VoicerLib gerará o evento OnErrorDetected com código 6 ao inves de interromper a gravação.
- Modificado funcionamento do timer interno

2.87 02/06/2006

- Melhoria na rotina de fechamento de arquivos na gravação
- Aumento do timeout de segurança do makecall para 60 segundos para prever discagens via celular ou sistemas voip

2.86 19/05/2006

- A função PlayListClear agora verifica se um PlayList estiver em execução antes de apagar a lista
- Corrigido problema na função IdleStart. O evento afterpickup era gerado indefinidamente caso fosse executado algum menu, prompt ou getdigits durante o curso da ligação.

v2.85 06/05/2006

- Atualização dos drivers para facilitar coexistencia com a nova versão 4.0 da VoicerLib para a nova família de placas

v2.84 23/03/2006

- Prompt nao ativa mais a detecção de pulso, evitando pegar digitos inválidos devido à ruidos na linha. Caso seja necessário, a detecção de pulso deverá ser iniciada manualmente

V2.83 17/02/2006

- Mudança na rotina de interrupção, sendo implementado o mesmo sistema de identificacao de placas da VoicerLib 4. Com isso foi resolvido o problema de reprodução de frases, que em algumas situações "pulava" partes da frase
- SetRecordGain agora passa o ganho multiplicado por 2 para o driver, fazendo com que a aplicação do ganho seja mais sensível, ajudando em casos com áudio da linha telefônica muito baixo.
- O firmware da placa PCI/1 foi modificado. O ganho de áudio de entrada não é mais dado por hardware e sim por software no proprio firmware.

V2.82 10/02/2006

- No PlayFile havia saida de função sem desligar o criticalsection, podendo ocasionar travamentos na voicerlib. Um efeito disso era parar de falar. O GSM também verifica a estrutura do cabeçalho. Agora o playfile analisa a extensão, mas se o cabeçalho não conferir, será utilizado o que o usuário definiu no formato de reprodução através da função SetPlayFormat

V2.81 31/01/2006

- Criado um timeout na função MakeCall, desligando a linha no caso de ficar sem resposta nenhuma por mais de 10 segundos
- Modificada o método interno para detecção de atendimento, deixando-o mais eficiente.

v2.80 - 16/01/2006

- Gerado novo firmware da placa VoicerPhone PCI/1 para

evitar que desconexão por ruido de rele/fone/mic.
- Na finalização da VoicerLib, a placa PCI/1 não era resetada, podendo causar travamentos da máquina, principalmente em ambiente de desenvolvimento.

v2.79 - 08/11/2005

- Corrigido problema de finalização quando utilizada em aplicações do tipo **Serviço**
- Corrigido problema na função GSMToWave, que escrevia o tamanho da area e dados errado no arquivo Wave
- Criado controle que evita a chegada de dois LineReady em sequencia quando a ligação já estiver em curso nas placas VoicerPhone PCI/1

Versão 2.78 - 15/09/2005

- Recompilado com bibliotecas de base atualizadas para computadores com tecnologia HT

Versão 2.77- 21/07/2005

- Quando o MakeCall retorna o estado *mkDialToneAfterDial* a VoicerLib nao dá mais HangUp automatico.
- Corrigido o problema o MenuStart/OnMenu. Na versão 2.75/76 foi introduzido um erro que fez com que se fosse passado um nome de arquivo específico no MenuStart ao invés de lista "@", o evento OnMenu não era gerado.
- Gerenciamento de eventos da placa foi modificado para evitar o uso de 100% de CPU livre.

Versão 2.76- 07/07/2005

- Corrigida a grafia do método RemoveConferencePort
- Corrigido o timeout da espera de digitos após o atendimento para as funções Idlexxx.

- Setup compatibilizado para máquinas Pentium 4 HT

Versão 2.75- 20/04/2005

- Adicionada consistencia na chamada do MakeCall no caso de as funções de Idle estiverem ativadas. Ao chamar o MakeCall, o Idle do canal é temporariamente desativado, sendo reativado ao chamar o evento OnAfterMakeCall
- Ao executar um ReadDigits após uma gravação que utilizava digitos terminadores, todos os eventuais digitos eram recuperados. Agora, o ReadDigits só retornará o digito que interrompeu a gravação.
- Problema na utilização do SampleToApp e SampleToCard em várias portas corrigido.
- Novo exemplo de Streaming em Delphi
- Corrigido problema de chamar o MenuStart de dentro do OnPlayStop. Ao fazer isso, o menu já entrava com uma deteção de digitos em curso. Foi criado novo estado inicial de menu para corrigir o problema.
- Ao ser gerado o evento OnMenu, a VoicerLib não desliga mais as detecções (CallProgress) automaticamente.
- No MakeCall foi adicionada a possibilidade de se configurar ZERO flashes de inicio, para os casos de retomadas que só necessitam de digitos.
- Aumentado timeout de tom de linha no curso de MakeCall
- Bloquado acesso à placas ISA (Este tipo de placa só é suportado até a versão 2.71)
- Corrigido problema de nao aparecer as propriedades DriverEnabled, DriverVersion e FirwareVersion

Versão 2.74- 28/02/2005

- Nas funções PlayXXX agora a parte fracionária é reconhecida pela vírgula ou pelo ponto.
- Função de extenso permite falar parte fracionária com 3 dígitos

Versão 2.73- 11/01/2005

- Corrigido problema da função CancelGetDigits introduzida na versão 2.72 não aparecer nos ambientes de desenvolvimento (Delphi, VB, etc...).
- Criados novos métodos EnableEchoCancel e DisableEcoCancel para habilitar ou desabilitar o cancelamento de eco nos recursos de conferência.
- Criado novo algoritmo de cancelamento de eco no device driver, permitindo conferências com melhor qualidade de audio.

Versão 2.72- 20/12/2004

- Novo método CancelGetDigits para cancelar qualquer GetDigits em curso em um determinado canal. Não gera evento nenhum e não mantém o buffer de dígitos.
- Corrigido problema do Idle de gerar dois eventos AfterPickup .
- Alterado padrão de timeout global do GetDigits. Agora o TimeOut é referente ao tempo de chegada do primeiro dígito. Apos o primeiro dígito, o timer é cancelado.
- Corrigido problema de se na passar número de flashes no digito pra retomada em caso de ocupado (MakeCall).
- Problema corrigido no extenso de data com formato m/d/y.
- Incluído o evento de "Esperando tom de discagem" durante o MakeCall.
- Corrigido o PromptStart que apagava o buffer de dígitos caso fosse passado um arquivo wave e não uma lista de arquivos (@).
- Novo diretório StockSigs regravado.

Versão 2.71- 18/06/2004

- O Idle agora utiliza o timeout interdigito para esperar o primeiro digito também. Se não chegar no tempo

especificado, termina e gera o AfterPickUp.
- Após o timeout interdigito, o AfterPickup é gerado sempre apos 1,5 segundos.
- Se estourar o timeout global, despreza todos os digitos recebidos.
- Corrigido problema de gravação e reprodução simultanea em GSM, que poderia causar travamentos.

Versão 2.70- 04/06/2004

- Corrigido problema de versão que impedia o uso da voicerlib no VC++ e no PowerBuilder.
- Prompt - Corrigido problema de não interromper a reprodução de mensagens quando fosse esperado apenas 1 digito.
- O PlayFile agora 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.
- Corrigido problema no MakeCall. Em algumas situações o MakeCall jamais gerava o AfterMakeCall, fazendo com que a aplicação ficasse eternamente no estado de Calling.

Versão 2.69- 03/02/2004

- Retirado atraso para geração do evento de ring.
- Firmware agora manda o evento de ring apos o ring e não no inicio do Ring.
- Versão inicializada com zero. Para correta exibição da versão é necessário esperar um tempo (~500ms) após o StartVoicerLib.
- O PlayCardinal agora permite falar em feminino, bastando passar um F no começo do numero. Ex: 22 fala Vinte e dois. F22 fala Vinte e duas.

- Reforma na função de extenso, tornando mais fluente, retirado uns "es" a mais, etc..
- O PlayList gera erro se o primeiro arquivo tiver PauseBefore igual a zero. Estou evitando isso chamando o timer independente do tempo passado.
- Corrigido problema de saturação no SetRecordGain. Agora o ganho pode chegar até a dez, diminuindo a diferença entre os interlocutores. O padrão é 2 e deve ser utilizado caso a diferença entre as vozes na gravação seja muito grande.
- Novo método SetFrequencyTime(smallint Duration) que permite alterar o tempo mínimo para detecção do tom de discagem. O padrão é 1500ms que permite detectar os tons de discagem padrão. Só deverá ser alterado em casos de sinalizações específicas do PABX.
- Corrigido problema de receber um RING após o atendimento, caso este tenha sido feito "em cima" do RING.

Versão 2.68- 24/11/2003

- Incluídas funções para tratamento de conferencia.
- O timer geral foi reduzido de 100ms para 80ms tornando-o mais próximo da realidade.
- Corrigido problema de atraso no StopRecordFile.
- SetRecordGain - Ganho de gravação - de 1 a 10 ganho padrão é 4.
- Detecção de fax corrigida.
- Correção da reprodução nos valores por extenso. Ex.: Se informa-se 5,6 o sistema entenderá 5,60.
- Na reprodução de número digitado no prompt é dado uma pausa de 400ms antes de começar a reproduzir os numeros.
- MakeCall agora detecta fax no evento OnAfterMakeCall. IMPORTANTE: O OnFaxDetected não é mais chamado se o fax for detectado via AfterMakeCall.
- Criada constante mkFaxDetected (valor 8) que é passada como parâmetro no AfterMakeCall

- Melhorada consistencia de inicialização. Se outra aplicação estiver usando a mesma placa o StartVoicerLib voltará erro 9.
- Reformulação das rotinas de gravação da placa PCI/4 com significativa melhora na qualidade do audio

Versão 2.67 - 29/08/2003

- Correção de erro no Windows98

Versão 2.66 - 26/08/2003

- Novo setup, sem alterações no funcionamento.
- Aumentado buffer de gravação para evitar truncamento.

Versão 2.65 - 28/07/2003

- Otimização do gerenciamento de memória quando existe apenas 1 placa instalada, resolvendo o problema de código 0 em máquinas com Windows 95 e 64MB de memória.
- Corrigido problema de instalação no Windows 98SE.

Versão 2.64 - 16/07/2003

- Corrigido o problema das funções de Menu que não apaga o buffer de dígitos no caso de digitos inválidos antes de receber a opção correta.
- Foi corrigido o problema introduzido pela versão 2.63 que causava "zumbidos" eventuais na reprodução de arquivos em situações com 4 ou mais canais.

Versão 2.63 - 24/06/2003

- Novo formato de gravação com compressão - ffGsm610 - Este formato gera arquivos de 1,65Kb/s de gravação (o Wave gerava 8Kb/s). A constante ffGsm610 equivale ao valor 3 e deve ser utilizada na propriedade **FileFormat** ou nos novos métodos **SetPlayFormat** e **SetRecordFormat**.
- Novos métodos **SetPlayFormat** e **SetRecordFormat** que permite utilizar formatos de gravação e reprodução independentes por canal. 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 estes métodos é possível manter formatos de gravação e reprodução diferentes no mesmo canal ou em canais distintos.
- Novos métodos **WaveToGSM** e **GsmToWave** permitindo a conversão de arquivos Wave (ffWave ou ffWavePCM) para o GSM ou do formato GSM para o WavePCM.
- Foram eliminados os métodos **ReadFifoToApp** e **WriteFifoToCard** que permitiam troca de amostras de áudio entre a aplicação e a placa. Estes métodos suportavam esta troca por apenas um canal. Para a próxima versão estaremos disponibilizando uma solução completa para este tipo de funcionalidade suportando até 24 canais.

Versão 2.62 - 14/05/2003

- Novo device driver chamado kpvlib.sys/vxd
- Suporte à Windows 2000 SP3 e Windows XP Pro SP1
- Limites do Twist alterado para 0 e 99
- Twist1 padrão alterado para 10
- Twist2 padrão alterado para 0
- Corrigido problema do PromptAbort que não interrompia o playback
- Retirado atende/desliga da inicialização para evitar desligamento de ligações em curso caso fosse necessário reativar o software.
- SetAnswerThreshold agora espera valores de 1 a 30 com

valor padrão 4, sendo que quanto maior o número, maior é a sensibilidade, exatamente ao contrário do que era antes...
- SetAnswerSensitivity modificado para o valor 6 ser o melhor possível.
- Novo programa de instalação.
- Exemplos em Delphi corrigidos com consistências melhoradas, vlib_diag melhorado, abrangendo a maioria das funções novas.

Versão 2.61 - 29/01/2003

- Criação do método SetToneTwist que ajusta a sensibilidade da detecção de tons. Foi criado para separar este efeito do SetDTMFTwist.

Versão 2.60 - 03/12/2002

- Colocado um delay de 2 segundos para que o evento de ring seja gerado somente depois do RING efetivamente ocorrer. Isto evita situações onde o evento de ring é gerado no meio do ring.

Versão 2.59 - 12/11/2002

- Novas mudanças na detecção de atendimento - fator padrão 6 para PCI/4
- Bug no Timeout da função MenuStart Corrigido - podia fazer com que o menu nunca desse timeout
- As funções que falam números, datas, etc... agora verificam se fala o arquivo wave ou sig de acordo com a propriedade FileFormat.
- Corrigido erro de não falar zero em cardinal
- Novo método SetAnswerThreshold que permite alterar o limiar para detecção de atendimento. Pode variar de 0 a 8192 -> se passar -1, no firmware o valor não é alterado o valor padrão é 440
- SetDTMFTwist - novo método para alterar a sensibilidade da

detecção de DTMF. Pode variar de 1 a 5 nos dois parâmetros. O primeiro parâmetro é a distância máxima entre as duas primeiras freqüencias e o segundo parâmetro é a distância entre as duas e a terceira freqüencia. O valor padrão no firmware é 5 e 2, respectivamente.

- Corrigido erro no GetDigits que afetava o Prompt. Quando se digitava o digito de confirmação após a mensagem, o GetDigits eliminava o digito da String deixando-a vazia. Agora o GetDigits só retira o terminador da string se ele não for o único digito.
- Corrigido problema de timeout de bina no Idlexxxx. Se chegasse uma identificação e a ligação nunca fosse atendida, o Idlexxxx não pegava mais nada. Foi criado um timer de 6 segs após o primeiro RING. Se estourar este timer sem a ligação ser atendida, os valores do bina são resetados (tmrTimeOutBINA).

Versão 2.58 - 20/09/2002

- Criação de um novo algoritmo para detecção de atendimento, tornando esta mais eficiente.
- O fator de sensibilidade de atendimento passou de 9 para 8.
- Aumento da sensibilidade de atendimento. Desta forma, os valores atualmente utilizados deverão ser reavaliados.

Versão 2.57 - 02/08/2002

- Novo parâmetro PauseAfterDigit do método SetCallStartFlash
- Novo parâmetro DialType do método Dial para determinar discagem por pulso ou tom independente por canal. A propriedade DialType se torna obsoleta e sem efeito.
- O método MakeCall agora aceita número de flashes igual a zero.
- Timeout entre tons de chamando aumentado para evitar falsos atendimentos durante o MakeCall
- Novo parâmetro **mkDialToneAfterDial** passado no evento

OnAfterMakeCall. Este valor de status indica que foi recebido um tom de linha após a discagem que indica provável problema no PABX/Linha.

- **Versão 2.56 – 17/07/2002**

- Novo sistema de help on line
- Erro no timer do MakeCall corrigido. Não dava timeout de tom de discagem.
- Criado parâmetro novo no SetCallAfterAnswer. O AutoHangUp indica se, após uma discagem do tipo Flash, desliga automaticamente quando detectado o atendimento. Se for false, gera o evento OnAfterMakeCall com mkAnswered porem o HangUp tem que ser dado na aplicação.
- Na inicialização da biblioteca, esta prevendo a PCI de 1 canal com PLX 9050 ou 9052 para evitar erros de instalação no Win 98
- Criado evento OnCallStateChange que permite monitorar a evolução de um makecall

Versão 2.54 – 20/05/2002

- Possibilidade de ligar até 6 placas PCI por computador
- Novo método SetAnswerSensitivity que permite customizar a sensibilidade de detecção de atendimento
- Corrigido problema de não detecção do dígito # durante a reprodução de frases
- Retirado obrigatoriedade de ter digito terminador no PromptStart
- Retirada obrigatoriedade de ter frase inicial no PromptStart
- O método PromptStart estava apagando o buffer de dígitos (ClearDigits) automaticamente. Este controle deve ficar especificamente com o programador.

Versão 2.5 – 26/03/2002

- Suporte à nova placa VoicerPhone PCI/1
- Nova propriedade ForcePlay
- Correção no método PlayTime quando tentava reproduzir horas com minutos igual à zero.
- Correção de problema na propriedade CardType em design-time. Ao iniciar a biblioteca a propriedade era ignorada forçando a atribuir o valor do tipo da placa no código.

Versão 2.4 –09/01/2002

- Bug nos métodos SetDTMFAttenuatingHigh e SetDTMFAttenuatingLow corrigidos
- A propriedade FileFormat não era alterada caso fosse setada apenas em design-time permanecendo sempre com o valor ffWave. Problema corrigido
- Correção do método Sig2Wave que deixou de funcionar corretamente na versão anterior.
- Correção do envio de comandos para o hardware. Desta maneira não é mais necessário utilizar o Sleep para dar uma pausa entre cada comando na inicialização.
- Novos métodos MakeCallxxxx que simplifica as funções de discagem/transferência com ou sem supervisão.
- Novos métodos Menuxxxx que reúne as funções de menu de atendimento em poucos parâmetros.
- Novos métodos Promptxxxx que simplificam as funções de entrada de dados com conferência e consistência do que foi digitado.
- Novos métodos Idlexxxxx que reúnem funções de atendimento automático e deteção de dígitos antes e depois do atendimento (BINA, sinalização do PABX,etc...)

Versão 2.3 – 06/11/2001

- Pequenas correções nos procedimentos de instalação
- Software de Diagnósticos corrigido
- Mudança internas nos procedimentos de interrupção de gravação, visando corrigir problemas de gravação quando

uma gravação era interrompida e outra iniciada rapidamente.

Versão 2.2 – 20/10/2001

- Implementado o método RecordPause, permitindo suspender a gravação e em seguida retomá-la.
- Correção do parâmetro TermDigits do GetDigits, PlayFile, etc... que estava sendo utilizado como Filtro de dígitos e não como digitos terminadores.
- Implementada a função de falar número digito a digito, através do método PlayNumber. Foi criado um método chamado PlayCardinal para falar os números cardinais (ex PlayNumber). No PlayListxxxx foi criado um tipo ptNumber para suportar este novo tipo.
- Implementado suporte a gravação de mensagens em formato wave definido pela propriedade FileFormat. Todas as placas e todos os canais deverão funcionar com o mesmo formato obrigatoriamente.

Versão 2.1 – 12/09/2001

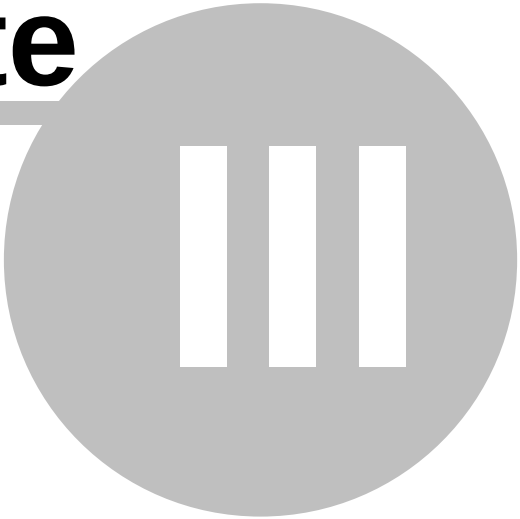
- Corrigido problema de não zerar a variável Duration do evento evento OnRecording a cada RecordFile
- Corrigido problema de travamento da placa ISA no Windows 2000 em certas condições
- Adicionado um exemplo de atendimento automático em VB 6.0
- Adicionado os fontes de um programa para conversão de arquivos Wave para SIG em lote, feito em Delphi.
- Adicionados arquivos SIG padrão para as funções de reprodução de extenso, data, etc. Disponíveis no diretório StockSigs do CD.
- Corrigido problema de conversão de arquivos Wave para SIG quando os waves são gerados com programas que adicionam informações no final do arquivo (ex.: SoundForge), o que causava "estalos" no final arquivo SIG
- Implementados novos métodos para falar valores por extenso, numerais cardinais, data e hora.

- Implementados métodos para tratamento de listas de mensagens.
- Se for detectado um dígito durante a reprodução de mensagens, além de gerar o evento OnPlayStop, também é gerado o evento OnDigitsReceived com o parâmetro edDigitOverMessage permitindo que o tratamento de entrada de dados seja feito em um lugar apenas.

Versão 2.0 – 01/08/2001

- Nova versão com suporte a placa PCI e múltiplas placas por micro

Parte



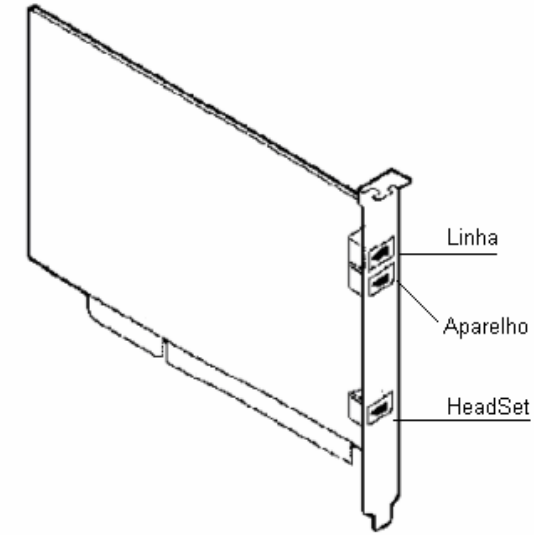
Iniciando

3 Iniciando

3.1 Cabos & Conectores

VoicerPhone ISA/PCI/1

Existem 3 conectores na placa, conforme mostrado na figura 1: Linha, Aparelho e HeadSet.



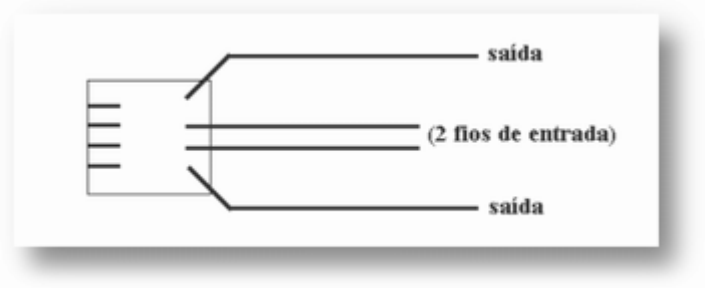
No conector Linha deverá ser ligado um ramal de PABX ou uma linha comum.
No conector Aparelho poderá ser ligado um aparelho telefônico comum.
No conector HeadSet poderá ser ligado um HeadSet.

VoicerBox PCI/4

Caso a placa VoicerBox PCI/4 seja destinada a funções de atendimento automático, URA, etc... basta conectar as linhas ou ramais diretamente em cada uma das 4 entradas disponíveis.

No caso de Gravação Digital da conversação, a VoicerBox PCI/4 ficará posicionada entre o trono e o destino (telefone ou PABX). Neste caso é necessário utilizar um cabo em "Y" já que a entrada e a saída da linha dar-se-á pelo mesmo conector.

Este cabo "Y" utilizará 4 fios, sendo dois para a entrada e dois para a saída, conforme pode ser observado no esquema abaixo:



3.2 Descrição do Produto

A VoicerLib é uma biblioteca contendo um componente ActiveX. Este componente fornece um acesso de alto nível para as placas Digivoice.

Atualmente a VoicerLib suporta a placa VoicerPhone ISA ou PCI (1 canal) e a VoicerBox PCI (4 canais), sendo possível gerenciar até 1 placa ISA ou 6 PCI em um mesmo equipamento. A partir da versão 2.5 é possível utilizar também a nova placa PCI/1 que vem substituir a placa ISA.

Sempre quando uma função for relacionada com a placa ISA assumir que esta função também funcionará na PCI/1, exceto s relativas à interrupção e endereço de I/O

Como principais funções podemos destacar:

- Atendimento e Desligamento
- Gravação e Reprodução de Mensagens
- Controle de Ganho (Volume)
- Detecção de Pulso e Tom
- Discagem por Pulso e Tom
- Controle de Microfone e Fone individuais, etc.

Na biblioteca foram incluídos exemplos de aplicação para Delphi e Visual Basic.

3.3 Instalando a Biblioteca

A VoicerLib é distribuida em um CD. Ao colocar o CD na unidade, aparecerá uma tela inicial mostrando as opções de instalação.

Caso esta tela inicial não apareça automaticamente, execute o programa IntroCD.exe no diretório raiz (usualmente D:\).

Opções da Tela Inicial:

- **Instalar a VoicerLib** - permite instalar a biblioteca de desenvolvimento e programas exemplo na máquina onde os aplicativos serão desenvolvidos
- **Instalar o Driver** - Esta opção deve ser utilizada somente para instalar o device driver nas máquinas onde a aplicação será utilizada.

ATENÇÃO: A licença de uso da VoicerLib permite a instalação somente no ambiente de desenvolvimento

portanto instalando-a em máquinas de produção você estará violando a licença de uso. Utilize sempre a opção Instalar o Driver nas máquinas dos clientes.

Ao executar o programa de instalação aparecerão as instruções. Basta seguir as telas que a VoicerLib instalará normalmente.

Após a instalação será criado um grupo de programa Digivoice VoicerLib 2.0 com alguns atalhos que serão explicados logo adiante.

Procedimentos de Instalação

Instalação Nova ***(****Em uma máquina que nunca teve versões anteriores da VoicerLib e/ou Kit Integrador.)*

1. Desligar o computador, desconectando o cabo de força da tomada
2. Conectar fisicamente a placa em um slot PCI disponível
3. Ligar o computador.
4. Se o Windows reconhecer o novo hardware, cancele a instalação do mesmo
5. Execute o **setup.exe** que se encontra na pasta **Driver**
6. Durante a instalação podem ocorrer alguns eventos dependendo da plataforma utilizada:

a. Win98/Me – Durante a instalação será pedido o **Disco de Instalação Digivoice**. Neste caso, aponte para a pasta **c:\arquivos de programas\VoicerLib2** pois lá estão todos os arquivos necessários para a correta instalação.

b. Win95 - Não é mais suportado pela VoicerLib.

c. WinNT - Não é mais suportado pela VoicerLib.

d. Win2000/2003/XP – Todo o processo nestas versão é automatizado mas durante a instalação o Windows mostrará

que está instalando um driver não certificado. Esta certificação é um procedimento burocrático da Microsoft, portanto basta reponder SIM cada vez que a pergunta aparecer. *Caso o progresso de instalação fique parado por muito tempo, é possível que uma destas janelas tenha aparecido atrás do programa de instalação. Na dúvida, dê um Alt+Tab para ver se não há nada por trás.* Em alguns casos pode ser pedido o disco de instalação. Caso isso ocorra, aponte para a pasta **c:\arquivos de programas\VoicerLib2** pois lá estão todos os arquivos necessários para a correta instalação.

Atualização ***(****Em uma máquina com versões anteriores da VoicerLib/Kit Integrador instaladas)*

1. No Gerenciador de Dispositivos (Painel de Controle -> Sistemas) remover (desinstalar) todas as ocorrências dos dispositivos VoicerPhone PCI/1, VoicerBox PCI/4 e Windriver.

2. Execute o **setup.exe** que se encontra na pasta **Driver** (Kit Integrador) ou na pasta **Instalar** (VoicerLib)

3. Durante a instalação podem ocorrer alguns eventos dependendo da plataforma utilizada:

· **Win98/Me** – Durante a instalação será pedido o **Disco de Instalação Digivoice**. Neste caso, aponte para a pasta **c:\arquivos de programas\VoicerLib2** pois lá estão todos os arquivos necessários para a correta instalação.

· **Win2000/2003/XP –** Todo o processo nestas versão é automatizado mas durante a instalação o Windows mostrará que está instalando um driver não certificado. Esta certificação é um procedimento burocrático da Microsoft, portanto basta reponder SIM cada vez que a pergunta aparecer. *Caso o progresso de instalação fique parado por muito tempo, é possível que uma destas janelas tenha aparecido atrás do programa de instalação. Na dúvida, dê um Alt+Tab para ver se*

não há nada por trás. Em alguns casos pode ser pedido o disco de instalação. Caso isso ocorra, aponte para a pasta **c:\arquivos de programas\VoicerLib2** pois lá estão todos os arquivos necessários para a correta instalação. **Importante:** O Windows 2000/XP tem a tendência de utilizar os INFs que já estão na pasta Windows\INF ao invés dos novos fornecidos para esta versão. Para garantir que tudo esteja atualizado, vá ao Gerenciador de Dispositivos onde aparece as placas Digivoice, escolha a opção Propriedades e depois na tela que aparece, a opção Atualizar/Reinstalar Driver. Neste caso aponte explicitamente para **c:\arquivos de programas\VoicerLib2.** Todos os dispositivos Digivoice deverão aparecer com um *(*)* no final do seu nome.

Observações Importantes:

- O programa vlib_diag é instalado na pasta Windows, portanto pode ser executado diretamente pelo Menu Iniciar -> Executar
- Se existir placas de barramento ISA, não se esqueça de executar o programa GeraIni.exe para configurar a interrupção e endereço de I/O.
- **Nunca instale ou remova uma placa com o computador conectado na rede elétrica, mesmo desligado.**
- Para instalar o Kit Integrador sem que apareça as janelas de diálogo, execute **setup.exe /silent** ao invés de **setup.exe /q**

3.4 Instalando o Hardware

Cuidado! Certifique-se de estar habilitado para executar a instalação do Hardware. O Computador deverá estar desligado.
A instalação é diferenciada para a versão ISA ou PCI.

Ignore a explicação da versão diferente da sua.

VERSÃO ISA

A placa VoicerPhone ISA exige que você configure manualmente a Interrupção e o Endereço de I/O que ela usará para se comunicar com o computador. Sai de fábrica utilizando a Irq7 e o Endereço de I/O 2a0h.

A tabela 1 mostra a posição dos jumpers na placa.

Tabela 1

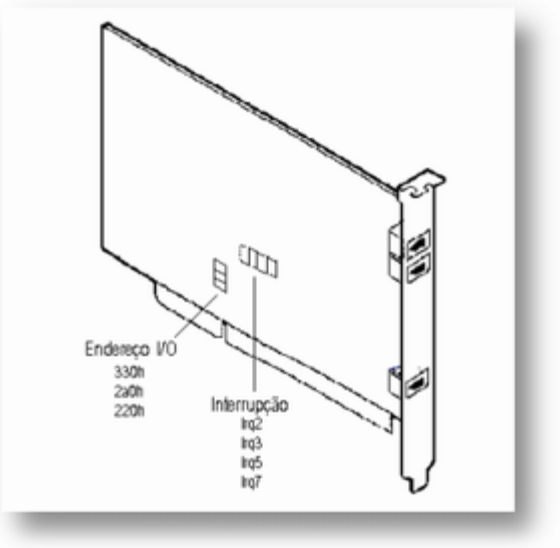


Figura 1

Certifique-se que a Interrupção não esteja sendo utilizada por outro dispositivo através do Painel de Controle à Sistema à Gerenciador de Dispositivos à Meu Computador à Propriedades.

Para instalar a Placa Voicer Box – ISA:

1. Desligue e desconecte a energia do seu computador.
2. Remova a tampa do gabinete do seu computador.
3. Escolha um slot ISA de 16 bits e remova a tampa do slot (pequena tampa de metal na parte traseira do computador).
4. Segure a placa pelo topo (figura 2a) colocando-a no slot até estar completamente encaixada.
Não toque os contatos dourados da placa pois qualquer impureza nos contatos pode prejudicar o funcionamento do equipamento.

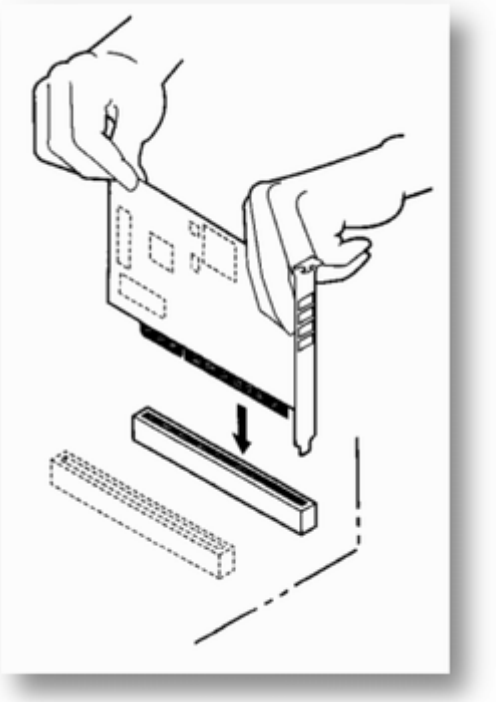


Figura 2a

5. Fixe a aleta da placa no gabinete do seu computador com um parafuso de fixação.
6. Recoloque a tampa do seu computador.
Ao religar o computador, a placa não será identificada pelo Windows por não ser uma placa ISA Plug & Play. Isto não afetará o seu funcionamento.
VERSÃO PCI

Para instalar a Placa Voicer Box – PCI/4 ou Voicer Phone PCI/1:

1. Desligue e desconecte a energia do seu computador.
2. Remova a tampa do gabinete do seu computador.
3. Escolha um slot PCI de 32 bits e remova a tampa do slot (pequena tampa de metal na parte traseira do computador).
4. Segure a placa pelo topo (figura 2a) colocando-a no slot até estar completamente encaixada.
Não toque os contatos dourados da placa pois qualquer impureza nos contatos pode prejudicar o funcionamento do equipamento.
5. Fixe a aleta da placa do Voicer Box no gabinete do seu computador com um parafuso de fixação.
6. Recoloque a tampa do seu computador.
A placa PCI não tem jumpers de configuração pois os recursos necessários são alocados automaticamente pelo computador no momento do boot.
7. Ligue o computador.
8 O Windows detectará automaticamente a placa VoicerBox PCI/4 normalmente identificando-a como Dispositivo de Áudio . Ao aparecer a tela de procura de Driver, aponte para o CD na pasta Driver\InfParaPCI . Desta maneira o Windows detectará e instalará a placa corretamente. Caso a máquina não tenha unidade de CD disponível, copie o conteúdo da pasta Driver\InfParaPCI do CD para um disquete e utilize-o.

Para instalar placas PCI no seu cliente, leia o capítulo Distribuindo sua Aplicação.

3.5 Preparando o Ambiente

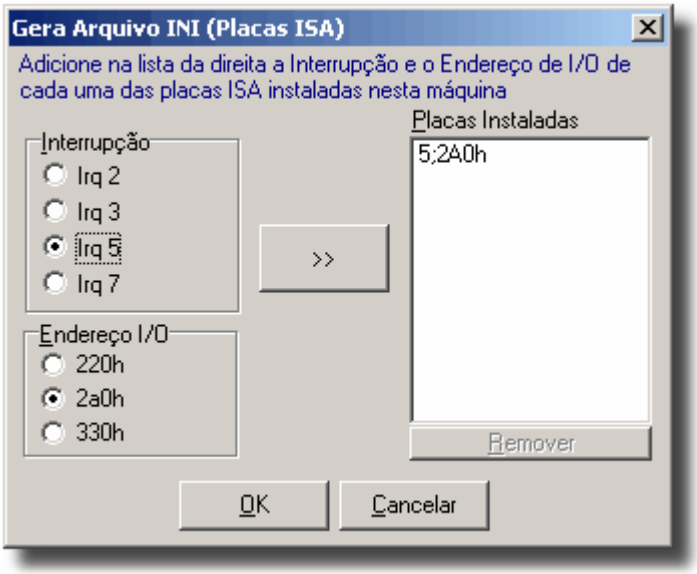
Considerando que a placa (ISA ou PCI) já está instalada no computador,o próximo passo será testá-la através do programa de Diagnóstico que foi fornecido junto com a biblioteca.

Placa ISA

No caso da placa ISA, é necessário configurar o Windows com a quantidade de placas, interrupções e endereços de I/O utilizados. Isto é possível através do utilitário Configurar placas ISA que pode ser acessado através do grupo de programas Digivoice VoicerLib 2.0.

Lembre-se de verificar se não há conflitos de interrupção com outro hardware instalado.

Ao clicar no atalho Configurar placas ISA aparecerá a seguinte tela:



Para adicionar uma placa selecione a Interrupção e o endereço de I/O desejado e clique no botão *">>"* .

Para remover, selecione a placa desejada na lista de Placas Instaladas e clique no botão *Remover*.

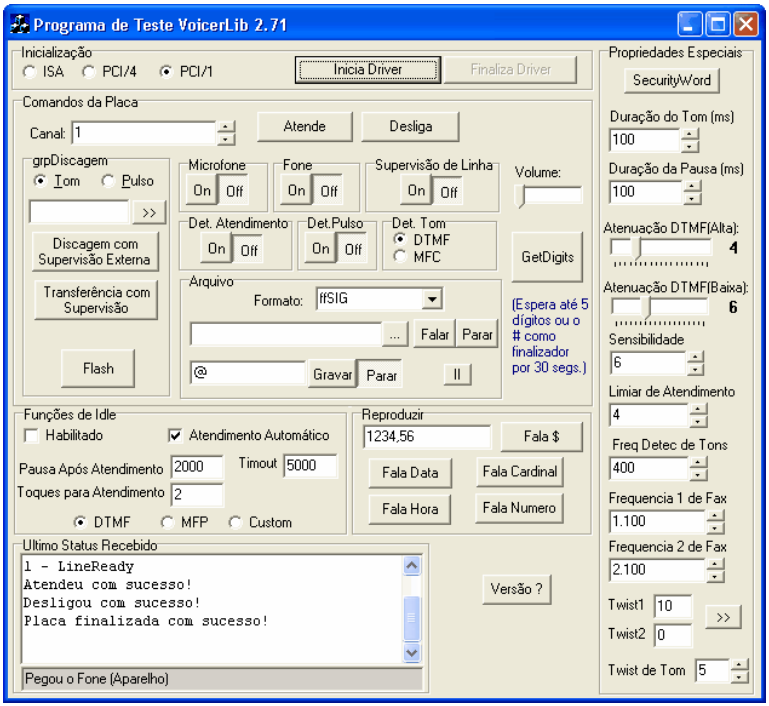
Para gravar a configuração, clique no botão *OK*.

Placa PCI/4 ou PCI/1

Não é necessário executar o programa Configurar placas ISA para as placas PCI pois a interrupção e endereço de I/O são atribuídos automaticamente.

Testando o Hardware

Para testar a placa instalada, execute o Programa de Diagnósticos encontrado no grupo de programa Digivoice VoicerLib 2.71.Ele tem a seguinte interface:



Para inicializar o hardware, primeiro selecione o tipo de placa instalada (ISA ou PCI) e clique no botão Inicia Driver.

Devido ao maior rigor na inicialização do hardware, este processo ficou ligeiramente mais lento que na versão 1.x

Caso ocorra alguma falha, você receberá uma mensagem com um código:

Para saber o significado dos códigos de erro, consulte o método **StartVoicerLib** no capítulo **Guia de Referência**.

Caso a inicialização seja feita com sucesso, você poderá testar as funções de Atende e Desliga, discagem, etc...

> *Lembre-se de conectar uma linha ou ramal na placa antes de efetuar os testes.*
>
> *Algumas funções são pertinentes somente a placa VoicerPhone ISA e portanto não tem efeito nas placas PCI. Maiores detalhes sobre isso no capítulo Tópicos de Programação.*

Nas placas ISA, caso o programa inicie corretamente, mas ao clicar no botão Atende e Desliga você não obtiver resposta é possível que a interrupção esteja configurada incorretamente ou em conflito com outro hardware.

3.6 Requisitos Mínimos

A princípio a VoicerLib rodará em qualquer plataforma que comporta o Windows com performance, entretanto, para melhor utilização recomendamos:

- Windows 95/98/Me/NT 4/2000/XP
- Pentium 233Mhz
- 64MB de memória

- 5MB de espaço livre em disco
- Slots ISA (VoicerPhone) ou PCI (PCI/1 ou PCI/4) disponíveis
- Delphi 5 ou maior e Visual Basic 6 ou maior.

Importante: O código da biblioteca foi otimizado para rodar em máquinas Pentium, portanto o sistema não funcionará em 486 ou menor.
Máquinas Cyrix ou AMD compatíveis com Pentium não terão problemas.

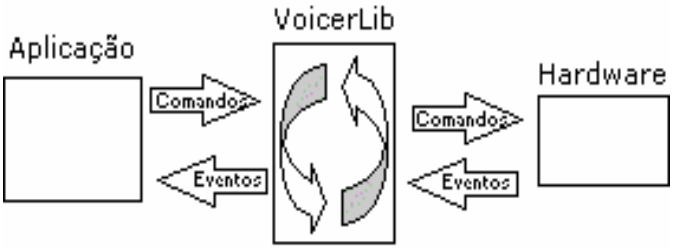
Parte

Tópicos de Programação

4 Tópicos de Programação

4.1 Estrutura Básica de Funcionamento

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.

Como esta forma de programar é um pouco diferente da que a maioria dos programadores sem experiência em aplicações de telefônia em tempo real estão acostumados, recomenda-se ler todo este capítulo: Aplicação Exemplo Passo a Passo e estudar os exemplos fornecidos.

4.2 Inicializando os Serviços

Sempre que iniciar a aplicação, antes de executar qualquer método, é necessário iniciar os serviços (device driver). Isto pode ser feito através do método StartVoicerLib.

É possível verificar se os serviços do hardware foram iniciados corretamente através do retorno da função que deve ser 99. Os outros códigos de erro podem ser consultados no Guia de Referência deste manual.

Exemplo:

```
Private Sub Form_Load()
   Dim x as Long
    'Inicia device driver
    x = VoicerLibX1.StartVoicerLib
    'se for diferente de 99 significa que deu
erro
    If x <> 99 Then
        MsgBox "Não foi possível inicializar a
placa", vbOKOnly, "Erro"
    End
End Sub
```

Como a inicialização é a primeira coisa a ser feita, recomenda-se sua colocação no início da aplicação.

4.3 Finalizando os Serviços

Antes de fechar a aplicação é necessário chamar o método **ShutdownVoicerLib**, que finaliza todos os serviços e reseta a placa.

Caso o método não seja chamado, 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 do método **ShutdownVoicerLib** poderá causar travamento no Windows ou algum comportamento imprevisível.

É possível verificar se os serviços do hardware foram finalizados corretamente através do retorno da função, que deve ser 99.

Exemplo:

```
Private Sub Form_UnLoad()
     Dim x as Long
    'Finaliza device driver
    x = VoicerLibX1.ShutdownVoicerLib
    'se for diferente de zero significa que deu
erro
    If x <> 99 Then
        MsgBox "Não foi possível finalizar a placa"
+ _
                          " corretamente. Reinicie o
micro", _
vbOKOnly, "Erro"
        End
End Sub
```

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

4.4 Detectando Ring

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

Sempre quando o ring for detectado, o evento OnRingDetected é gerado.

Exemplo:

```
'Chegou o ring, então atende
Private Sub VoicerLibX1_OnRingDetected(Port As Integer)
    lblStatus.Caption = "RING!! – Atendi!!!"
   VoicerLibX1.PickUp(1)
End Sub
```

Como o Ring ocorre em situações bem específicas a sua detecção está sempre habilitada não havendo métodos para ligar ou desligar.

4.5 Atendendo e Desligando

Para atender a ligação ou tomar uma linha para discagem, o método que deve ser utilizado é o **PickUp**. O método 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 OnAfterPickUp. O capítulo Aplicação Exemplo Passo a Passo explicará bem esta relação.

Para desligar a ligação, o método utilizado é o **HangUp**. Deve ser passado somente a porta que se deseja desligar.

Exemplos:

```
'Atende a ligação ou toma a linha para discar na
porta 3
Private Sub cmdAtende_Click()
    x = VoicerLibX1.PickUp(3,0) 'sem pausa
End Sub

'Desliga a linha
Private Sub cmdDesliga_Click()
    x = VoicerLibX1.HangUp(3)
End Sub
```

A variável "x" nos exemplos recebe zero se os comandos foram executados com sucesso. Para deixar o aplicativo mais completo, recomenda-se testar "x" sempre.

No caso da placa **Voicer Phone**, ao tomar a linha (PickUp) da primeira vez que o aplicativo é executado, o usuário provavelmente não ouvirá nada, pois o microfone e o fone do headset estarão desabilitados.

4.6 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
- Detecção de sinal de chamada
- Detecção de Atendimento

É importante ressaltar que não é aconselhável ficar com estes sinais habilitados o tempo todo, porque é possível que aconteça uma interpretação errada por parte do hardware. Uma situação típica que esclarece bem a questão é com relação à detecção

de atendimento.

Ela deve ser habilitada somente no momento que você quiser detectar o atendimento, que via de regra seria após a discagem, e desabilitada logo após o atendimento ter sido detectado. Como o atendimento é detectado através de certos ruídos na linha, se a detecção estiver habilitada durante a conversação, por exemplo, um ruído qualquer durante a conversa pode ser entendido como atendimento.
Isto vale para todos os tipos de detecção, portanto, sempre mantenha habilitada a detecção que seja necessária em um determinado momento.

Sempre quando um destes sinais for detectado, o evento correspondente será chamado.

O sinal de ocupado, fax, tom de discagem e chamada são controlados pelos métodos EnableCallProgress e DisableCallProgress. Já a detecção de atendimento é controlada em separado pelos métodos EnableAnswerDetection e DisableAnswerDetection.

> *A partir da versão 2.61, o método SetToneTwist permite controlar a sensibilidade de detecção destes tons quando gerados em cima de uma mensagem. Este método deve ser utilizado caso a detecção de tons (notadamente o ocupado) esteja gerando situações de talk-off*

Sinal de Ocupado

Ao ser detectado o sinal de ocupado, o evento OnBusyDetected é gerado, podendo ser tratado pelo usuário da maneira como quiser.

O evento é gerado sempre que for detectado o ocupado, portanto, caso o sinal de ocupado permaneça por muito tempo sem desligar, o programa desviará várias vezes para o OnBusyDetected. Uma saída é logo após o primeiro ocupado, desabilitar a detecção com o método DisableCallProgress.

Outro efeito que pode ocorrer muito raramente é o sinal de ocupado ser detectado durante a conversação. Isto acontece com certos tons e candências 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.

É sempre recomendado não fazer tratamentos muito demorados dentro das rotinas que manipulam os eventos para evitar que a biblioteca perca outros eventos que possam chegar em seguida. NUNCA chame MessageBox ou outras janelas modais de dentro destes eventos.

Exemplo:

```
Private Sub VoicerLibX1_OnBusyDetected(Port As
Integer)
    lblStatus.Caption = "Sinal de Ocupado..."
    VoicerLibX1.DisableCallProgress 1
End Sub
```

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 OnFaxDetected.

Exemplo:

```
Private Sub VoicerLibX1_OnFaxDetected(Port As
Integer)
    lblStatus.Caption = "Transferindo para o
Fax..."
   'executa comando de flash para transferencia
    VoicerLibX1.Flash(1,600,1000)
   'disca para o ramal do fax
```

```
     VoicerLibX1.Dial(1,"202',1000)
End Sub
```

Detecção de Atendimento

A VoicerLib permite ao programador detectar quando uma ligação foi atendida do outro lado da linha. Os métodos EnableAnswerDetection e DisableAnswerDetection controlam a ocorrência ou não deste evento.

O evento OnAnswerDetected é gerado sempre quando este sinal for detectado. Uma aplicação exemplo seria uma rotina esperar o atendimento para então discar o número do ramal desejado

Exemplo:

```
Private Sub VoicerLibX1_OnAnswerDetected(Port As
Integer)
    'Primeiro desabilita a detecao para nao
entrar aqui
    'de novo
    VoicerLibX1.DisableAnswerDetection(1)
    lblStatus.Caption = "Atendeu, discando para o
ramal"
   VoicerLibX1.Dial(1,"220',0)
End Sub
```

A detecção de atendimento pode ter sua sensibilidade alterada através do método SetAnswerSensitivity. Nele é possível passar parâmetros de 1 (menos sensível) até 10 (mais sensível). O padrão é 6. A variação de sensibilidade pode ser necessária em situações onde esteja difícil de pegar o atendimento ou o outro extremo: estar detectando atendimento falso.

Detecção de sinal de discagem

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 pelos métodos EnableCallProgress e DisableCallProgress.

O evento OnDialToneDetected é gerado sempre quando este sinal for detectado.

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.

Exemplo:

```
'Habilita Supervisão de linha
'e toma a linha para discar a partir do click do
botão
Private Sub cmdAtende_Click()
    VoicerLibX1.EnableCallProgress(1)
    x = VoicerLibX1.PickUp(1,0)
End Sub

'Quando o tom de discagem for detectado
'esta rotina é chamada
Private Sub VoicerLibX1_OnDialToneDetected(Port
as Integer)
   'disca
   VoicerLibX1.Dial("0,72952557")
End Sub
```

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 pelos métodos EnableCallProgress e 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 OnCalling ocorre sempre quando o sinal for detectado, o que faz com que ele seja chamado várias vezes até a ligação

ser atendida.

Exemplo:

```
Private Sub VoicerLibX1_OnCalling(Port As
Integer)
nToques = nToques + 1
lblStatus.Caption = "Chamando... " & nToques
if nToques = 5 then
   VoicerLibX1.HangUp(1)
End if
End Sub
```

Neste exemplo, após o 5˚ toque, o sistema desiste e desliga. (Supor que existe uma variável global chamada nToques inicializada com zero)

4.7 Detecção de Dígitos

A VoicerLib permite detectar dígitos tanto em tom como em pulso, entretanto a deteção de pulso é desabilitada por default podendo ser habilitada pelo método EnablePulseDetection e desabilitada pelo método DisablePulseDetection.

> *A deteção por tom esta sempre habilitada e disponível. Não há como desabilitá-la*

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

> *Habilite a deteção de pulso somente quando for necessário, pois caso contrário, uma deteção de pulso habilitada, por exemplo, durante a reprodução pode causar inúmeros talk-offs*

A VoicerLib permite tratar o reconhecimento de dígitos de duas

maneiras diferentes.

A primeira é através do evento OnDigitDetected, que é gerado sempre que um digito 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 neste caso estará habilitada o tempo todo.

É uma maneira pouco recomendada por causa problema de talk-off que pode ocorrer durante o atendimento.

> *TALK-OFF: A voz humana, em uma conversa normal, pode conter a mesma freqüência 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 ter algum mal funcionamento operacional.*
> *Para mensagens faladas através de arquivos SIG, o problema de talk-off não ocorre, pois o hardware consegue filtrar as freqüências conhecidas.*

Uma aplicação típica que se recomenda a utilização do evento OnDigitDetected é na identificação de chamadas. Leia mais no tópico Identificação de Chamadas logo a seguir.

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 Digito " +
```

```
                                    Chr$(voicerlibx1.ReadDig
its(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 digito 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.

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.

A propriedade AutoClearDigits também interfere no funcionamento do método GetDigits, ou seja, se esta propriedade estiver true (default), os dígitos são apagados dos buffers internos dos canais quando o GetDigits for chamado; se estiver false os dígitos permanecerão até ser chamado o método ClearDigits(Porta).

Se for necessário apagar a propriedade Digits manualmente, utilize o 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.

A detecção de DTMF sai com um nível de sensibilidade que funcionará em 99% das instalações. Podem existir situações extremas nas quais a detecção pode parecer muito ou pouco sensivel. Neste caso, o programador poderá fazer uso do método **SetDTMFTwist** que permite alterar a sensibilidade desta detecção.

O padrão DTMF é composto de um par de freqüências (tons) para cada dígito discado que chamaremos respectivamente de F1 e F2.

Devido às características da linha telefônica e dos equipamentos a elas ligados, as amplitudes destes tons podem ter atenuações diferentes na propagação pela rede telefônica.

A presença de áudio juntamente com o par de tons do DTMF (quando a placa está falando uma mensagem) pode prejudicar sua detecção pois gera novos tons F3, junto com F1 e F2.

O parâmetro **TWIST1** diz respeito à tolerância na diferença de amplitude entre as duas freqüências do DTMF – F1 e F2. Quanto **maior** o valor de TWIST, maior a tolerância a variações de amplitude entre as duas freqüências ou mais fácil será a detecção de digitos *(podendo até chegar em situações de talk-off)*. Quando necessário, o ajuste desta variável deve ser feito sem que a placa esteja falando.

O parâmetro **TWIST2** diz respeito à seletividade de uma terceira freqüência F3 com relação às duas freqüências, F1 e F2 presentes no áudio recebido pela placa, isto quer dizer que quanto **menor** o valor de TWIST2, **maiores** poderão ser as amplitudes das outras freqüências presentes no áudio sem que seja rejeitada a detecção de DTMF, ou seja, os valores menores aumentam a sensibilidade de DTMF podendo chegar até situações de talk-off.

Os dois valores deverão ser ajustados empiricamente em situações que os dígitos não estejam sendo detectados a contento ou, no outro extremo, em situações de *talk-off*.

4.8 Discagem

A plataforma Digivoice permite discagem por Tom (DTMF) ou Pulsos, permitindo a utilização de aplicações baseadas em qualquer tipo de instalação ou central pública.

O método utilizado para discagem é o Dial que basicamente necessita da porta, de uma string contendo o número a ser discado, da pausa após a discagem e do tipo de discagem (Pulso ou Tom).

```
x = VoicerLibX1.Dial(1,"72952557",1000, dtTone)
```

O tipo de discagem permite cada canal discar por pulso ou tom de forma independente, sendo que o parâmetro pode assumir o valor dtTone para Tom (DTMF) ou dtPulse para Pulso (Decádico).

Em algumas situações é necessário dar pausas entre cada dígito. Um caso típico é a ligação externa através do PABX que é necessário discar "0" e em seguida discar o número de destino.

A VoicerLib disponibilizou três símbolos que podem ser utilizados dentro da string do número que indicarão ao sistema para esperar "n" milissegundos antes de prosseguir. Estas propriedades são: DelayDot, DelayComma e DelaySemicolon que definem a pausa para, respectivamente, o "ponto", a "vírgula" e o "ponto-e-vírgula".

> ***O valor da pausa deve ser expresso em milissegundos, isto é, para esperar 1 segundo, atribuir o valor 1000.***

No exemplo abaixo é mostrado um caso onde é setado a discagem por pulso e é utilizado a vírgula como símbolo de pausa para efetuar uma ligação externa:

```
Sub cmdDisca_Click()
```

```
    VoicerLibX1.DelayComma = 1000
    VoicerLibX1.Dial(1,"0,72952557",1000, dtPulse)

End Sub
```

Neste outro exemplo, após a discagem do número, o sistema esperará 5 segundos para poder discar o ramal.

```
Sub cmdDisca_Click()
    VoicerLibX1.DelayComma = 1000
    VoicerLibX1.DelayDot = 5000
    VoicerLibX1.Dial(1,"0,72952557.220",1000,
dtTone);
End Sub
```

Além das pausas através das propriedades Delayxxx, existe a pausa após discagem (3º. Parâmetro) que também é representado em milissegundos.

Este parâmetro deve ser utilizado principalmente em casos de transferência com supervisão, pois existem centrais que geral ruídos após a discagem atrapalhando a deteção de atendimento, por exemplo. Se este parâmetro estiver com valor maior que 0 o evento OnAfterDial só será gerado após acabar a discagem e a pausa.

Como a discagem pode demorar vários segundos, a VoicerLib disponibiliza o evento AfterDial, que é gerado sempre quando um método Dial acabar de discar todos os números e pausas. Isto é muito útil para habilitar alguma supervisão de linha necessária após a discagem, como por exemplo, a detecção de atendimento (maiores detalhes sobre supervisão no tópico seguinte).

Aproveitando o exemplo anterior, após a discagem é possível fazer algo como no exemplo a seguir:

```
Sub frmExemplo.VoicerLibAfterDial()
    'Acabou de discar – espera atendimento
    VoicerLibX1.EnableAnswerDetection 1
End Sub
```

*Obs.: Até a versão **2.56**, a propriedade DialType determinada o tipo de discagem. A partir da versão **2.57** esta propriedade tornou-se obsoleta e o tipo de discagem deve ser passado em toda chamada ao método **Dial**.*

4.9 Configurações de DTMF

As placas Digivoice já saem de fábrica ajustadas para ser compatível com o maior número de centrais possíveis, entretanto é possível que existam situações que o tom gerado pela placa, principalmente nas discagem no PABX, possam não ser compreendidas.

Para que os sistemas desenvolvidos com a Voicer Lib possam atender 100% dos equipamentos, existem alguns métodos que permitem ajustar certos aspectos do tom gerado pela placa.

O método SetDTMFDuration permite modificar a duração de um tom, que é iniciada em 100 milisegundos. Normalmente este tempo funcionará sem problemas mas o programador poderá ajustar esta duração conforme a necessidade.

O método SetDTMFPause permite determinar a pausa inter-digitos em uma discagem por tom. Isso só é válido quando o número é passado completo no método Dial. Também tem como valor default 100 ms.

Os tons DTMF são gerados a partir de pares de freqüências pré-estabelicidas em normas internacionais. Os métodos SetDTMFAttenuatingHigh e SetDTMFAttenuatingLow permitem determinar índices de atenuação para cada uma destas duas freqüências.

Por padrão a freqüência alta é iniciada com 4 dB de atenuação e a freqüência baixa em 6 dB. Normalmente estes valores serão suficientes mas caso haja problemas na deteção de tons por

parte do PABX, estes valores poderão ser ajustados entre 0 e 50 dB.

Na prática, a diferença entre as duas freqüências não pode exceder 4 dB, ou seja, se a freqüência alta estiver com atenuação de 1 dB, a freqüência baixa não poderá estar com atenuação maior que 5 dB.

Se os dois valores estiverem em zero, significa que não há atenuação nenhuma.

> *Lembre-se: Na maioria dos casos estes valores não deverão ser alterados. Somente recorra a eles se a sua central não entender os tons de discagem.*

4.10 Controle de Volume

A placa VoicerPhone permite que o usuário controle o ganho do áudio em três níveis, sendo que já vem com o nível intermediário (voTwo) como padrão. Este ganho refere-se apenas ao headset do usuário, não tendo efeito na reprodução ou gravação de mensagens.

O controle de ganho ou volume pode ser feito em qualquer momento, mesmo durante uma conversa. Para alterar o volume, utilize o método SetVolume que pode assumir voOne, voTwo ou voThree. É interessante disponibilizar este controle para o usuário final, pois existem grandes variações na qualidade de áudio de instalação para instalação.

Exemplo:

Neste exemplo, utilizamos o controle Slider do Visual Basic para aplicar o volume.

```
Private Sub trkVolume_Change()
    'Modifica Volume de Acordo com a posição do
slider
    Select Case trkVolume.Value
        Case 1
            VoicerLibX1.SetVolume(voOne)
        Case 2
            VoicerLibX1.SetVolume(voTwo)
        Case 3
            VoicerLibX1.SetVolume(voThree)
    End Select
End Sub
```

4.11 Microfone

Como foi dito anteriormente, a placa VoicerPhone tem tratamentos diferenciados para o microfone do headset.

Como padrão, o microfone está desabilitado, portanto o interlocutor do outro lado da linha não poderá ouvir o que o usuário fala. Para habilitar o microfone, basta chamar o método **MicOn** e para desabilitar, **MicOff**.

Assim como os métodos **PhoneOn** e **PhoneOff**, é possível colocar os métodos **MicOn** e **MicOff** em qualquer ponto do programa.

Também é possível associá-los a um botão com a função de MUTE, permitindo que o interlocutor fique em espera sem ouvir o que o usuário fala.

Exemplo:

```
'Liga MUTE – Interlocutor não pode ouvir nada
Private Sub cmdMUTEON_Click()
    x = VoicerLibX1.MicOff(1)
```

```
End Sub

'Desliga MUTE – Interlocutor ouve
Private Sub cmdMUTEOFF_Click()
    x = VoicerLibX1.MicOn(1)
End Sub
```

Importante: As funções para microfone e fone não funcionam para o telefone, somente para o headset.

4.12 Fone do Headset

A placa VoicerPhone tem tratamentos diferenciados para o fone e o microfone do headset. Como padrão, o fone está desabilitado, portanto o usuário não poderá ouvir nada (voz, tom de discagem, ocupado, etc...).

Para habilitar o fone, basta chamar o método **PhoneOn** e para desabilitar, **PhoneOff**.

Isso pode ser feito associado a um botão por exemplo, ou ainda o programador pode automaticamente habilitar o fone quando uma ligação é atendida e desabilitá-lo quando a ligação é finalizada.

Exemplo:

```
'Atende a ligação e habilita o fone
Private Sub cmdAtende_Click()
    x = VoicerLibX1.PickUp(1,0)
       if x = 0 then
              'Liga o fone
              VoicerLibX1.PhoneOn(1)
       End if
End Sub

'Ao desligar, desabilita
Private Sub cmdDesliga_Click()
    x = VoicerLibX1.HangUp(1)
```

```
        if x = 0 then
               'Liga o fone
               VoicerLibX1.PhoneOff(1)
        End if
End Sub
```

4.13 Gravando uma Conversa

A VoicerLib permite gravar qualquer conversa tanto pelo headset como pelo monofone (aparelho comum). Para isso é necessário chamar o método RecordFile onde você deve informar o nome do arquivo que será criado.

O sistema utiliza o formato SIG ou o formato Wave (8 Bits, Mono, 8 Khz).

Além disso, é possível 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 na propriedade Digits, e pode ser tratado posteriormente.

Dependendo do fluxo de operação do sistema desenvolvido, é necessário que a propriedade Digits vá acumulando os digitos detectados. Neste caso é necessário mudar a propriedade AutoClearDigits para false, pois o padrão true faz com que a propriedade Digits seja automaticamente apagada quando uma gravação é iniciada.

Sempre quando for iniciada a gravação o evento OnRecordStart é gerado, permitindo tratamento nesta situação. Durante a gravação a propriedade Recording estará TRUE.

Ao término da gravação o evento OnRecordStop é gerado automaticamente, inclusive informando qual o motivo da

interrupção da gravação.

Este motivo pode ser uma ação direta do operador (com a chamada do método StopRecordFile), um dígito recebido ou ainda um erro qualquer (disco cheio, por exemplo).

Exemplo:

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

```
Private Sub DeixarRecado()
    '.... código para falar a mensagem ....
    'Inicia gravação
    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 digito 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
    End sub
```

Além disso, é possível monitorar o andamento da gravação através do evento OnRecording. O parâmetro Duration indica a quantidade de segundos decorridos até aquele momento.

A gravação pelo monofone conectado na VoicerPhone ISA ou através do cabo especial da VoicerBox PCI/4 pode ser iniciado através do evento OnLineReady e finalizada através do evento OnLineOff.

A partir da versão 2.2, está disponível o método RecordPause, que permite interromper temporariamente uma gravação e em seguida continuar no mesmo arquivo. Neste método é necessário passar a porta e uma flag booleano indicando se coloca em pausa (TRUE) ou retira da pausa (FALSE).

4.14 Identificação de Chamadas

A identificação de chamadas é mais uma característica da placas Digivoice o que permite a criação de aplicações voltadas para este tipo de aplicação.

A placa deverá estar conectada diretamente no tronco, e este deverá estar com a sinalização habilitada pela companhia telefônica.

A sinalização enviada pela companhia telefônica pode ser em DTMF ou MFP. Você deve verificar na companhia qual o tipo enviado pela sua central. Na VoicerLib, esta sinalização deve ser modificada através do método SetDetectionType(Porta). A sinalização só é enviada antes do primeiro ring, portanto

monitore-a através do evento OnDigitDetected somente com a placa desligada (HangUp).

O evento OnDigitDetected recebe todos os digitos detectados o tempo todo, com a placa atendida ou desligada portanto é importante saber em que estado está para evitar pegar talkoff durante a conversação.

Caso a sinalização seja em MFP, você deve, logo após o atendimento modificar o tipo de detecção para DTMF que é o padrão utilizado nos aparelhos telefônicos (o tom dos dígitos).

A VoicerLib também oferece os métodos Idle*XXX* que facilitam o processo de identificação de chamadas.

4.15 Reproduzindo Data

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

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

O primeiro parâmetro refere-se ao canal da placa. O segundo parâmetro e 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 milisegundos, 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.

4.16 Reproduzindo Hora

A VoicerLib permite ao programador implementar sistemas que falem hora através do método PlayTime.

O formato da função é:

```
vlb.PlayDate Porta, Time, "#", 0
```

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

A VoicerLib permite falar a hora no formato hh:mm:ss ou hh:mm. No parâmetro Value deve ser passado uma string com a hora separada por ":". Se você passar, por exemplo, "12:23:32" a VoicerLib entenderá que deve falar a hh:mm:ss. Se for passado apenas "12:32" a VoicerLib só reproduzirá "hh:mm".

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 milisegundos, 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.

4.17 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:

```
Voice1.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 milisegundos, 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:

```
voice1.PlayListAdd Port, ptFile,
"olahojeedia.sig","",0
voice1.PlayListAdd Port, ptDate, Date, "d/m/y", 0
voice1.PlayListAdd Port, ptFile, "as.sig", "", 0
voice1.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:

```
voice1.PlayListClear Port    'Apaga a lista antes
de usá-la
voice1.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 = voice1.PlayListGetCount(Port)
```

Removendo um ítem 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
voice1.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:

```
voice1.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 interrrompida 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.

4.18 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:

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

Até a versão 2.1 esta função era desempenhada pelo método PlayNumber, porém, devido à novas funcionalidades do PlayNumber de falar digito a digito, a função de falar o número cardinal foi colocada em um método a parte.

O primeiro parâmetro refere-se ao canal da placa. O segundo parâmetro e 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 décimal (nnnnnnn,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 milisegundos, 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.

4.19 Reproduzindo Números Dígito a Dígito

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:

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

Até a versão 2.1 esta função era desempenhada pelo método PlayNumber, porém, devido à novas funcionalidades do PlayNumber de falar digito a digito, a função de falar o número cardinal foi colocada em um método a parte.

O primeiro parâmetro refere-se ao canal da placa. O segundo parâmetro e 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 milisegundos, 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.

4.20 Reproduzindo uma Mensagem

A VoicerLib permite reproduzir qualquer mensagem gravada em formato SIG, tanto pelo headset como pelo monofone (ou tronco). Para isso basta utilizar o método PlayFile.

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, assim ele não precisará se preocupar em converter o arquivo antes de reproduzir. 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.

Dependendo do fluxo de operação do sistema desenvolvido, é necessário que os digitos detectados sejam acumulados. Neste caso é necessário mudar a propriedade AutoClearDigits para false, pois o padrão true faz com que os digitos sejam automaticamente apagados quando uma gravação é iniciada.

Sempre quando for iniciada a reprodução o evento OnPlaytStart

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).

A partir da versão 2.1, se um digito 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 tópico seguinte (detecção de dígitos) aborda este evento em detalhes.

Exemplo:

O sistema atende uma ligação no evento OnRingDetected, fala uma mensagem e fica esperando o dígito de uma das opções: "Digivoice, disque 3 para vendas, 4 para técnica ou 5 para fax". Dependendo da opção, disca para o ramal correspondente.

> *A partir da versão 2.1 recomenda-se fazer o tratamento de dígitos no evento OnDigitsReceived e não no OnPlayStop. Neste caso o Case ssDigitReceived do exemplo abaixo seria movido para o evento OnDigitsReceived. O capítulo Aplicação Passo a Passo explica em detalhes.*

```
'Evento gerado quando a placa detecta um
'Ring. Neste exemplo, atende no primeiro toque
Private Sub VoicerLibX1_OnRingDetected()

    'Mensagem de boas vindas
    VoicerLibX1.PlayFile(1,"c:\boas.sig,"345",0)
```

```
   'Lembre que o programa segue o fluxo normal
após iniciada
   'a reprodução
End Sub


'Evento que ocorre quando a reprodução é
finalizada
'É necessário analisar o motivo
Private Sub VoicerLibX1_OnPlayStop(Port as
Integer,
                           Status As
VoicerLib.TxStopStatus)
Select Case Status
  Case ssNormal:
   'Terminou a mensagem sem opção digitada
   'então manda para a telefonista
   VoicerLibX1.Flash 1,600,1000
   VoicerLibX1.Dial 1,"200",0
    VoicerLibX1.HangUp 1
  Case ssDigitReceived:
   'Recebeu o digito finalizador
   'Verifica qual é e transfere para o ramal
    VoicerLibX1.Flash 1,600 'Executa flash
   if Digits = "3" then
          VoicerLibX1.Dial 1,"220",0   'vendas
   ElseIf Digits = "4" then
          VoicerLibX1.Dial 1,"215",0   'técnica
   ElseIf Digits = "5" then
      VoicerLibX1.Dial 1,"202",0   'fax
   else
      VoicerLibX1.Dial 1,"200",0   'telefonista
   End if
   'Desliga
    VoicerLibX1.HangUp 1
    End Select
End sub
```

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:

```
VoicerLibX1.PlayFile(1,"c:\boas.sig,"345",10)
```

Isto é útil para funções de reprodução de mensagens. Se for passado 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.

4.21 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 e 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 (nnnnnnn,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 milisegundos, 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.

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.A relação das frases e seus respectivos arquivos enconta-se no Apêndice A deste manual

4.22 Status dos Canais

A VoicerLib disponibiliza o método GetPortStatus para que o programador tenha acesso a situação da placa naquele instante.

Exemplo:

```
sts = VoicerLibX1.GetPortStatus(1)
```

O retorno poderá ter os seguintes valores:

- spFlashing - Executando um Flash
- spDialing - Discando
- spNone - Nada
- spWaitingDigits - Esperando Dígitos

4.23 Tratamento de Erros

Basicamente os erros que podem ocorrer com a biblioteca são decorrentes de mal funcionamento do device driver ou mesmo má configuração da placa (Interrupção, etc...).

Além dos valores de retorno do método StartVoicerLib que permitem saber se os serviços foram inicializados corretamente, existe o evento OnErrorDetected, que é ativado quando a placa para de responder ao aplicativo por mais de 15 segundos. Se isto ocorrer, o mais indicado é reiniciar o micro e verificar a configuração da placa.

Também existe a propriedade booleana DriverEnabled, que indica se o driver está ativo (true) ou não (false).

No evento OnErrorDetected é passado o parâmetro ErrorType que pode assumir os seguintes valores:

1. Placa Parou de Interromper – Este erro ocorrerá caso a placa pare de interromper a VoicerLib. Isso pode ser ocasionado principalmente por conflitos de interrupção.
2. Erro de Buffer Ativo – Este erro ocorrerá se a VoicerLib perder algumas interrupções geradas pela placa. Esse problema também está relacionado à conflitos de interrupção ou sobrecarga de processamento na máquina.
3. Erro de Buffer Invalido – O erro 3 é ocasionado por problemas de carga do firmware na placa.

Exemplo:

```
  Private Sub VoicerLibX1_OnErrorDetected(Port
as Integer, ErrorType as Integer)
      If not VoicerLibX1.DriverEnabled then
           MsgBox "Erro na VoicerLib – Codigo "
& ErrorType
      End if
  End Sub
```

4.24 Protegendo sua Aplicação

A Voicer Lib disponibiliza ao programador uma área de memória da placa de 10 caracteres. Esta área é voltada a proteção dos aplicativos desenvolvidos com a Voicer Lib.

Importante: Somente as placas Voicer Phone a partir da versão 1.6 (impresso no circuito impresso) tem esta facilidade.

Nesta área é possível, por exemplo, gravar números de série, senhas, nome do cliente ou qualquer outra coisa que se faça necessária para proteção. Estão disponíveis 10 bytes (caracteres) para gravação.

Gravando na Memória

Para efetuar a gravação, utilize o método WriteSecurityWord, passando como parâmetro a string desejada.

Esta função não permite acesso a endereços específicos nesta memória portanto é necessário passar a string completa. Se você necessitar armazenar vários dados, monte a string com estes dados e em seguida grave-a utilizando o método WriteSecurityWord.

No exemplo a seguir, temos uma rotina que grava o número de série e o nome do cliente na memória.

```
Private Sub GravaMemoria
       Dim ret as integer
       ret =
VoicerLibX1.WriteSecurityWord("000001DF","")
    if ret <> 0 then
              MsgBox "Erro ao gravar"
    end if
End Sub
```

Lendo Dados da Memória

Para ler os dados da memória da placa Voicer Phone, é necessário utilizar o método ReadSecurityWord que retorna a string gravada, com 10 caracteres. Caso haja algum problema de leitura, o método retornará uma string vazia (ou nula).

Exemplo:

```
Private Sub LeMemoria
       Dim sDados as String
       sDados = VoicerLibX1.ReadSecurityWord("")
    if sDados = "" then
            MsgBox "Erro ao ler da memória"
    end if
End Sub
```

Quando o método ReadSecurityWord é chamado, o fluxo de execução do programa só continuará depois que o dados for lido. Esta ação de leitura pode demorar até alguns poucos segundos portanto não utilize o método ReadSecurityWord dentro de looping ou de funções que necessitam de performance.

Os dois métodos tem um parâmetro mostrado nos exemplos acima passados como vazio (""). Este é um parâmetro reservado e sempre deve ser passado o vazio ("").

Parte V

Utilizando um aparelho telefônico

5 Utilizando um aparelho telefônico

Na placa VoicerPhone, existe uma entrada para conectar um aparelho telefônico comum, que pode ser utilizado em alternativa headset.

Quando é utilizado o aparelho, os métodos HangUp, PickUp, PhoneOn, PhoneOff, MicOn e MicOff não são aplicáveis, pois referem-se exclusivamente ao headset, entretanto, é possível gravar e ouvir mensagens e fazer todos os outros tratamentos.

Se o usuário estiver com o telefone fora do gancho e executar o método PickUp, o telefone ficará mudo e o áudio será transferido para o headset.

Quando o telefone é retirado do gancho, o evento OnLineReady é gerado.

Exemplo:

```
Private Sub VoicerLibX1_OnLineReady(Port As
Integer)
    lblStatus.Caption = "Telefone pegou linha"
End Sub
```

Quando o telefone é colocado novamente do gancho, o evento OnLineOff é gerado.

Exemplo:

```
Private Sub VoicerLibX1_OnLineOff(Port As
Integer)
    lblStatus.Caption = "Telefone desligado"
End Sub
```

*A placa **VoicerBox PCI/4** utiliza os eventos **OnLineReady** e **OnLineOff** utilizando-se um cabo especial em que é possível fazer a*

entrada da linha e a saída pelo mesmo conector. Esta característica é voltada principalmente para aplicações de gravador digital de tronco.

Parte VI

Funções Especiais

6 Funções Especiais

6.1 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 apresentadas no capítulo anterior, 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 conferencia 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...
- **Funções de Conferência** - Comunicação entre canais.

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.*

6.2 Funções de Idle

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

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

Até a versão 2.3, estas atividades podiam ser desempenhadas através dos métodos/eventos já conhecidos. Todas estas funções são configuradas através do método IdleSettings (veja-o no guia de referência para acompanhar melhor este tópico), entram em ação a partir da chamada ao método IdleStart e são interrompidas com a chamado do IdleAbort.

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.

Sem está função, o programador tinha que "contar" os toques no evento OnRingDetected e atender através de uma chamada ao evento PickUp.

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 digito terminador. Deixar vazio se não for utilizado.

Para a compreensão destas funções, recomendamos estudo atento do exemplo fornecido na pasta encontrado no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores.

6.3 Funções de Prompt

As funções de Prompt 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 StockSigs 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.*

6.4 Funções de Menu

As funções de menu 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.

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 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.

6.5 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 SetCall*xxxx*.

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 digito 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 deteções erradas de atendimento. Definindo um tempo com este método a supervisão só iniciará a nnn milisegundos 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).*

6.6 Funções de Conferência

Funções que facilitam o gerenciamento entre canais em conferência.

É possível criar "salas" (ou recursos) de conferência contendo de 2 a, no máximo,10 canais.Esse limite é muito dependente dos recursos de CPU. Em testes, foi possível incluir 10 canais em uma mesma conferência com boa qualidade de áudio em um Pentium 4 1.6Ghz com 256MB de memória.

Também é possivel criar várias salas em um mesmo computador. Tomando-se como exemplo um computador com 6 placas VoicerBox PCI/4 instaladas, é possivel criar, por exemplo, 6 salas de 4 canais, 4 salas de 6 canais ou mesmo 12 salas de 2 canais cada.

Exemplo:

É possível criar uma conferência que funciona como uma sala

de bate-papo, sendo que as pessoas que entram nas sala serão os canais que se alocam em determinadas conferências.

Para criar uma nova sala utiliza-se o método *CreateConferenceResource*, passando o numero máximo de pessoas que poderão se comunicar nesta sala. Quando se cria uma conferência ela retorna um valor, o Handle, que será utilizado pelos outros métodos como identificação da sala. Para adicionar uma pessoa por exemplo informa-se esse valor para saber em qual sala ela irá entrar.

O método *ConferenceAddPort* permite adicionar as pessoas as salas, caso a sala esteja cheia, sendo utilizada pelo numero máximo de pessoas determinado, o método retorna o valor -2, podendo então avisar o usuário que a sala já está cheia e que ele tem que escolher outra sala.

Para que uma pessoa esteja em comunicação com as outras da sala é preciso que ela esteja habilitada, através do método *ConferenceEnablePort*. A flexibilidade de habilitar e desabilitar uma pessoa é importante pois, caso a pessoa queira ouvir o menu, os outros canais não irão ouvir também, basta desabilitá-la, através do método *ConferenceDisablePort*, reproduzir o menu que somente ele ouvirá, e após isso habilitá-la novamente. Isso sem precisar removê-la da sala.

O método *ConferenceRemovePort* permite remover as pessoas das salas e o método *DeleteConferenceResource* deleta a sala.

Quando vários canais são somados em uma conferência certamente haverá um efeito de eco, podendo causar a total degeneração do áudio desta "sala". Para diminuir o efeito de eco, deverá ser habilitado o cancelamento através do método **EnableEchoCancel** e para desabilitá-lo, **DisableEchoCancel**. Em uma conferência recomenda-se sempre utilizar o cancelamento de eco. Esse tipo de recurso consome CPU portanto, em situações controladas e com poucas portas (duas), é possivel ter boa qualidade de audio com o cancelamento de eco desabilitado. Por padrão, o cancelamento de eco está

desabilitado na inicialização da VoicerLib.

6.7 Funções para Streaming de Audio

Uma nova característica que está sendo implementada na VoicerLib é a disponibilidade de eventos e métodos que viabilizariam o envio da voz por uma rede local, por exemplo.

Do lado do servidor ou quem gerará as amostras para a rede, o princípio é disponibilizar as amostras de áudio captadas pela placa para aplicação. O envio das amostras para a rede propriamente dito não é responsabilidade da VoicerLib e deverá ser implementado pelo desenvolvedor utilizando bibliotecas com suporte ao WinSock ou mesmo através das funções de baixo nível das APIs do Windows.

Para iniciar o envio das amostras do hardware para a aplicação, deverá ser chamado o método **EnableSampleToApp(porta)**. Este método faz com que o evento OnSampleReceived será gerado de tempos em tempos. Neste evento, o programador deverá extrair as amostras da FIFO e enviá-las pela rede.

Para extrair as amostras da fila que foi criada entre o hardware e a aplicação, utilize o método **GetSamples**.

Do lado do cliente as amostras chegarão através dos eventos TCP/IP (que independem da VoicerLib) e deverão ser tratados de acordo com a vontade do programador. Se na máquina cliente houver uma placa VoicerPhone e o desejo é que o áudio seja ouvido através dela, deverá ser chamado inicialmente o método **EnableSampleToCard** que habilita o envio de amostras da aplicação para a placa.

Em seguida, a cada ***n*** amostras recebidas via TCP/IP, deverá

ser chamado o método **PutSamples** que efetivamente envia a amostra para a placa. Isto tem o mesmo efeito do PlayFile, por exemplo.

Normalmente o programa deverá manter um buffer para receber e armazenar as amostras em memória para então enviá-las pela rede. Lembre-se que a taxa de envio de amostras de e para a placa deverá ser de 1.65 Kb/s pois o padrão é o envio e recebimento apenas no formato GSM.

Parte VII

Distribuindo uma Aplicação

7 Distribuindo uma Aplicação

7.1 Introdução

Visando facilitar a distribuição de todos os arquivos necessários para a execução de uma aplicação construída utilizando a VoicerLib, as placas VoicerPhone e VoicerBox são acompanhadas de um disquete contendo os componentes run-time.

> *No CD de instalação o dri ver para distribuição encontra-se na pasta* **Driver\Instalação**

Para placas VoicerBox PCI/4 e VoicerPhoe PC/1, também é necessário criar um disquete com o conteúdo da pasta Driver\InfParaPCI. Esta pasta contém os arquivos necessários para instalação do hardware no momento que o Windows detectad a(s) nova(s) placa(s). Após detectar, o Windows pede a localização do Driver. Neste caso informe o disquete ou o caminho onde se encontra os arquivos da pasta Driver\InfParaPCI do CD-ROM.

Além do procedimento acima, é necessário instalar os outros arquivos da biblioteca através do programa setup.exe da pasta Driver\Instalação do CD-ROM. Ele pode ser executado diretamente do CD ou através de disquete ou drive de rede.

Caso o desenvolvedor queira agregar a instalação do driver ao seu próprio set de instalação, o programa setup.exe pode ser executado em modo silencioso, ou seja, sem as janelas de diálogo iniciais. Isto pode ser feito passando o parâmetro /q . Ex.:

Setup.exe /q

Desta maneira o desenvolvedor pode mandar executar o setup

do Digivoice Driver após o término da sua própria instalação.

7.2 Preparando o Ambiente

Placa PCI/4 e PCI/1

Não é necessário nenhum procedimento de configuração para as placas PCI pois a interrupção e endereço de I/O são atribuídos automaticamente pela BIOS do PC.

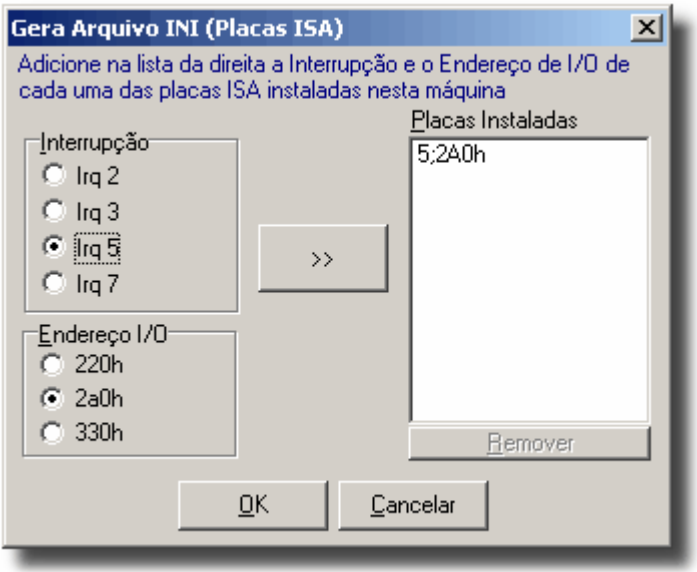
Placa ISA

No caso da placa ISA, é necessário configurar o Windows com a quantidade de placas, interrupções e endereços de I/O utilizados. Isto é possível através do utilitário GERAINI.EXE instalado no diretório do Windows. Também é possível o desenvolvedor criar o arquivo de configuração por conta própria, pois segue o padrão dos arquivos tipo INI do Windows

Lembre-se de verificar se não há conflitos de interrupção com outro hardware instalado.

GeraIni.EXE

Você pode executá-lo através do menu Iniciar à Executar...



- Para adicionar uma placa selecione a Interrupção e o endereço de I/O desejado e clique no botão ">>" .
- Para remover, selecione a placa desejada na lista de Placas Instaladas e clique no botão remover.
- Para gravar a configuração, clique no botão OK.

Criando o próprio arquivo de configuração

O arquivo de configuração padrão das placas VoicerPhone ISA é o voicerlib.ini que fica no diretório do Windows. Ele segue o padrão dos arquivos INI do Windows e tem o seguinte formato:

```
[ISA]
NumCards=2      ß Número de placas instaladas no
micro

[ISA1]            ß Cada placa deve ter o nome
ISAn
irq=5             ß Número da Interrupção desta
```

```
placa
io=672          ß Endereço de I/O (em decimal)

[ISA2]          ß Informações da segunda placa
se houver
irq=5
io=816
```

Os endereços de I/O devem ser representados através do seu valor decimal. Neste manual e na propria placa eles estao representados em hexadecimal portanto é necessário convertê-los. Veja abaixo os valores que devem ser utilizados:

Hexadecimal	Decimal
220H	544
2a0H	672
330H	816

O Windows disponibiliza as funções GetPrivateProfileString e WritePrivateProfileString que manipulam este tipo de arquivo com facilidade.

Para maiores detalhes consulte o Guia de Referência de sua linguagem de programação.

Os uuários do Borland Delphi e C++Builder tem a classe TIniFile que facilita ainda mais a manipulação deste tipo de arquivo.

Parte VII

Aplicação Exemplo

8 Aplicação Exemplo

8.1 Apresentação

Este capítulo apresenta uma aplicação de atendimento automático comentada passo a passo. Com ele, o programador poderá facilmente entender os procedimentos corretos de programação utilizando a VoicerLib.

A aplicação foi desenvolvida utilizando-se o Borland Delphi, portanto o código apresentado aqui é Pascal. Os programadores de Visual Basic e outras linguagens não terão dificuldades em entender a lógica apresentada pois tudo é comentado em detalhes.

> *O mesmo programa escrito em Visual Basic pode ser encontrado no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores.*

O exemplo é voltado a explicar a VoicerLib, portanto alguns detalhes que seriam imprescindíveis em um programa comercial serão omitidos ou mesmo utilizaremos técnicas hardcode (dados cravados no código) para detalhes de configuração.

Aqui optamos por utilizar os métodos de "baixo nível" para firmar os conceitos da VoicerLib e passar técnicas de programação para este tipo de aplicação que difere bastante das aplicações procedurais tradicionais. Um exercício interessante é fazer a mesma aplicação utilizando-se das funções especiais apresentadas no capítulo anterior. Com elas o programa ficará bem menor e mais simples!

O programa de atendimento executa as seguintes funções:

1. Espera chamada

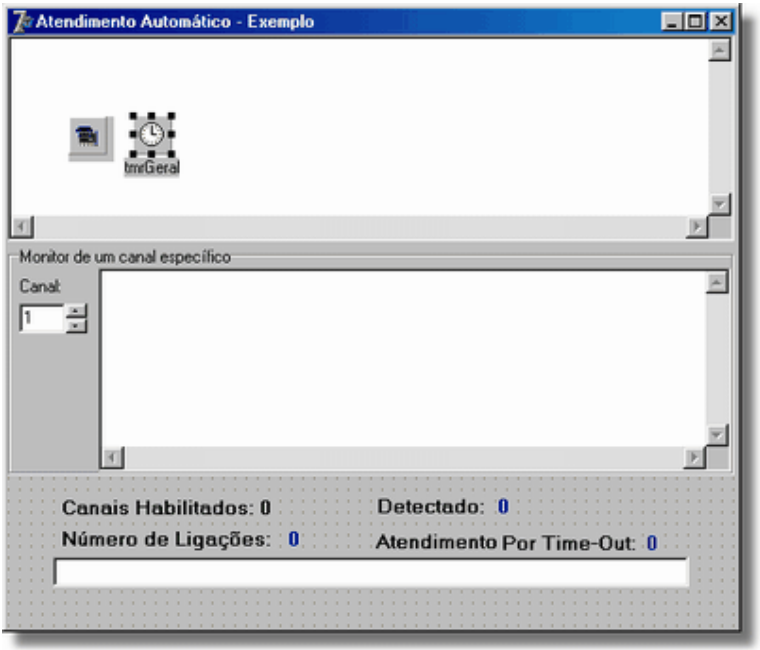
2. Executa frase de atendimento com menu
3. Espera digitação das opções de menu ou do ramal
4. Abre opções de sub-menu - se houver
5. Transfere para o ramal desejado
6. Efetura supervisão de ramal, verificando se está ocupado ou não atende.
7. Transfere para os ramais de fuga com supervisão
8. Desliga.

Para otimizar o código aqui no manual, em alguns pontos da listagem aparecerá o símbolo ***(.....)*** que indicará que neste ponto existe algum código que não é importante para o escopo da explicação naquele momento. Qualquer dúvida no posicionamento correto do código, o exemplo pode ser encontrado no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores.

O Programa

O programa apresentado a seguir pode ser encontrado no site da DigiVoice www.digivoice.com.br, na seção de desenvolvedores.

A interface de um programa como este é extremamente simples;



Também é utilizado um objeto Timer para funções de timeout.

> *Repare que não foi utilizado nenhum elemento exclusivo do Delphi, o que torna a transcrição do programa para outra linguagem extremamente simples. Consulte periodicamente o site **www.digivoice.com.br/** na seção de desenvolvedores para verificar se o exemplo já foi traduzido para outras linguagens.*

O componente VoicerLibX200 é colocado no form principal e é dado o nome **voice1**.

8.2 Atendimento de uma ligação

Neste exemplo, todas as ligações sempre serão atendidas no primeiro toque. Neste caso, o primeiro evento que deve ser codificado é o OnRing que é gerado sempre que chegar um tom de ring pela linha/ramal.

> *No código abaixo utilizamos o **case .. of** do Pascal pois é muito mais legível que uma seqüência de if aninhados (ifs dentro de ifs!) principalmente quando houverem muitos estados a tratar. O Visual Basic tem o equivalente **Select Case** e o C/C++ tem o **switch**.*
> *Caso sua linguagem não tenha uma estrutura equivalente, só caberá utilizar o if...*

```
procedure TForm1.voice1RingDetected(Sender:
TObject; Port:

   Smallint);
begin
  case Estado[Port] of
     NADA:    //Só entra aqui se estiver no
estado de espera
     begin
        //Limpa o buffer de digitos da porta
corrente
        voice1.ClearDigits(Port);
        //comentario na janela de status
        InsereDado(Port,'Atendeu');
        //Atende a ligação
        voice1.PickUp(Port,4000);
        //indica para esperar DTMF
        voice1.SetDetectionType(Port,dtDTMF);
     end;
  end;
end;
```

Repare que somente o Estado **NADA** é tratado, ou seja, se chegar um ring e a variável Estado estiver com outro valor, nada

será feito.

Dentro do case do Estado **NADA**, temos a chamada ao método ClearDigits que limpa o buffer interno de dígitos. Este buffer que conterá os números digitados pelo usuário que ligou para a placa.

Em seguida é chamado o método PickUp que efetivamente atende a ligação. É passado como parâmetros a porta e a pausa após atendimento de 4 segundos (4000). Esta pausa é necessária para prever situações onde o PABX demora algum tempo para comutar a linha com o ramal, ou mesmo nos casos onde existe um bloqueador de chamadas à cobrar (como é o caso do atendimento da Digivoice). A continuação do atendimento será efetuada no evento OnAfterPickUp que é gerado após os 4 segundos passados no método PickUp.

Também chamamos aqui o método **SetDetectionType** passando a porta e o parâmetro dtDTMF que é necessário para que as placas entendam os tons gerados pelo telefone do cliente. Este método é chamado aqui mas também poderia ser chamado no evento OnAfterPickUp.

8.3 Conclusão

Este exemplo prevê uma série de situações comuns a sistemas de auto-atendimento, URA, etc....

Ele poderá ser utilizada e modificada como ponto de partida para suas próprias aplicações.

> ***Atenção:*** *como a aplicação apresentada foi criada para fins didáticos, a Digivoice não se responsabiliza pelo seu funcionamento em ambientes de produção. Caberá aos desenvolvedores adequar e aprimorar seu funcionamento para atender suas necessidades específicas.*

8.4 Definindo Constantes para a Aplicação

Uma boa prática para aplicações de telefonica em tempo real é a utilização de estados. Nada mais são que variáveis nas quais são atribuídos valores diferenciados para cada situação possível (falando, atendendo, esperando, etc...).
Por ter métodos assíncronos, a VoicerLib exige do programador uma certa disciplina para manter o código legível e de fácil manutenção. O uso de estados ajuda neste aspecto.
Para aumentar ainda mais a legibilidade de código, definiremos constantes para cada estado possível da aplicação, ao invés de utilizar o número literal na hora de atribuir valores aos estados. Observe o código:

```
Estado := 6;    //Ilegível -> Dificil manutenção
(....)
Estado := TRANSFERENCIA;   //Facilidade de
manutenção
```

Isto não é um procedimento obrigatório mas, todos hão de convir que é muito mais fácil trabalhar com palavras do que com valores.
No Delphi, as constantes são definidas em qualquer janela ou módulo, na área const. Todas as linguagens conhecidadas tem seu equivalente.

No caso do nosso exemplo, isto será feito no form principal, pouco antes da palavra-chave **implementation**.

```
const

   //AS CONTANTES SERÃO DEFINIDAS AQUI
   //Você deve digitar a palavra const também

var
  Form1: TForm1;


implementation
```

```
{$R *.DFM}
```

Obviamente você pode ir definindo as constantes a medida que os estados vão aparecendo. Aqui já temos todos os estados que serão necessários na execução do programa:

```
const

  // Estados de Atendimento
  NADA = 1;
  BOASVINDAS = 3;
  ESPERA_DIGITOS_MENU = 4;
  ESPERA_DIGITOS_RAMAL = 5;
  TRANSFERENCIA = 6;
  TENTEMAISTARDE = 7;
  SUBMENU = 8;
  ESPERA_SUBMENU = 9;
```

Além dos estados principais do fluxo de atendimento do programa, teremos também alguns estados que serão utilizados durante a transferência e supervisão das ligações: os estados de ramais e de flash:

```
const

  // Estados de Atendimento
  NADA = 1;
  BOASVINDAS = 3;
  ESPERA_DIGITOS_MENU = 4;
  ESPERA_DIGITOS_RAMAL = 5;
  TRANSFERENCIA = 6;
  TENTEMAISTARDE = 7;
  SUBMENU = 8;
  ESPERA_SUBMENU = 9;

  //Estado de Ramais
  LIVRE  = 0;
  OCUPADO = 1;
  NAOATENDE = 2;

  //Estados de Flash
  DISCA = 0;
  RETOMA = 1;
```

Por último temos algumas constantes que serão utilizadas no exemplo para simplificar as coisas. São configurações de funcionamento da aplicação como ramal de fuga e toques para retomada.

Em uma aplicação real estes valores ficam melhor em uma arquivo de configuração e lidos na inicialização da aplicação:

```
const

 (...colocar abaixo dos estado de flash.....)

  //Ramais de fuga (telefonista, portaria,
etc...)
  RAMAL_FUGA1 = '220';
  RAMAL_FUGA2 = '222';

  //Toque para retomada da ligacao em caso do
ramal
  //não atender
   NUMTOQUESRETOMADA = 5;
```

8.5 Finalizando a VoicerLib

A finalização da biblioteca deve ser feita, preferencialmente, no último evento chamado antes do término da aplicação. Aqui faremos isso no evento OnClose.

```
//Aqui testamos se a placa está inicializada
//antes de finalizá-la
if voice1.DriverEnabled then
     voice1.ShutdownVoicerLib;
```

8.6 Inicializando a VoicerLib

O método StartVoicerLib deve ser utilizado para inicializar a(s) placa(s) que serão utilizadas. Aqui, como é um programa de apenas uma janela, a biblioteca é inicializada quando ela é mostrada pela primeira vez, no evento OnShow.

```
  //Indica se utiliza placa ISA (ctVPISA) ou PCI
(ctVPPCI ou ctVBPCI)
  voice1.CardType := ctVPISA;

  //Inicializa e já testa se deu tudo certo
  if voice1.StartVoicerLib <> 99 then
     ShowMessage('Erro de Inicialização');
```

Antes da inicialização, a propriedade CardType é preenchida com a constante ctVPISA que indica que serão utilizadas placas VoicerPhone ISA. Se fossem placas VoicerBox PCI/4, a constante deveria ser ctVBPCI e se for a placa Voicer Phone PCI/1, utiliza-se a constante ctVPPCI

A função StartVoicerLib retorna o set status. Qualquer valor diferente de 99 significa erro. Consulte o guia de referência para saber os erros possíveis e seus respectivos códigos.

8.7 Inicializando o Estado da Aplicação

No mesmo lugar onde iniciamos o driver da placa (OnShow) é necessário inicializar a variável Estado para o correto funcionamento.

```
//Inicializa e já testa se deu tudo certo
if voice1.StartVoicerLib <> 99 then
   ShowMessage('Erro de Inicialização');

//Inicialização do Estado Inicial
for i:=1 to 20 do
```

```
Estado[i] := NADA;
```

O estado NADA indica que o canal não está fazendo nada, apenas esperando uma ligação chegar.

8.8 Iniciando o menu de atendimento

A frase de atendimento é reproduzida a partir do evento OnAfterPickUp . No exemplo utilizamos o enfileiramento de mensagens para poder reproduzir a seqüência de mensagens:

Digivoice → Bom Dia → Disque o número do ramal ....

Neste caso, utilizamos 3 mensagens ao invés de uma simplesmente para poder implementar a facilidade do atendimento falar bom dia, boa tarde ou boa noite, conforme a hora da ligação. Também neste exemplo utilizamos frases de menu diferenciadas para o período comercial e não comercial.

Para isso utilizaremos o conjunto de métodos ***PlayListxxxxxxxx*** que disponibilizam uma lista de mensagens independente por canal. Isto é importante porque cada canal das placas estarão em um ponto diferente de atendimento durante uma operação real.

No evento OnAfterPickUp teremos:

```
procedure TForm1.voice1AfterPickUp(Sender:
TObject; Port:
                                                    Sma
llint);
begin
//muda o estado
Estado[Port] := BOASVINDAS;

//limpa lista de frases do canal
voice1.PlayListClear(Port);
```

```
//Adiciona a primeira frase
voice1.PlayListAdd(Port,ptFile,'empresa.sig','',0
);

//bom dia boa tarde boa noite
if ( (Time >= StrToTime('08:00:00')) and
    (Time <=  StrToTime('12:00:00'))) then
          voice1.PlayListAdd(Port,ptFile,'bomdia.
sig','',0)
else
   if ( (Time > StrToTime('12:00:00')) and
      (Time <= StrToTime('18:00:00'))) then
      voice1.PlayListAdd(Port,ptFile,'boatarde.si
g','',0)
   else
      voice1.PlayListAdd(Port,ptFile,'boanoite.si
g','',0);

//menu principal de acordo com o periodo
if ( (Time >= StrToTime('07:59:00')) and
                  (Time <=
StrToTime('18:00:00'))) then
   voice1.PlayListAdd(Port,ptFile,'menu_dia.sig',
'',0)
else
   voice1.PlayListAdd(Port,ptFile,'menu_noite.sig
','',0);

//inicia a reproducao das 3 mensagens em
sequencia
ret := voice1.PlayList(Port, '23456');

end;
```

No código acima, primeiro setamos a variável Estado com a constante BOASVINDAS, indicando que agora este canal encontra-se neste estado. Utilizamos a variável Port como índice do vetor pois ela é passada como parâmetro do evento OnAfterPickUp indicando a porta atual.

Em seguida apagamos todo o conteúdo da lista de mensagens. Isto é necessário para que somente as mensagens seguintes serão reproduzidas, não ficando nenhuma do atendimento

anterior.

A primeira seqüência de ifs serve para o programa saber se fala "bom dia", "boa tarde" ou "boa noite", dependendo do hora do relógio do micro.

O segundo if (menu principal) serve para o programa decidir se fala uma mensagem diurna ou uma noturna.

Por fim, é chamado o método PlayList que começa as mensagem da lista,. Foi configurado que ela pode ser interrompida pelos dígitos 23456 (terceiro parâmetro).

Como o PlayList é assíncrono (assim como todos os outros métodos) o fluxo de execução continua após o início da mensagem. Desta maneira, o próximo tratamento deve ser feito quando terminar a mensagem, ou seja, no evento OnPlayStop ou o evento OnDigitReceived no caso da mensagem ser interrompida por dígitos.

8.9 Janela de Monitoramento

Apesar de não fazer parte propriamente do fluxo de atendimento, uma área onde se monitora o que está acontecendo é muito útil.

Aqui utilizamos um objeto TMemo, que é uma área de texto com possibilidade de inserir várias linhas.

> *O VisualBasic utiliza o TEditBox com a propriedade MultiLine igual a True. Você também pode utilizar um ListBox.*

Criaremos uma procedure (sub-rotina) que simplesmente adiciona uma linha neste TMemo, junto com a hora e o canal corrente. Esta rotina chama-se InsereDado:

```
//Concatena hora e insere no TMemo
procedure TForm1.InsereDado(nPorta:integer;sDado:
string);
begin
  memStatus.Lines.Add(TimeToStr(Time) +
                      ': <' +
                      IntToStr(nPorta)+'>
'+sDado);

end;
```

Desta maneira, sempre que quisermos adicionar um comentário de status, bastará chamar a procedure InsereDado assim:

```
InsereDado(Porta,'Seu comentário');
```

8.10 Transferência e Supervisão

Quando o usuário seleciona uma opção de menu ou disca um número de ramal, o atendedor tenta transferir para este ramal. Caso dê ocupado ou não atender ele transfere para um ramal de fuga e se ainda assim não conseguir, transfere para outro ramal de fuga. Este ciclo é idêntico para o ramal desejado e para os ramais de fuga independente da quantidade.

Por isso, utilizamos a técnica de criar uma lista para os ramais que serão discados. Obviamente se o ramal desejado for atendido, a ligação é entregue e o ciclo termina. Esta lista é do tipo TStringList e se chama **lstRamal**.

Ela deve ser declarada como global, criada no evento **OnShow** e destruída no evento **OnClose**.

A supervisão consiste no sistema monitorar se um ramal está sendo chamado sem sucesso, der ocupado ou ainda ser

atendido. No caso de ramal ocupado, a VoicerLib gera o evento **OnBusyDetected**. Já o tom de chamada gera o evento **OnCalling** e o atendimento deve ser tratado no evento **OnAnswerDetected**.

No sistema de atendimento da Digivoice, como em tantos outros, para transferir a ligação é necessário dar um flash e em seguida discar o ramal desejado. O início da discagem foi implementado no evento OnPlayStop porque sempre antes de transferir uma ligação o sistema falará "*aguarde um instante...*" como pode ser observado na listagem anterior (PlayFile).

A mensagem "*aguarde um instante...*" não admite nenhum tipo de interrupção por dígito, por isso somente o caso *ssNormal* do evento **OnPlayStop** precisa ser considerado:

```
procedure TForm1.voice1PlayStop(Sender: TObject;
Port,
     StopStatus: Smallint);
begin

  case StopStatus of
    ssNormal:  //MENSAGEM FOI ATE O FIM
    begin
       case Estado[Port] of
         (........)
       TRANSFERENCIA:
       begin
         if (lstRamal.Count > 0) then  //ainda
tem ramais
         begin
           if nEstadoRamal[Port] <> LIVRE then
           begin
             //verifica se
             nEstadoRamal[Port] := LIVRE;
             voice1.PlayFile(Port,'sisaguar.sig',
'');
           end
           else
           Begin
             //prepara discagem
             InsereDado(Port,'Executando
```

```
Flash...');
             nEstadoFlash[Port] := DISCA;
             voice1.Flash(Port,700,1500);
           end;
         end
         else
         begin
           //Acabaram os ramais da lista
          //Fala "tente mais tarde"
           Estado[Port] := TENTEMAISTARDE;
           voice1.PlayFile(Port,'MAISTARD.SIG',''
);
         end;
       end;
       TENTEMAISTARDE:
       begin
         Sleep(100);
         voice1.HangUp(Port);
         Estado[Port] := NADA;
         InsereDado(Port,'Não ouve atendimento.
Desligado!');
         InsereDado(Port,'Esperando nova
ligação...');
       end;
     end;
     ssDigitReceived:
     begin
       (........)
    end;   //submenu
  end;
   (........)
```

Repare que a lógica de tratamento da lista é idêntica à lista de frases de boas vindas no início do atendimento. Cada vez que o sistema entrar neste bloco, ele terá discado para o ramal, supervisionado e detectado que o ramal não atendeu o estava ocupado. Isto se repetirá até que a lista lstRamal esteja vazia. Quando isso acontecer, o sistema falará "tente mais tarde", sairá da rotina e voltará a ela no tratamento do estado TENTEMAISTARDE que desligará e colocará no estado NADA, pronto para esperar uma nova ligação.

Mais uma vez reforçamos esta característica assíncrona que a VoicerLib tem. Repare que dentro do evento OnPlayStop é executado o PlayFile. O fluxo continua e sai do OnPlayStop. Quando a mensagem termina o evento é chamado novamente. Não se trata de recursividade.

O primeiro bloco da rotina anterior inicia a transferência através do comando Flash. Antes é necessário atribuir a variável **nEstadoFlash** com o valor DISCA para que possa ser diferenciado a situação de flash para discar e flash para retomar a ligação (em caso de ocupado ou não atender). Como o flash é um comando lento, sua conclusão dar-se-á no evento **OnAfterFlash**:

```
procedure TForm1.voice1AfterFlash(Sender:
TObject; Port:

   Smallint);
begin
   case nEstadoFlash[Port] of
     DISCA:
     begin
       InsereDado(Port,'Discando para o Ramal ' +
                                        lstRamal.S
   trings[0]);
       voice1.Dial(Port,
   lstRamal.Strings[0],1000);
       nChamando[Port] := 1;
     end;
   (......)
```

A variável *nChamando[Port]* é inicializada em **1** para contar os toques de chamada. No nosso sistema as ligações são retomadas após **5** toques.

O início da supervisão se dará após a discagem no evento OnAfterDial:

```
procedure TForm1.voice1AfterDial(Sender: TObject;
Port:
```

```
                                                              S
mallint);
begin
  case Estado[Port] of
   TRANSFERENCIA:
   begin
      InsereDado(Port,'Iniciando Supervisão...');
      voice1.EnableCallProgress(Port);
      voice1.EnableAnswerDetection(Port);
      //12 segundos ate o primeiro toque
      nContaTimeOutAtende[Port] := 12;
   end;
 end;
end;
```

A supervisão é iniciada com o **EnableCallProgress** que monitora os tons de "chamando" e ocupado e o **EnableAnswerDetection** que monitora o atendimento.

> *Com a placa VoicerPhone utilizando um headset é necessário desligar o microfone antes de habilitar as supervisões de linha*

Neste momento o sistema deverá estar preparado para 3 situações: atendimento, ocupado e não atende.

O atendimento poderá ser tratado de duas maneiras: a primeira é através da detecção propriamente dita que gera o evento **OnAnswerDetected**:

```
procedure TForm1.voice1AnswerDetected(Sender:
TObject; Port:

Smallint);
begin
  case Estado[Port] of
   TRANSFERENCIA:
   begin
      voice1.DisableAnswerDetection(Port);
      voice1.DisableCallProgress(Port);
      InsereDado(Port,'Ligação Entregue.
   Esperando nova...');
```

```
        voice1.HangUp(Port);
        Estado[Port] := NADA;
     end;
   end;
end;
```

A segunda forma de detecção de atendimento não envolve comandos da placa. Em algumas ocasiões a placa poderá não detectar o atendimento através do método acima. Isso é devido a condições de linha, etc... Nestes casos o sistema jamais entenderia o atendimento e consequentemente não transferiria a ligação, causando um comportamento indesejado.

Para cobrir 100% das vezes é necessário criar um timeout para atendimento através de um componente do tipo Timer, colado no form. Este timer seria utilizado da seguinte maneira.

```
procedure TForm1.tmrGeralTimer(Sender: TObject);
var
  Port: integer;
begin
  for Port := 1 to 20 do
   begin
    case Estado[Port] of
    TRANSFERENCIA:
    begin
       Dec(nContaTimeOutAtende[Port]);
       if nContaTimeOutAtende[Port] = 0 then
       begin
         //timeout, significa que atendeu
         InsereDado(Port,'Timeout de
   atendimento');
         //chama a rotina que trata o atendimento
         voice1AnswerDetected(Sender,Port);
       end;
    end;
   end;
  end;
end;
```

A partir do momento que o sistema disca, considera-se que, se não houver sinal de chamada ou ocupado por um determinado

tempo, houve um atendimento. Por isso que no evento OnAfterDial a variável nContaTimeOutAtende é inicializada com 12, indicando que o sistema esperará até 12 segundos para o primeiro sinal de chamada (OnCalling). Após o primeiro, o timeout cairá para 8 segundos entre cada sinal de chamada. Para maiores detalhes observe também a listagem do evento OnCalling logo a seguir.

Ao ser detectado o atendimento, é importante desabilitar todas as supervisões para não interferir no funcionamento do próximo atendimento. Em seguida desliga-se e muda a variável Estado para NADA.

A cada sinal de "chamando" o evento OnCalling é chamado:

```
procedure TForm1.voice1Calling(Sender: TObject;
Port:

Smallint);
begin
  case Estado[Port] of
   TRANSFERENCIA:
   begin
      InsereDado(Port,IntToStr(nChamando[Port])+
                                                'o.
   Toque...');
      //8 segundos ate os proximos toques
      nContaTimeOutAtende[Port] := 8;
      if nChamando[Port] > NUMTOQUESRETOMADA then
      begin
          //deu ocupado
          voice1.DisableAnswerDetection(Port);
          Sleep(10);
          voice1.DisableCallProgress(Port);
          InsereDado(Port,'Ramal não atende...');
          nEstadoRamal[Port] := NAOATENDE;
          nEstadoFlash[Port] := RETOMA;
          voice1.Flash(Port,700,1500);
      end
     else
         Inc(nChamando[Port]);
   end;
   end;
```

```
end;
```

A variável **nChamando[Port]** é incrementada a cada tom recebido. Se ela ultrapassar **NUMTOQUESRETOMADA** o programa prepara a retomada da ligação. Seta a variável nEstadoRamal para NAOATENDE e a nEstadoFlash para RETOMA. A primeira é necessária para falar *"ramal não atende"* e a segunda para o tratamento específico de retomada no evento **OnAfterFlash** (listada mais adiante):

No caso de dar ocupado, o evento **OnBusyDetected** é gerado:

```
procedure TForm1.voice1BusyDetected(Sender:
TObject; Port:

Smallint);
begin
   case Estado[Port] of
    TRANSFERENCIA:
    begin
      //deu ocupado na transferencia
      voice1.DisableAnswerDetection(Port);
      Sleep(10);
      voice1.DisableCallProgress(Port);
      InsereDado(Port,'Ocupado na
   Supervisão...');
      nEstadoRamal[Port] := OCUPADO;
      nEstadoFlash[Port] := RETOMA;
      voice1.Flash(Port,700,1500);
    end;
    else
    begin     //qualquer outro caso desliga
      Sleep(100);
      voice1.HangUp(Port);
      Estado[Port] := NADA;
      InsereDado(Port,'Cliente Desligou.');
      InsereDado(Port,'Esperando nova
   ligação...');
     end;
    end;
end;
```

Você tem o caso de TRANSFERENCIA e todos os outros no

bloco else. Nos casos diferentes de transferência, ao receber um ocupado o sistema simplesmente desliga a ligação e volta o Estado para NADA, esperando novas chamadas.

No caso de TRANSFERENCIA o sistema desabilita as supervisões, seta a variável nEstadoRamal para OCUPADO e a nEstadoFlash para RETOMA. A primeira é necessária para falar "ramal ocupado" e a segunda para o tratamento específico de retomada no evento OnAfterFlash:

```
procedure TForm1.voice1AfterFlash(Sender:
TObject; Port:

   Smallint);
begin
   case nEstadoFlash[Port] of
      (......)
     RETOMA:
     begin
       InsereDado(Port,'Retomando Ligação...');
       try
          lstRamal.Delete(0);
       except
       end;
       //fala ramal ocupado ou ramal nao atende
       if nEstadoRamal[Port] = OCUPADO then
       begin
            voice1.PLayFile(Port,'SISROCUP.SIG','
   ')
       end
       else
       begin
             voice1.PLayFile(Port,'SISRNAOA.SIG',
   '');
       end;
    end;
   (......)
```

Ao retomar a ligação, depois do flash, é necessário excluir o primeiro ramal da lista (para que o próximo se torne o ramal corrente) e falar "ramal ocupado" ou "ramal não atende". A discagem para o próximo ramal (se houver) será iniciada

novamente no evento OnPlayStop (consulte a listagem da página 70). Este ciclo se repetirá até que não haja mais ramais.

8.11 Tratamento de Entrada de Dados

O evento OnPlayStop é chamado sempre quando uma mensagem termina, portanto este evento será chamado diversas vezes durante o atendimento. Ele junto com o evento OnDigitsReceived são os mais complexos do exemplo e merecem maior atenção.

Nesta primeira parte do tratamento do OnPlayStop, contemplaremos apenas a execução em seqüência da mensagem de boas vindas.

O evento OnPlayStop passa o status como parâmetro através da variável StopStatus. No nosso atendimento é necessário tratar duas situações.: quando a mensagem termina normalmente ou quando ela é interrompida por causa do recebimento de um dígito. No primeiro caso a variável StopStatus recebe o valor ssNormal. Nesta situação o sistema ficará em silêncio por 5 segundos esperando a digitação do ramal ou opção de menu pelo usuário.

```
procedure TForm1.voice1PlayStop(Sender: TObject;
Port,
                                          StopStatu
s: Smallint);
Begin
     case StopStatus of
        ssNormal:  //MENSAGEM FOI ATE O FIM
        begin
           case Estado[Port] of
              BOASVINDAS:
              begin
                  //aqui espera 1 digito que pode
   ser a opção
                  //de menu ou o primeiro digito
```

```
      do ramal
                        InsereDado(Port,'Esperando
      Digitação');
                        Estado[Port] :=
      ESPERA_DIGITOS_MENU;
                        voice1.GetDigits(Port,1,'',5000
      ,5000);
                        //habilita espera de ocupado
                        voice1.EnableCallProgress(port)
      ;
                        //Habilita detecção de pulso
                        voice1.EnablePulseDetection(Por
      t);
                  end;
            end;
        end;
    end;
```

Este if acima trata o caso de as mensagens de boas vindas terem sido reproduzidas sem a interrupção do usuário. O fluxo de atendimento pede que após a mensagem o sistema fique 5 segundos em silêncio esperando a digitação do usuário. Isto é feito apenas com a chamada a função GetDigits.

No código apresentado o GetDigits indica que a VoicerLib esperará por 1 digito que poderá ser a opção de menu ou o primeiro dígito do ramal. O tratamento desta função GetDigits será feito no OnDigitsReceived, explicado mais adiante.

Repare que o estado é alterado para ESPERA_DIGITOS_MENU para que o evento OnDigitsReceived saiba qual a situação a tratar.

Ainda é preciso prever a situação do usuário teclar algo sobre a mensagem. Até a versão 2.0 isto deveria ser feito no evento OnPlayStop.

A partir da versão 2.1, o evento OnDigitsReceived também é gerado quando um dígito é detectado durante a reprodução de mensagem sendo que a variável Status recebe o valor edDigitOverMessage para indicar a situação. Isto faz com que

tanto o tratamento de dígito sobre a mensagem como o gerado a partir do método GetDigits sejam tratados em um único lugar, no caso, o evento OnDigitReceived.

Agora apresentaremos o caso de dígito sobre mensagem. Se for o primeiro dígito da numeração de ramais (dois no nosso exemplo) o sistema deverá ficar esperando os proximos 2 dígitos. Se for outro, deve ser tratado como opção de menu.

```
procedure TForm1.voice1DigitsReceived(Sender:
TObject; Port,
  Status: Smallint);
var
  ret: integer;
begin
  case Status of
     EdMaxDigits,
     edDigitOverMessage:  //msg interrompida por
digito
     begin
       case Estado[Port] of
          ESPERA_DIGITOS_MENU,
          BOASVINDAS:
          begin
              InsereDado(Port,
                       'Tratamento de Digito após
mensagem');
              TrataPrimeiroDigito(Port);  {<-
IMPORTANTE}
          end;
        end;
     end;
  end;

end;
```

O case mais externo (Status) testa se o valor é EdMaxDigits ou edDigitOverMessage. O primeiro indicará que a mensagem de boas vindas foi executada até o fim e o método GetDigits foi acionado, conforme mostrado no código da página anterior. Já o segundo indica que a mensagem foi interrompida por dígito. Como o tratamento é o mesmo para os dois casos, o fato de

separarmos as constantes por virgula indica um OU outro.

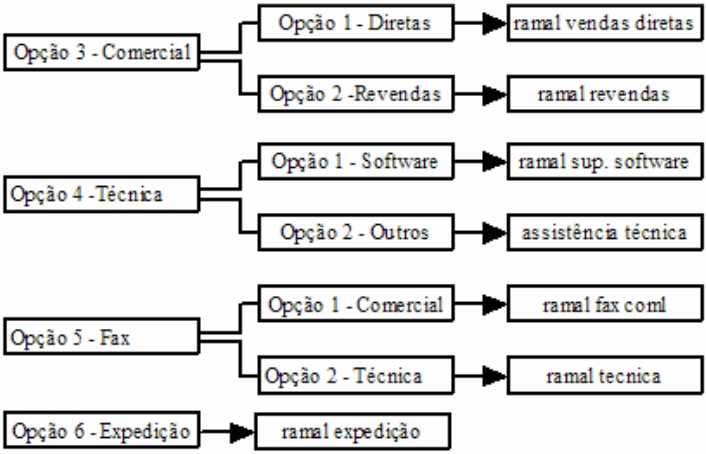
A função TrataPrimeiroDigito é uma subrotina que verificará se é ramal ou opção de menu. Este código foi colocado em uma subrotina para deixar o programa mais organizado. Eis o código da rotina TrataPrimeiroDigito:

```
procedure TForm1.TrataPrimeiroDigito(Port:
smallint);
begin
   if Copy(voice1.ReadDigits(Port),1,1) = '2'
then
   begin
      InsereDado(Port,'Esperando resto do
   ramal');
      Estado[Port] := ESPERA_DIGITOS_RAMAL;
      voice1.GetDigits(1,3,'',5000,5000);
   end
   else
   begin
      nOpMenu[Port] :=
                 StrToInt(Copy(voice1.ReadDigits(
   Port),1,1));

      if rcMenu[nOpMenu[Port]].sRamalMenu <> ''
   then
      begin
         //disca direto
         //adiciona o ramal
         lstRamal.Add(rcMenu[nOpMenu[Port]].sRama
   lMenu);
         //adiciona fuga
         lstRamal.Add(RAMAL_FUGA1);
         lstRamal.Add(RAMAL_FUGA2);
         Estado[Port] := TRANSFERENCIA;
         nEstadoRamal[Port] := LIVRE;
         //aguarde um instante...
         voice1.PlayFile(Port,'sisaguar.sig','');
      end
      else
         if rcMenu[nOpMenu[Port]].sFrase <> ''
   then
         begin
            voice1.ClearDigits(Port);
```

```
            Estado[Port] := SUBMENU;
            //fala submenu
            voice1.PlayFile(Port,
                        rcMenu[nOpMenu[Port]].sFr
ase,
                        rcMenu[nOpMenu[Port]].sOp
coesSubMenu);

         end
         else
         begin
            //estrutura errada, sem ramal nem
submenu
            //então vai parafuga
  (.....)
         end;
      end;
end;
```

Aqui é importante entender um detalhe de implementação do aplicativo exemplo. Como ele prevê a possibilidade de menus e sub-menus era necessário criar uma estrutura de dados que simplificasse a codificação desta necessidade.

No exemplo, se o usuario escolher a opção de menu 3, por exemplo, ele irá para um sub-menu com duas opções. Cada uma destas duas opções transfere para um ramal diferente. Já a opção 6, transfere para um ramal diretamente, sem sub-menu. O esquema abaixo ilustra a estrutra de menu e sub-menu do nosso exemplo:



No programa, foi criado uma estrutura do tipo record da seguinte maneira:

```
  TrcMenu = record
     sRamalMenu: string;   //se for vazio é pq
tem submenu
     sFrase: string;      //preenhida no caso
de submenu
     sOpcoesSubMenu: string;
  end;
```

Em seguida foram criadas duas variáveis globais, uma do tipo acima que contemplará o menu e outra do tipo string que contemplará os ramais de sub-menu.

```
Public
   (......)
   rcMenu: array[0..9] of TrcMenu;
   sRamalSubMenu:array[0..9,0..9] of string;
   (......)
```

A variável rcMenu é um vetor de 0 à 9 pois utilizaremo o índice do vetor como a opção digitada pelo usuário. Por exemplo, se o usuário digitar a opção 3, pegaremos a variável rcMenu índice 3

(rcMenu[3]) para saber se deve tem sub-menu ou discagem para ramal direto.

Já o array sRamalSubMenu é um vetor bidimencional, onde o primeiro índice é a opção de menu, e o segundo a opção de sub-menu. Desta maneira, quando o usuário escolher o menu e em seguida o sub-menu, o programa já saberá para qual ramal discar:

sRamalSubMenu[op_menu,op_submenu].

No evento OnShow do form, inicializaremos essas duas variáveis com a configuração do nosso atendendor. Aqui tudo está hardcoded mas em uma aplicação real é interessante ter estes dados lidos de arquivos de configuração.

```
procedure TForm1.FormShow(Sender: TObject);
   (.........)
     //estrutura de menus e submenus
     rcMenu[3].sRamalMenu := '';
     rcMenu[3].sFrase := 'sub_coml.sig';
     rcMenu[3].sOpcoesSubMenu := '12';
     rcMenu[4].sRamalMenu := '';
     rcMenu[4].sFrase := 'sub_sup.sig';
     rcMenu[4].sOpcoesSubMenu := '12';
     rcMenu[5].sRamalMenu := '';
     rcMenu[5].sFrase := 'sub_fax.sig';
     rcMenu[5].sOpcoesSubMenu := '12';
     rcMenu[6].sRamalMenu := '241';
     rcMenu[6].sFrase := '';
     //submenu
     sRamalSubMenu[3][1] := '231';    //cons
   final
     sRamalSubMenu[3][2] := '220';    //revendas
     sRamalSubMenu[4][1] := '212';    //sup soft
     sRamalSubMenu[4][2] := '211';    //tecnica
     sRamalSubMenu[5][1] := '202';    //fax coml
     sRamalSubMenu[5][2] := '235';    //fax tec
   (.........)
end;
```

A estrutura rcMenu funciona da seguinte forma: se o campo

sRamalMenu tiver algum valor, significa que não há sub-menu e deve discar direto para o ramal (caso do índice 6). Caso contrário, o campo sFrase deverá estar preenchido com a mensagem a ser reproduzida com o sub-menu e o campo sOpcoesSubMenu contém os dígitos a serem aceitos como opções de cada um dos sub-menus.

Agora, voltemos a comentar o código da rotina TrataPrimeiroDigito da página 71.

O primeiro if (abaixo) verifica se o primeiro dígito recebido é 2, que no nosso caso significa o início de um ramal, pois todos os ramais começam com 2.

```
(....)
   if Copy(voice1.ReadDigits(Port),1,1) = '2'
then
   begin
      InsereDado(Port,'Esperando resto do
   ramal');
      Estado[Port] := ESPERA_DIGITOS_RAMAL;
      voice1.GetDigits(Port,3,'',5000,5000);
   end
   else
(....)
```

O programa deverá então esperar os outros dois dígitos do ramal. Repare que a função GetDigits passa o 3 como segundo parâmetro, indicando que deve esperar 3 dígitos. Isto está correto pois, como a propriedade AutoClearDigits está false, o buffer de dígitos do canal já entra com aquele número 2 digitado anteriormente, restando apenas mais 2 números a serem digitados para completar os 3 que a função GetDigits está esperando. Em resumo, o segundo parâmetro da função GetDigits indica, na realidade, quantos dígitos devem existir no buffer interno de dígitos do canal e não quantos dígitos devem ser digitados a partir de sua chamada (Se a propriedade AutoClearDigits estiver true, as duas situações se equivalem).

No else do if acima está o tratamento da situação de ter sido

digitado um dígito diferente do plano de numeração de ramais. Então deverá ser verificado se o dígito é uma opção de menu válida para sub-menu ou para transferência direta.

Observe a porção de código abaixo:

```
(....)
   else
   begin
      nOpMenu[Port] :=
                  StrToInt(Copy(voice1.ReadDigits(
   Port),1,1));

      if rcMenu[nOpMenu[Port]].sRamalMenu <> ''
   then
      begin
         //disca direto
         //adiciona o ramal
         lstRamal.Add(rcMenu[nOpMenu[Port]].sRama
   lMenu);
         //adiciona fuga
         lstRamal.Add(RAMAL_FUGA1);
         lstRamal.Add(RAMAL_FUGA2);
         Estado[Port] := TRANSFERENCIA;
         nEstadoRamal[Port] := LIVRE;
         //aguarde um instante...
         voice1.PlayFile(Port,'sisaguar.sig','');
      end
      else
(....)
```

Para facilitar, atribuímos à variável nOpMenu[Port] o número da opção digitada pelo usuário. Ela deve ser declarada como global. É um array de 20 posições para prever até 20 canais:

```
(....)
 public
    nOpMenu: array[1..20] of integer;
//menu
    nOpSubMenu: array[1..20] of integer;
//submenu
(....)
```

O ***if rcMenu[nOpMenu[Port]].sRamalMenu <> ''*** testa se o campo sRamalMenu tem algum conteúdo. Se tiver, entra no bloco deste if e trata a discagem direta ao ramal. Aqui pressupomos que o conteúdo de **sRamalMenu** será um ramal válido.

Quando o sistema disca para o ramal, existe a possibilidade de o ramal estar ocupado ou não atender. No nosso aplicativo, caso isso aconteça o sistema tentará transferir para os ramais de fuga que são dois. Utilizamos a mesma técnica das mensagens em seqüência, criando uma lista para os ramais. O tratamento de transferência será explicado em detalhes no item **"Transferência e Supervisão"**.

Ainda na rotina TrataPrimeiroDigito temos a necessidade de tratar o caso de sub-menu:

```
procedure TForm1.TrataPrimeiroDigito(Port:
smallint);
begin
   if Copy(voice1.ReadDigits(Port),1,1) = '2'
then
   begin
   (.......)
   end
   else
   begin
      nOpMenu[Port] :=
                  StrToInt(Copy(voice1.ReadDigits(
   Port),1,1));

      if rcMenu[nOpMenu[Port]].sRamalMenu <> ''
   then
      begin
   (.......)
      end
      else
         if rcMenu[nOpMenu[Port]].sFrase <> ''
   then
         begin
            voice1.ClearDigits(Port);
            Estado[Port] := SUBMENU;
```

```
            //fala submenu
            voice1.PlayFile(Port,
                        rcMenu[nOpMenu[Port]].sFr
ase,
                        rcMenu[nOpMenu[Port]].sOp
coesSubMenu);
      end
      else
(.......)
```

No bloco que trata o sub-menu, o sistema terá que falar uma nova mensagem e em seguida esparar a digitação de uma das opções propostas. Então é necessário apagar o buffer interno de dígitos através do método **ClearDigits(port)** , mudar o estado da aplicação para **SUBMENU** e iniciar a reprodução da mensagem de sub-menu (**PlayFile**).

Toda esta entrada de dados pode ser simplificada com os métodos **Menu*xxx*** e **Prompt*xxx***.

8.12 Tratamento de Sub-Menu

O estado de SUBMENU é muito semelhante ao de BOAS VINDAS pois reproduz uma mensagem e espera opções. A única diferença é que no SUBMENU não existe a possibilidade de digitar o ramal diretamente.

Desta forma teremos que fazer tratamentos no evento **OnPlayStop** para quando a mensagem de sub-menu acabar e no evento **OnDigitsReceived** para tratar as opções digitadas durante ou após a mensagem de sub-menu.

No evento OnPlayStop, como foi dito, podemos ter a situação da mensagem de sub-menu ter sido reproduzida até o fim (StopStatus = *ssNormal*).

Neste caso, deveremos esperar até 5 segundos para digitação da opção.

```
procedure TForm1.voice1PlayStop(Sender: TObject;
Port,
     StopStatus: Smallint);
begin

  case StopStatus of
    ssNormal:  //MENSAGEM FOI ATE O FIM
    begin
      case Estado[Port] of
        (.......)
         SUBMENU:
         begin
            //mensagem de sub menu terminou
corretamente
            //espera as opcoes
            voice1.ClearDigits(Port);
            InsereDado(Port,'Espera opcao de sub-
menu');
            Estado[Port] := ESPERA_SUBMENU;
            voice1.GetDigits(Port,1,'',5000,5000)
;
            //para esperar o ocupado
            voice1.EnableCallProgress(Port);
          end;   //fim estado submenu
      end;

    end;
  end;
```

Devemos fazer este tratamento dentro do bloco *ssNormal* pois a mensagem terminou sem interrupção e no **case** do Estado de **SUBMENU**. Primeiro apagamos os dígitos do *buffer* interno através do **ClearDigits**. Em seguida entramos na espera de um dígito por 5 segundos (GetDigits). Aqui, após chamar o **GetDigits**, habilitamos a deteção de tom com o método **EnableCallProgress(Port)** para que seja possível detectar o sinal de ocupado.

Com isso o programa poderá desligar caso o usuário desligue

antes de completar a chamada.

Da mesma forma que na mensagem principal, aqui trataremos a situação de dígito sobre mensagem ou durante o silêncio após a mensagem no evento **OnDigitsReceived**.

```
procedure TForm1.voice1DigitsReceived(Sender:
TObject; Port,
  Status: Smallint);
var
  ret: integer;
begin
  case Status of
     EdMaxDigits,
     edDigitOverMessage:  //msg interrompida por
digito
     begin
       case Estado[Port] of
          ESPERA_DIGITOS_MENU,
          BOASVINDAS:
          begin
              (.....)
          end;
          ESPERA_SUBMENU,   //tratamento de
submenu
          SUBMENU:
          begin
            //recebeu um dos digitos esperados -
disca para
            //o ramal
            voice1.DisablePulseDetection(Port);
            //testa se eh digito valido entre 0 e
9
            nOpSubMenu[Port] := StrToInt(Copy(
                                voice1.ReadDigits
(Port),1,1));

            //insere ramais para
transferencia
            lstRamal[Port].Clear;

            if sRamalSubMenu[nOpMenu[Port]]
                                 [nOpSubMenu[Port
]] <> '' then
```

```
              lstRamal[Port].Add(sRamalSubMenu[nO
pMenu[Port]]
                                                 [nOpSub
Menu[Port]]);
            //adiciona fuga
            lstRamal[Port].Add(RAMAL_FUGA1);
            lstRamal[Port].Add(RAMAL_FUGA2);
            Estado[Port] := TRANSFERENCIA;
            nEstadoRamal[Port] := LIVRE;
            //aguarde um instante...
            voice1.PlayFile(Port,'sisaguar.sig','
',0);
          end;
        end;
     end;
  end;

end;
```

Como as opções de sub-menu requerem somente um dígito, o tratamento de digito sobre mensagem recai no bloco de programa mostrado acima, onde o sistema pega a opção de sub-menu e disca para o ramal.

Aqui também utilizamos o recurso de criar uma lista para discar para os ramais em seqüência no caso de algum não atender ou estar ocupado. A variável Estado deve mudar para o estado de TRANSFERENCIA e pela primeira vez é utilizada a variável nEstadoRamal, que guardará se o ramal deu ocupado ou não atendeu. Isto é apenas necessário para o sistema falar "ramal ocupado" ou "ramal não atende".

8.13 Variáveis Globais da Aplicação

Algumas variáveis de controle necessitam ser declaradas como globais, para poderem ser acessadas de qualquer ponto da aplicação. No nosso exemplo, declaramos na área private da declaração do form, para que elas sejam globais apenas para esta janela. Isto é mera implementação. Você pode declará-las

em outro lugar, contanto que sejam globais.

> *Nem todas as variáveis serão inseridas neste momento. Faremos isso a medida que elas vão aparecendo no programa. Lembre-se: sempre que for comentado que a variável é global, declare-a neste mesmo ponto.*

A primeira e principal variável global é a que conterá o estado em que o fluxo de atendimento se encontra. Como temos que prever o tratamento independente de vários canais, é necessário que ela seja um vetor (ou array). Vamos declará-la como um vetor de 20 posições:

```
TForm1 = class(TForm)
  memStatus: TMemo;
 (.. outras linhas que nao vêem ao caso..)
private
  { Private declarations }
  Estado: array[1..20] of integer;
```

Declaramos com 20 posições, permitindo que se trate até 20 portas na mesma aplicação (canal 1 ao 20). A VoicerLib suporta atualmente até 24 portas por computador.

Parte IX

Guia de Referência

9 Guia de Referência

9.1 Propriedades

9.1.1 AutoClearDigits

Tipo: Boolean

Função: Determina se o buffer interno de dígitos de cada canal deve ser apagado ou não, automaticamente antes da chamada dos métodos GetDigits, PlayFile ou RecordFile.

Descrição:

Se a propriedade AutoClearDigits for true, sempre que um dos métodos GetDigts, PlayFile ou RecordFile forem chamados, o conteúdo ddo buffer de dígitos é apagado automaticamente. Se AutoClearDigits for false, o buffer de dígitos irá acumulando os dígitos recebidos, até que o método ClearDigits( ) seja chamado explicitamente.

Exemplo:

Delphi:

```
VoicerLibX1.AutoClearDigits := true;
```

Visual Basic:

```
VoicerLibX1.AutoClearDigits = True
```

9.1.2 CardsCount

Tipo: Integer

Função: Devolve a quantidade de placas instaladas

Descrição: É uma propriedade *readonly* e disponível somente em *runtime*. Ela informa quantas placas estão em funcionamento no sistema.

Exemplo:

```
Delphi:
X := VoicerLibX1.CardsCount;


Visual Basic:
X = VoicerLibX1.CardsCount
```

9.1.3 CardType

Tipo: Integer

Função: Determina o tipo da placa instalada na máquina.

Descrição: Podendo assumir os valores **ctVPISA** para placas VoicerPhone ISA e **ctVBPCI** para placas VoicerBox PCI/4. Quando o sistema trabalhar com placas ISA é necessário ter um arquivo de configuração definido, através do utilitário GERAINI.EXE.

Exemplo:

```
Delphi:
VoicerLibX1.CardType := ctVPISA;


Visual Basic:
VoicerLibX1.DelayComma = ctVPISA
```

9.1.4 ConfigFile

Tipo: String

Função: Determina o arquivo e o caminho de configuração para as placas VoicerPhone ISA.

Descrição: Esta propriedade permite ao programador customizar o nome e o caminho do arquivo de configuração para as placas ISA. O padrão, voicerlib.ini, será procurado no diretório do Windows.

Exemplo:

```
Delphi:
VoicerLibX1.ConfigFIle :=
'c:\teste\meuconfig.ini';


Visual Basic:
VoicerLibX1.ConfigFIle = "c:\teste\meuconfig.ini"
```

9.1.5 DelayComma

Tipo: Integer

Função: Determina o tempo em milissegundos para a pausa relativa ao caracter "," (vírgula) .

Descrição: Quando o método Dial( ) é chamado, um dos parâmetros passados é uma string contendo uma seqüência de dígitos a serem discados pela placa. Será dado uma pausa entre os dígitos anterior e posterior à "vírgula", cujo tempo é determinado pela propriedade DelayComma, para cada "vírgula" da string.

Exemplo: *Para discar o digito ZERO, uma pausa de 1 segundo*

(1000 milissegundos) e o resto do número em seqüência, utilize:

Delphi:
```
VoicerLibX1.DelayComma := 1000;
VoicerLibX1.Dial('0,72952557');
```

Visual Basic:
```
VoicerLibX1.DelayComma = 1000
VoicerLibX1.Dial("0,72952557")
```

Veja Também: DelaySemicolon, DelayDot, Dial

9.1.6 DelayDot

Tipo: Integer

Função: Determina o tempo em milissegundos para a pausa relativa ao caracter "." (ponto) .

Descrição: Quando o método Dial( ) é chamado, um dos parâmetros passados é uma string contendo uma seqüência de dígitos a serem discados pela placa. Será dado uma pausa entre os dígitos anterior e posterior ao "ponto", cujo tempo é determinado pela propriedade DelayDot, para cada "ponto" da string.

Exemplo: *Para discar o digito ZERO, uma pausa de 1 segundo (1000 milissegundos) e o resto do número em seqüência, utilize:*

Delphi:
```
VoicerLibX1.DelayDot := 1000;
VoicerLibX1.Dial('0.72952557',0);
```

Visual Basic:
```
VoicerLibX1.DelayDot = 1000
VoicerLibX1.Dial("0.72952557",0);
```

Veja Também: DelaySemicolon, DelayComma, Dial

9.1.7 DelaySemicolon

Tipo: Integer

Função: Determina o tempo em milissegundos para a pausa relativa ao caracter ";" (ponto-e-vírgula) .

Descrição: Quando o método Dial( ) é chamado, um dos parâmetros passados é uma string contendo uma seqüência de dígitos a serem discados pela placa. Será dado uma pausa entre os dígitos anterior e posterior ao "ponto-e-vírgula", cujo tempo é determinado pela propriedade DelaySemicolon, para cada "ponto-e-vírgula" da string.

Exemplo: *Para discar o digito ZERO, uma pausa de 1 segundo (1000 milissegundos) e o resto do número em seqüência, utilize:*

Delphi:
```
VoicerLibX1.DelaySemicolon := 1000;
VoicerLibX1.Dial('0;72952557');
```

Visual Basic:
```
VoicerLibX1.DelaySemicolon = 1000
VoicerLibX1.Dial("0;72952557")
```

Veja Também: DelayComma, DelayDot, Dial

9.1.8 DriverEnabled

Tipo: Boolean

Função: Determina o estado de funcionamento da placa.

Descrição: A propriedade DriverEnabled é do tipo "ReadOnly" e só está disponível em tempo de execução. Se a interrupção estiver ocorrendo DriverEnabled assume **True**, caso contrário assume False.
O evento OnErrorDetected pode ser utilizado para diagnosticar rapidamente a falha de processamento.

Exemplo: Tratamento de erro utilizando o evento OnErrorDetected:

Delphi:
```
if (not VoicerLibX1.DriverEnabled) then
    ShowMessage("Placa Inativa!");
```

Visual Basic:
```
if not VoicerLibX1.DriverEnabled then
    MsgBox "Placa Inativa"
           End if
```

Veja Também: OnErrorDetected

9.1.9 DriverVersion

Tipo: Integer

Função: Mostra a versão interna do Driver da VoicerLib

Descrição: A propriedade DriverVersion é do tipo "ReadOnly" e só está disponível em tempo de execução. Ela exibe a versão da VoicerLib e é ideal para verificar com facilidade qual a versão instalada no computador.

9.1.10 FirmwareVersion

Tipo: Integer

Função: Mostra a versão interna do firmware das placas Digivoice.

Descrição: A propriedade FirmwareVersion é do tipo "ReadOnly" e só está disponível em tempo de execução. Ela exibe a versão do firmware das placas da Digivoice (firmware é o programa que roda *dentro* do hardware).

9.1.11 ForcePlay

Tipo: Boolean

Função: Indica o tipo de tratament a ser dado no caso de erros de reprodução de arquivo.

Descrição: Até a versão 2.4, ao tentar reproduzir um arquivo e outro já estar sendo reproduzido, a biblioteca voltada erro.

Se a propriedade **ForcePlay** estiver com valor **True**, o novo arquivo a ser reproduzido interromperá o anterior e sua reprodução iniciará. Se o valor estiver False (default) o comportamento será como antes.

9.1.12 PortsCount

Tipo: Integer

Função: Devolve a quantidade de canais disponiveis.

Descrição: É uma propriedade *readonly* e disponível somente após iniciar o driver.

Exemplo:

```
Delphi:
X := VoicerLibX1.PortsCount;


Visual Basic:
X = VoicerLibX1.PortsCount
```

9.1.13 StockSigs

Tipo: String

Função: Determina o caminho onde serão encontradas os arquivos SIG 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.

Exemplo:

Delphi:
```
VoicerLibX1.StockSigsPath := 'r:\frasespadrao';
```

Visual Basic:
```
VoicerLibX1.StockSigsPath = "r:\frasespadrao"
```

A seguir apresentamos os nomes dos arquivos fornecidos com a biblioteca e a respectiva frase que reproduz.

Arquivo SIG	Frase		Arquivo SIG	Frase
Hora.sig	Hora		Horas.sig	Horas
Minuto.sig	Minuto		Minutos.sig	Minutos
Segundo.sig	Segundos		Real.sig	Real
Reais.sig	Reais		DeReais.sig	De Reais
Centavo.sig	Centavo		Centavos.sig	Centavos
Cento.sig	Cento		100.sig	Cem
200.sig	Duzentos		300.sig	Trezentos
400.sig	Quatrocentos		500.sig	Quinhentos
600.sig	Seiscentos		700.sig	Setecentos
800.sig	Oitocentos		900.sig	Novecentos
Mil.sig	Mil		milhão.sig	Milhão
milhoes.sig	Milhões		bilhão.sig	Bilhão
bilhoes.sig	Bilhões		trilhão.sig	Trilhão
trilhoes.sig	Trilhões		Virgula.sig	Vírgula
Ponto.sig	Ponto		De.sig	De

Arquivo SIG	Frase		Arquivo SIG	Frase
e.sig	E		Barra.sig	Barra
traco.sig	Traço		0.sig	Zero
1.sig	Um		2.sig	Dois
3.sig	Três		4.sig	Quatro
5.sig	Cinco		6.sig	Seis
7.sig	Sete		8.sig	Oito
9.sig	Nove		10.sig	Dez
11.sig	Onze		12.sig	Doze
13.sig	Treze		14.sig	Quatorze
15.sig	Quinze		16.sig	Dezesseis
17.sig	Dezessete		18.sig	Dezoito
19.sig	Dezenove		20.sig	Vinte

Arquivo SIG	Frase		Arquivo SIG	Frase
30.sig	Trinta		40.sig	Quarenta
50.sig	Cinqüenta		60.sig	Sessenta
70.sig	Setenta		80.sig	Oitenta
90.sig	Noventa		Janeiro.sig	Janeiro
Fevereiro.sig	Fevereiro		Marco.sig	Março
Abril.sig	Abril		Maio.sig	Maio
Junho.sig	Junho		Julho.sig	Julho
Agosto.sig	Agosto		Setembro.sig	Setembro
Outubro.sig	Outubro		Novembro.sig	Novembro
Dezembro.sig	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.

9.2 Eventos

9.2.1 OnAfterDial

Ocorre ao término de uma discagem feita por um único comando Dial.

Declarações:

Delphi:
```
OnAfterDial(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnAfterDial (Port As Integer)
```

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.

9.2.2 OnAfterFlash

Ocorre ao término de um comando Flash

Declarações:

Delphi:
```
OnAfterFlash(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnAfterFlash (Port As Integer)
```

Descrição:

De maneira semelhante ao evento OnAfterDial, o evento OnAfterFlash ocorrerá ao término do comando executado pelo método Flash.

9.2.3 OnAfterMakeCall

Ocorre ao término da função de discagem MakeCall.

Declarações:

Delphi:
```
OnAfterMakeCall(Sender: TObject,  Port, Status:
smallint);
```

Visual Basic:
```
OnAfterMakeCall (Port As Integer, Status as Integer)
```

Descrição:

Ocorre ao término da discagem com supervisão.

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.
- mkFaxDetected - Indica detecção de fax.

9.2.4 OnAfterPickUp

Ocorre ao término de um comando PickUp

Declarações:

Delphi:
```
OnAfterPickup(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnAfterPickup (Port As Integer)
```

Descrição:

De maneira semelhante ao evento OnAfterDial, o evento OnAfterPickup ocorrerá após a pausa determinada na chamada do método PickUp.

9.2.5 OnAnswerDetected

Ocorre quando a placa detecta atendimento da ligação originada.

Declarações:

Delphi:
```
OnAnswerDetected(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnAnswerDetected(Port As Integer)
```

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.

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.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Veja Também: EnableAnswerDetection, DisableAnswerDetection

9.2.6 OnBusyDetected

Ocorre quando a placa detecta tom de ocupado na linha.

Declarações:

Delphi:
```
OnBusyDetected(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnBusyDetected(Port As Integer)
```

Descrição:

A placa possui recursos de monitoração do status da linha à ela conectada (Call Progress), e de repassa-los para a aplicação, para tanto, ela deve ser habilitada a utilizar estes recursos.

O programador tem a flexibilidade de a qualquer ponto da aplicação habilitar e desabilitar tais recursos, fazendo uso dos métodos EnableBusyDetection() para habilitar e DisableBusyDetection() 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.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Veja Também: EnableCallProgress, DisableCallProgress

9.2.7 OnCallerID

Ocorre quando a placa detecta o identificador de A.

Declarações:

Delphi:
```
OnCallerID(Sender: TObject,  Port: smallint, Number:
string);
```

Visual Basic:
```
OnCallerID(Port As Integer, Number as string)
```

Descrição:

Este evento é gerado quando o identificador de A é recebido pelo tronco. Isto ocorre somente se o método IdleSettings for configurado para tal.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.
Numero– Número identificado exatamente como enviado pela companhia telefônica.

9.2.8 OnCalling

Ocorre quando a placa detecta tom chamada na linha.

Declarações:

Delphi:
```
OnCalling(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnCalling(Port As Integer)
```

Descrição:

A placa possui recursos de monitoração do status da linha à ela conectada (Call Progress), e de repassa-los para a aplicação, para tanto, ela deve ser habilitada a utilizar estes recursos.

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.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Veja Também: EnableCallProgress, DisableCallProgress, Dial, PickUp, HangUp

9.2.9 OnCallStateChange

Ocorre em várias situações durante o MakeCall.

Declarações:

Delphi:
```
OnCallStateChange(Sender: TObject,  Port: smallint;
Status: smallint);
```

Visual Basic:
```
OnCalling(Port As Integer, Status as Integer)
```

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:

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 supervisao
- 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

Veja Também: MakeCall, AbortCall

9.2.10 OnDialToneDetected

Ocorre quando a placa detecta tom de discagem na linha.

Declarações:

Delphi:

```
OnDialToneDetected(Sender: TObject,  smallint Port);
```

Visual Basic:

```
OnDialToneDetected (Port As Integer)
```

Descrição:

A placa possui recursos de monitoração do status da linha à ela conectada (Call Progress), e de repassa-los para a aplicação, para tanto, ela deve ser habilitada a utilizar estes recursos.

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 um tom de discagem, o evento OnDialToneDetected ocorrerá, desde que os recursos de monitoração estejam habilitados.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Veja Também: EnableCallProgress, DisableCallProgress, Dial, PickUp, HangUp

9.2.11 OnDigitDetected

Ocorre sempre que a placa detecta um dígito.

Declarações:

Delphi:
```
OnDigitDetected(Sender: TObject,  smallint Port,
                      smallint Digit);
```

Visual Basic:
```
OnDigitDetected(Port As Integer, Digit as Integer)
```

Descrição:

Toda vez que a placa detectar um dígito (DTMF), o evento OnDigitDetected ocorrerá, e o código ASCII dígito será passado através do parâmetro Digit.

Um dígito DTMF é um par de freqüências pré-definidas e na

faixa da voz. O programador deve estar ciente que a placa sempre está habilitada a detectar os digitos, logo, durante uma conversação é comum a placa detectar alguns dígitos indevidamente, pois ela analisa todo o conteúdo de freqüência da voz dos locutores e toda vez em que ela encontrar na voz um par de freqüências 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 digito recebido

Veja Também: GetDigits, OnDigitsReceived, PlayFile, RecordFile

9.2.12 OnDigitsReceived

Ocorre sempre depois que o método GetDigits( ) é chamado.

Declarações:

Delphi:
```
OnDigitsReceived(Sender: TObject,  smallint Port;
                        var Status: TOleEnum);
```

Visual Basic:
```
OnDigitsReceived(Port As Integer, Status As
                          VoicerLib.TxWaitDigit)
```

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 digitos

detectados devem ser recuperados através da propriedade Digits.

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 digitos estipulado.
EdTermDigit(5) – Recebeu o digito finalizador
EdDigitTimeout(3) – Estourou o tempo total para entrada de todos os digitos.
EdInterDigitTimeOut(4) – Estourou o tempo inter-dígito.
EdDigitOverMessage – Detectou dígito durante a reprodução.

Veja Também: GetDigits, PlayFile, RecordFile

9.2.13 OnErrorDetected

Ocorre quando a placa pára de responder

Declarações:

Delphi:
```
OnErrorDetected(ASender: TObject; Port,ErrorType:
Smallint);
```

Visual Basic:
```
OnErrorDetected (Port As Integer,ErrorType As Integer)
```

Descrição:

A placa deve permanecer com a interrupção ativa durante toda a operação. Caso haja algum problema e a placa pare de responder aos comandos da VoicerLib, o evento OnErrorDetected é gerado, permitindo avisar ao operador do problema e indicando o procedimento a tomar.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1. Não é utilizado neste evento.

ErrorType:

Veja Também: DriverEnabled

9.2.14 OnFaxDetected

Ocorre quando um sinal de fax é detectado pela placa.

Declarações:

Delphi:
```
OnFaxDetected(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnFaxDetected (Port As Integer)
```

Descrição:

A placa possui recursos de monitoração do status da linha à ela conectada (Call Progress), e de repassa-los para a aplicação, para tanto, ela deve ser habilitada a utilizar estes recursos.

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:

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Veja Também: EnableCallProgress, DisableCallProgress, Dial, PickUp, HangUp, Flash

9.2.15 OnLineOff

Ocorre quando o usuário desliga o aparelho telefônico conectado à placa.

Declarações:

Delphi:
```
OnLineOff(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnLineOff(Port As Integer)
```

Descrição:

O Evento OnLineOff ocorre sempre que o aparelho telefônico for desligado. É importante ressaltar que isso só ocorrerá se antes, o aparelho for retirado do "gancho" com a placa desligada.

Tanto este evento como o OnLineReady são para tratamento exclusivo do aparelho telefônico conectado à placa.

9.2.16 OnLineReady

Ocorre quando um aparelho telefônico conectado à placa toma a linha para originar uma ligação.

Declarações:

Delphi:
```
OnLineReady(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnLineReady(Port As Integer)
```

Descrição:

Existem na placa 3 conectores: "Linha", onde é conectado um linha ou ramal; "Aparelho", onde pode ser conectado um aparelho telefônico e "HeadSet", onde pode ser conectado um HeadSet.

Se a placa tomar a linha ou para originar uma ligação ou para atender uma ligação, o aparelho telefônico automaticamente fica mudo, e a conversação será feita através do HeadSet.

Caso a linha esteja livre e o aparelho conectado, este poderá atender e efetuar ligações, bastando para isso, tirar o monofone do gancho. Sempre que isso ocorrer, o evento OnLineReady também ocorrerá. Neste caso, o áudio também estará disponível no HeadSet, e a placa poderá monitorar toda conversação.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento

Veja Também: Dial

9.2.17 OnMenu

Ocorre ao término de uma função de menu iniciada pelo MenuStart

Declarações:

Delphi:

```
OnMenu(ASender: TObject; Port: Smallint;const
OptionSelected: WideString; Status: Smallint);
```

Visual Basic:

```
OnMenu(Port As Integer, OptionSelected as string,
Status as
       Integer)
```

Descrição:

O evento OnMenu é gerado após o término da função especial de menu

Parâmetros Recebidos:

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

9.2.18 OnPlayStart

Ocorre sempre quando for iniciado o playback de qualquer mensagem.

Declarações:

Delphi:
```
OnPlayStart(Sender: TObject,  smallint Port);
```

C++ Builder:
```
OnPlayStart(TObject *Sender,  short &Port);
```

Visual Basic:
```
OnPlayStart(Port As Integer)
```

Visual C++:
```
OnPlayStartVoicerlibx1(short FAR* Port);
```

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

Veja Também: PlayFile, StopPlayFile, OnPlayStop

9.2.19 OnPlayStop

Ocorre sempre quando for finalizado o playback de qualquer

mensagem.

Declarações:

Delphi:
```
OnPlayStop(ASender: TObject; Port,StopStatus:
                                        Smallint);
```

Visual Basic:
```
OnPlayStop(Port As Integer, StopStatus As Integer)
```

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 digito 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.

Veja Também: PlayFile, StopPlayFile

9.2.20 OnPrompt

Ocorre quando a função de prompt chega ao fim.

Declarações:

Delphi:
```
OnPrompt(Sender: TObject,  smallint Port, Value:
string, Status: smallint);
```

Visual Basic:
```
OnPrompt(Port As Integer, Value as string, Status as
integer)
```

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

9.2.21 OnRecording

Ocorre durante a gravação de uma mensagem.

Declarações:

Delphi:
```
OnRecordgin(Sender: TObject,  smallint Port, var
```

```
Duration: Integer);
```

Visual Basic:
```
OnResetDetected (Port As Integer, Duration as Long)
```

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.

9.2.22 OnRecordStart

Ocorre sempre quando for iniciado a gravação de qualquer mensagem.

Declarações:

Delphi:
```
OnRecordStart(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnRecordStart(Port As Integer)
```

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.

Veja Também: RecordFile, StopRecordFile, OnRecordStop

9.2.23 OnRecordStop

Ocorre sempre quando for finalizado a gravação de qualquer mensagem.

Declarações:

Delphi:
```
OnRecordStop(ASender: TObject; Port,StopStatus:
        Smallint);
```

Visual Basic:
```
OnRecordStop(Port As Integer, StopStatus As Integer)
```

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 digito 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.

Veja Também: RecordFile, StopRecordFile

9.2.24 OnRingDetected

Ocorre sempre quando for detectado o sinal de Ring na linha.

Declarações:

Delphi:
```
OnRingDetected(Sender: TObject,  smallint Port);
```

Visual Basic:
```
OnRingDetected (Port As Integer)
```

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.

Parâmetros Recebidos:

Port – Indica o canal da Placa que gerou o evento – Para VoicerPhone, o valor é fixo em 1.

Veja Também: PickUp

9.2.25 OnSampleReceived

Ocorre em intervalos regulares de tempo permitindo a leitura de amostras GSM.

Declarações:

Delphi:

```
OnSampleReceived(Sender: TObject, smallint Port);
```

Visual Basic:

```
OnSampleReceived (Port As Integer)
```

Descrição:

Este evento ocorre em intervalos de tempo suficiente para ler amostras vindas da placa através do método GetSamples.

9.3 Métodos

9.3.1 AbortCall

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

Declarações:

Delphi:
```
function AbortCall(Port: Smallint): Smallint;
```

Visual Basic:
```
Function AbortCall (Port As Integer) As Integer
```

Descrição:

Durante o processo de discagem automática é possível cancelar através da chamada deste método

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

Retorna zero se foi executado com sucesso.

9.3.2 ClearDigits

Apaga o conteúdo do buffer de dígitos de cada canal.

Declarações:

Delphi:
```
function ClearDigits(Port: Smallint): Smallint;
```

Visual Basic:
```
Function ClearDigits(Port As Integer) As Integer
```

Descrição:

É utilizada para apagar o buffer interno de dígitos detectados, principalmente quando a propriedade AutoClearDigits está com valor *false.*

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

Retorna zero se foi executado com sucesso.

Veja Também: ReadDigits, AutoClearDigits

9.3.3 ConferenceAddPort

Adiciona canal a determinada conferencia.

Declarações:

Delphi:
```
function ConferenceAddPort(Handle: Smallint; Port:
Smallint): Smallint;
```

Visual Basic:
```
Function ConferenceAddPort(Handle As Integer, Port As
Integer) As Integer
```

Descrição:

Após criada a conferencia é necessário adicionar os canais desejados a ela, para adicionar os canais chama-se essa função passando o canal e a conferencia em que ele será adicionado.

Parâmetros:

Handle - Obtido como retorno da função CreateConferenceResource
Port - Canal fisico da placa.

Valor de Retorno:

-1 - Driver não habilitado
-2 - Numero maximo de canais por recurso alcançado (sala cheia)
-3 - Porta já está alocada em outro recurso
0 - Porta Inserida com sucesso

Veja Também: ConferenceRemovePort

9.3.4 ConferenceDisablePort

Desabilita porta alocada em um recurso de conferencia a converar com as outras

Declarações:

Delphi:
```
function ConferenceDisablePort(Port: Smallint):
Smallint;
```

Visual Basic:
```
Function ConferenceDisablePort(Port As Integer) As
Integer
```

Descrição:

Este método é importante pois, caso um dos canais queira ouvir um menu, os outros canais não irão ouvir também, basta desabilitar o canal através do método deste método, reproduzir o menu que somente ele ouvirá, e após isso habilitá-lo novamente através do método ConferenceEnablePort. Isso sem precisar remover o canal da conferencia.

Parâmetros:

Port - Canal fisico da placa.

Valor de Retorno:

0 - Retorno Padrão

Veja Também: ConferenceEnablePort

9.3.5 ConferenceEnablePort

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

Declarações:

Delphi:
```
function ConferenceEnablePort(Port: Smallint):
Smallint;
```

Visual Basic:
```
Function ConferenceEnablePort(Port As Integer) As
Integer
```

Descrição:

Mesmo que o canal esteja alocado em uma determinada conferencia é necessário que ele esteja habilitado para que possa se comunicar com os outros canais. Isso é importante pois, caso um dos canais queira ouvir um menu, os outros canais não irão ouvir também, basta desabilitar o canal, através do método ConferenceDisablePort, reproduzir o menu que somente ele ouvirá, e após isso habilitá-lo novamente através

deste método. Isso sem precisar remover o canal da conferencia.

Parâmetros:

Port - Canal fisico da placa.

Valor de Retorno:

-1 - Driver não habilitado ou canal não alocado em nenhum recurso
0 - Canal habilitado com sucesso

Veja Também: ConferenceDisablePort

9.3.6 ConferenceGetHandle

Pega o Handle de determinada porta.

Declarações:

Delphi:
```
function ConferenceGetHandle(Port: Smallint): Smallint;
```

Visual Basic:
```
Function ConferenceGetHandle(Port As Integer) As
Integer
```

Descrição:

Esse método deve ser utilizado para saber em que conferencia o canal está alocado.

Parâmetros:

Port - Canal fisico da placa.

Valor de Retorno:

>= 0 - O handle a que a porta pertence
-1 - Canal não alocado

Veja Também: CreateConferenceResource

9.3.7 ConferenceRemovePort

Remove canal de determinada conferencia.

Declarações:

Delphi:
```
function ConferenceRemovePort(Handle: Smallint; Port:
Smallint): Smallint;
```

Visual Basic:
```
Function ConferenceRemovePort(Handle As Integer, Port
As Integer) As Integer
```

Descrição:

Para remover os canais de determinada conferencia chama-se essa função passando o canal e a conferencia de que ele será removido.

Parâmetros:

Handle - Obtido como retorno da função CreateConferenceResource
Port - Canal fisico da placa.

Valor de Retorno:

-1 - Driver não habilitado
0 - Porta Removda com sucesso

Veja Também: ConferenceAddPort

9.3.8 CreateConferenceResource

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

Declarações:

Delphi:
```
function CreateConferenceResource(MaxPorts: Smallint):
Smallint;
```

Visual Basic:
```
Function CreateConferenceResource(MaxPorts As Integer)
As Integer
```

Descrição:

A criação da conferencia permite que um numero determinado de canais conversem entre si. Ao criar esse recurso o valor de retorno é utilizado como um código para cada conferência (cada sala).

Parâmetros:

MaxPorts – Maximo de canais permitido nesta conferencia ("sala").

Valor de Retorno:

>= 0 é o Handle válido a ser usado nas outras funções
-1 - Erro ao criar recurso

Veja Também: DeleteConferenceResource

9.3.9 DeleteConferenceResource

Remove recurso para conferencia ("sala de bate papo").

Declarações:

Delphi:
```
function DeleteConferenceResource(Handle: Smallint):
Smallint;
```

Visual Basic:
```
Function DeleteConferenceResource(Handle As Integer) As
Integer
```

Descrição:

Deve-se utilizar esse método para excluir uma conferencia já existente, passando o Handle que é código utilizado para identificar cada conferencia.

Parâmetros:

Handle - Obtido como retorno da função CreateConferenceResource

Valor de Retorno:

-1 - Driver não foi iniciado
-2 - Recurso não existia
0 - Remoção do recurso sem problemas

Veja Também: CreateConferenceResource

9.3.10 Dial

Disca uma seqüencia de números ou pausa.

Declarações:

Delphi:
```
function Dial(Port: Smallint; const Number: WideString;
                        PauseAfterDial: Integer,
DialType: integer): Smallint;
```

Visual Basic:
```
Function Dial(Port As Integer, Number As String,
                        PauseAfterDial as Integer,
DialType as Integer) As Integer
```

Descrição:

O método Dial permite enviar a VoicePhone até 16 digitos 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 das propriedades DelayComma, DelaySemicolon e DelayDot (assumem o padrão de 1 segundo).

O tipo de discagem (pulso ou tom) é determinado pelo 4o.

parâmetro, o *DialType*.

Parâmetros:

Port – Indica o canal da Placa
Number – String contendo os dígitos ou pausas para discagem.
PauseAfterDial - É um tempo extra em milisegundos que é dado após a discagem. O evento OnAfterDial só será gerado após este tempo.
DialType - Pode assumir o valor dtTone ou dtPulse, indicando se a discagem é por pulso e tom.

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Exemplos:

Delphi:
```
VoicerLibX1.Dial(1,'0,72952557...202',1000);
```

Visual Basic:
```
x = VoicerLibX1.Dial(1,"0,72952557...202" ,1000)
```

Veja Também: DelayComma, DelaySemicolon, DelayDot

9.3.11 DisableAnswerDetection

Desabilita a supervisão de atendimento.

Declarações:

Delphi:
```
function DisableAnswerDetection(Port: Smallint):
```

```
Smallint;
```

Visual Basic:

```
Function DisableAnswerDetection (Port As Integer) As
Integer
```

Descrição:

O método DisableAnswerDetection desabilita a supervisão de atendimento. Esta supervisão permite detectar ou não quando a ligação foi atendida.

Deve ser usado somente em momentos onde o atendimento tem sentido, principalmente durante a chamada.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: EnableAnswerDetection, OnAnswerDetected, OnFaxDetected, OnCalling

9.3.12 DisableCallProgress

Desabilita a supervisão de linha, fax e ocupado.

Declarações:

Delphi:

```
function DisableCallProgress(Port: Smallint): Smallint;
```

Visual Basic:
```
Function DisableCallProgress(Port As Integer) As
Integer
```

Descrição:

O método DisableCallProgress desabilita a supervisão de linha. A supervisão de linha permite monitorar o sinal de chamada, ocupado, fax e tom de discagem.

Quando o sistema é iniciado, a supervisão está desabilitada como padrão.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: EnableCallProgress, OnAnswerDetected, OnFaxDetected, OnCalling

9.3.13 DisableEchoCancel

Desabilita cancelamento de eco em recursos de conferência.

Declarações:

Delphi:
```
function DisableEchoCancel: Smallint;
```

Visual Basic:
```
function DisableEchoCancel As Integer
```

Descrição:

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

Valor de Retorno:

Retorna sempre zero.

9.3.14 DisablePulseDetection

Desabilita a deteção de pulso.

Declarações:

Delphi:
```
function DisablePulseDetection(Port: Smallint):
Smallint;
```

Visual Basic:
```
Function DisablePulseDetection (Port As Integer) As
Integer
```

Descrição:

O método DisablePulseDetection desabilita a deteção de pulso.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de

erro.

9.3.15 DisableSampleToApp

Desabilita o envio de amostras GSM da placa para a aplicação.

Declarações:

Delphi:

```
procedure DisableSampleToApp(Port: Smallint)
```

Visual Basic:

```
Sub DisableSampleToApp (Port As Integer)
```

Descrição:

Este método desliga os procedimentos iniciados pelo EnableSampleToCard

Parâmetros:
Port – Indica o canal da Placa.

Valor de Retorno:

0 - OK

9.3.16 DisableSampleToCard

Desabilita o envio de amostras GSM da aplicação para a placa.

Declarações:

Delphi:

```
procedure DisableSampleToCard(Port: Smallint)
```

Visual Basic:

```
Sub DisableSampleToCard (Port As Integer)
```

Descrição:

Este método desliga os procedimentos iniciados pelo EnableSampleToCard

Parâmetros:

Port – Indica o canal da Placa.

Valor de Retorno:

0 - OK

9.3.17 EnableAnswerDetection

Habilita a supervisão de atendimento.

Declarações:

Delphi:

```
function EnableAnswerDetection(Port: Smallint):
Smallint;
```

Visual Basic:

```
Function EnableAnswerDetection(Port As Integer) As
Integer
```

Descrição:

O método EnableAnswerDetection habilita a supervisão de atendimento. A supervisão de linha permite monitorar quando uma ligação for atendida.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: DisableAnswerDetection, OnAnswerDetected, OnBusyDetected, OnFaxDetected, OnCalling

9.3.18 EnableCallProgress

Habilita a supervisão de linha.

Declarações:

Delphi:
```
function EnableCallProgress(Port: Smallint): Smallint;
```

Visual Basic:
```
Function EnableCallProgress(Port As Integer) As Integer
```

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 chamar este método

quando se quiser monitorar a linha.

Parâmetros:

Port – Indica o canal da Placa.

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: DisableCallProgress, OnAnswerDetected, OnFaxDetected, OnCalling

9.3.19 EnableEchoCancel

Habilita cancelamento de eco em recursos de conferência.

Declarações:

Delphi:
```
function EnableEchoCancel: Smallint;
```

Visual Basic:
```
function EnableEchoCancel As Integer
```

Descrição:

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

Valor de Retorno:

Retorna sempre zero.

9.3.20 EnablePulseDetection

Habilita a deteção de pulso.

Declarações:

Delphi:
```
function EnablePulseDetection(Port: Smallint):
Smallint;
```

Visual Basic:
```
Function EnablePulseDetection (Port As Integer) As
Integer
```

Descrição:

O método EnablePulseDetection habilita a deteção de pulso. Só é possível detectar pulso com razoável precisão durante o silêncio.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.21 EnableSampleToApp

Habilita o envio de amostras GSM da placa para a aplicação.

Declarações:

Delphi:
```
procedure EnableSampleToApp(Port: Smallint)
```

Visual Basic:
```
Sub EnableSampleToApp (Port As Integer)
```

Descrição:

Este método faz com que todas as amostras de áudio (GSM) vindas da placa sejam disponibilizadas para a aplicação através do método OnSampleReceived.

Parâmetros:

Port – Indica o canal da Placa.

Valor de Retorno:

0 - OK
1 - Amostras já habilitadas
3 - Driver Desabilitado

9.3.22 EnableSampleToCard

Habilita o envio de amostras GSM da aplicação para a placa.

Declarações:

Delphi:

```
procedure EnableSampleToCard(Port: Smallint)
```

Visual Basic:

```
Sub EnableSampleToCard (Port As Integer)
```

Descrição:

Este método faz com que amostras de áudio (GSM) possam ser enviadas para a placa através do método PutSamples, fazendo com que sejam reproduzidas pela placa através da porta especificada, com funcionamento similar a um PlayFile.

Parâmetros:

Port – Indica o canal da Placa.

Valor de Retorno:

0 - OK
1 - Amostras já habilitadas
3 - Driver Desabilitado

9.3.23 Flash

Executa um comando de flash para a central PABX.

Declarações:

Delphi:

```
function Flash(Port: Smallint; Milliseconds: Integer;
                    PauseAfterFlash: integer):
Smallint;
```

Visual Basic:

```
Function Flash(Port As Integer, Milliseconds As Long,
                    PauseAfterFlash as Integer) As
Integer
```

Descrição:

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.

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:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.24 GetDigits

Inicia a espera de uma seqüência de dígitos.

Declarações:

Delphi:

```
function GetDigits(Port, MaxDigits: Smallint; const
                   TermDigits: WideString;
                   DigitsTimeOut, InterDigitsTimeOut:
                   Integer): Smallint;
```

Visual Basic:

```
Function GetDigits(Port As Integer,
MaxDigits As Integer, TermDigits As
String, DigitsTimeOut As Long,
InterDigitsTimeOut As Long) As
Integer
```

Descrição:

O método GetDigits permite iniciar a espera de um conjunto de digitos, por um determinado tempo ou até receber um digito 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.

Parâmetros:

Port – Indica o canal da Placa
MaxDigits – Número máximo de dígitos permitido. Utilize esta propriedade para limitar o número de digitos 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).
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 correspondete ao time-out interdígito. Em milissegundos.

Valor de Retorno:

Retorna zero se foi *iniciado* com sucesso ou 1 no caso de erro.

Veja Também: OnDigitsReceived, ReadDigits

9.3.25 GetPortStatus

Recupera o status do canal especificado

Declarações:

Delphi:
```
function GetPortStatus(Port: Smallint): Smallint;
```

Visual Basic:
```
Function GetPortStatus(Port As Integer) As Integer
```

Descrição:

Este método permite ao desenvolvedor 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

9.3.26 GetSamples

Lê amostras da placa.

Declarações:

Delphi:
```
function GetSamples(Port: Smallint): Variant;
```

Visual Basic:
```
Sub GetSamples(Port as Integer) as Variant
```

Descrição:

Este método lê um grupo de 33 (1 frame GSM) amostras vindas da placa e devolve em um array do tipo Variant (tipo de dado variável). Normalmente esta função deve ser chamada de dentro do evento OnSampleReceived.

Parâmetros:
Port – Indica o canal da Placa.

Valor de Retorno:
Vetor de amostras lidas

9.3.27 GsmToWave

Converte do formato GSM para o formato Wave.

Declarações:

Delphi:

```
function GsmToWave(const Source: WideString,
Destination: WideString):
                                                    Sma
                                                    lli
                                                    nt;
```

Visual Basic:

```
Function GsmToWave(Source As String, Destination As
String)
                                                    As
                                                    Int
                                                    ege
                                                    r
```

Parâmetros:

Source – String contendo o nome e caminho completo do arquivo GSM que será convertido.
Destination – String contendo o nome e caminho completo do arquivo Wave que será gravado.

Valores de Retorno:

0 - Executado com sucesso
1 - Não foi encontrado o arquivo de origem (Source)
2 - Não foi possível criar o arquivo de destino (Destination)
3 - Arquivo GSM corrompido.
4 - Erro durante a gravação do arquivo de destino.
5 - Erro durante a leitura do arquivo de origem.

Veja Também: WaveToGsm.

9.3.28 HangUp

Libera a linha conectada a placa (desliga).

Declarações:

Delphi:
```
function HangUp(Port: Smallint): Smallint;
```

Visual Basic:
```
Function HangUp(Port As Integer) As Integer
```

Descrição:

Ao chamar este método, a linha é desconectada e a porta liberada. Equivale ao desligar do telefone.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: PickUp

9.3.29 IdleAbort

Interrompe a execução de uma função de Idle.

Declarações:

Delphi:

```
function IdleAbort(Port: Smallint): Smallint;
```

Visual Basic:
```
Function IdleAbort (Port As Integer) As Integer
```

Descrição:

Ao chamar este método a função iniciada pelo IdleStart será interrompida para o canal específico.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.30 IdleSettings

Configura a monitoração do estado de espera.

Declarações:

Delphi:
```
function IdleStart(Port: Smallint; AutoPickUp: boolean;
    RingCount, PauseAfterPickUp: smallint;
    WatchTrunkBefore,
    WatchTrunkAfter: boolean; Format, TimeOut, Max:
    smallint; TermDigits: string): Smallint;
```

Visual Basic:
```
Function IdleStart (Port As Integer AutoPickUp as
boolean,
    RingCount, PauseAfterPickUp as integer,
    WatchTrunkBefore,
```

```
WatchTrunkAfter as boolean, Format, TimeOut, Max as
integer, TermDigits as string) as integer
```

Descrição:

Esta função configura as opções a serem monitoradas durante o estado de espera. Permite configurar deteção automática de bina (OnCallerID), atendimento automático e integração com o PABX

Parâmetros:

- **Port –** Indica o canal da Placa que gerou o evento.
- **AutoPickUp –** Indica se atende automático ou não.
- **RingCount –** Número de rings para o atendimento automático
- **PauseAfterPickUp –** Pausa após o pickup
- **WatchTrunkBefore** – Monitora a linha antes do atendimento. Deve ser utilizado para sinalizações antes do Ring (BINA)
- **WatchTrunkAfter -** Monitora a linha depois do atendimento.
- **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.

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: IdleAbort, IdleStart

9.3.31 IdleStart

Inicia a execução de uma função de Idle.

Declarações:

Delphi:
```
function IdleStart(Port: Smallint): Smallint;
```

Visual Basic:
```
Function IdleStart (Port As Integer) As Integer
```

Descrição:

Ao chamar este método a função de monitoração de estado de espera é iniciada.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: IdleAbort, IdleSettings

9.3.32 IsCallInProgress

Indica se existe uma discagem em curso.

Declarações:

Delphi:

```
function IsCallInProgress (Port: Smallint): Boolean;
```

Visual Basic:
```
Function IsCallInProgress (Port As Integer) As Boolean
```

Descrição:

Esta função permite saber se existe uma discagem em curso.

Parâmetros:

Port – Indica o canal da Placa que gerou o evento

Valor de Retorno:

True ou False

9.3.33 IsPlaying

Indica se a placa está reproduzindo alguma mensagem

Declarações:

Delphi:
```
function IsPlaying(Port: Smallint): boolean;
```

Visual Basic:
```
Function IsPlaying(Port As Integer) As Boolean
```

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:

True **ou** False

9.3.34 IsRecording

Indica se a placa está gravando alguma mensagem

Declarações:

Delphi:
```
function IsRecording(Port: Smallint): boolean;
```

Visual Basic:
```
Function IsRecording (Port As Integer) As Boolean
```

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:

True **ou** False

9.3.35 MakeCall

Inicia a discagem com supervisão.

Declarações:

Delphi:
```
procedure MakeCall(Port: Smallint; CallType: Smallint;
           Number: string; InitialPhrase: String;
           WithAnalysis: boolean; DialType:smallint):
           smallint;
```

Visual Basic:
```
Function MakeCall(Port As Integer, CallType as integert;
           Number as string; InitialPhrase as String;
           WithAnalysis as boolean;DialType as
           Smallint) as integer;
```

Descrição:

Este método inicia a discagem com supervisão conforme configurado pelos métodos **SetCall*xxxxx*.** O término será tratado no evento OnAfterMakeCall.

Parâmetros:

Port – Indica o canal da Placa.
CallType – ctExternal ou ctWithFlash
Number – Número que será discado
InitialPhrase – 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:

Retorna Zero se executado com sucesso.

9.3.36 MenuAbort

Interrompe a execução de uma função de Menu.

Declarações:

Delphi:
```
function MenuAbort(Port: Smallint): Smallint;
```

Visual Basic:
```
Function MenuAbort(Port As Integer) As Integer
```

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:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: MenuStart

9.3.37 MenuErrorSettings

Configura as opções de menu

Declarações:

Delphi:
```
function MenuErrorSettings(Port: Smallint;
           InvalidDigitPhrase:string;
           InvalidDigitRetries: smallint;TimeOutPhrase:
           String;): smallint;
```

Visual Basic:

```
Function MenuErrorSettings(Port As Integer,
InvalidDigitPhrase
                          as string, InvalidDigitRetries
                          as integer,
                          TimeOutPhrase as string) as
                          integer
```

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:

Retorna Zero se executado com sucesso.

9.3.38 MenuStart

Inicia a função de menu.

Declarações:

Delphi:

```
function MenuStart(Port: Smallint;
           Message:string; ValidDigits: string;TimeOut:
           smallint): smallint;
```

Visual Basic:

```
Function MenuStart(Port As Integer, Message as string,
                          ValidDigits as string,
                          TimeOut as integer) as integer
```

Descrição:

Inicia a execução de um menu. Após sua execução o método OnMenu será chamado devolvendo o dígito escolhido e o status.

Parâmetros:

Port – Indica o canal da Placa.
Message – Frase do menu
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**.

Retorno:

Retorna Zero se executado com sucesso.

9.3.39 MicOff

Desconecta o áudio do microfone ao HeadSet. Somente para placas de 1 canal.

Declarações:

Delphi:
```
function MicOff(Port: Smallint): Smallint;
```

Visual Basic:
```
Function MicOff(Port As Integer) As Integer
```

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.

Veja Também: MicOn

9.3.40 MicOn

Conecta o áudio do microfone ao HeadSet.

Declarações:

Delphi:
```
function MicOn(Port: Smallint): Smallint;
```

Visual Basic:
```
Function MicOn(Port As Integer) As Integer
```

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.

Veja Também: MicOff

9.3.41 PhoneOff

Desconecta o áudio do fone ao HeadSet.

Declarações:

Delphi:
```
function PhoneOff(Port: Smallint): Smallint;
```

Visual Basic:
```
Function PhoneOff(Port As Integer) As Integer
```

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.

Veja Também: PhoneOn

9.3.42 PhoneOn

Conecta o áudio do fone ao HeadSet.

Declarações:

Delphi:
```
function PhoneOn(Port: Smallint): Smallint;
```

Visual Basic:
```
Function PhoneOn(Port As Integer) As Integer
```

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.

Veja Também: PhoneOff

9.3.43 PickUp

Atende a linha conectada a placa (Liga).

Declarações:

Delphi:

```
function PickUp(Port: Smallint; PauseAfterPickUp:
```

```
integer):
                                                                Sma
llint;
```

Visual Basic:

```
Function PickUp (Port As Integer, PauseAfterPickUp as
Integer)
                                                               As
Integer
```

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. 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 é gerado após decorrido este tempo.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: HangUp

9.3.44 PlayCardinal

Permite reproduzir numerais cardinais inteiros ou fracionários por extenso.

Declarações:

Delphi:
```
function PlayCardinal(Port: Smallint; Value: string;
                              TermDigits: string;
                              PauseBefore: integer):
                              smallint;
```

Visual Basic:
```
Function PlayCardinal (Port As Integer, Value as
String,
                  TermDigits as String, PauseBefore as
                  integer) as Integer
```

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* milisegundos antes de iniciar a reprodução

Valor de Retorno:

Retorna zero se foi iniciado com sucesso ou diferente de Zero no caso de erro

9.3.45 PlayCurrency

Permite reproduzir valores monetários por extenso.

Declarações:

Delphi:

```
function PlayCurrency(Port: Smallint; Value: string;
                              TermDigits: string;
                              PauseBefore: integer):
                              smallint;
```

Visual Basic:

```
Function PlayCurrency (Port As Integer, Value as
String,
                  TermDigits as String, PauseBefore as
                  integer) as Integer
```

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* milisegundos antes de iniciar a reprodução

Valor de Retorno:

Retorna zero se foi iniciado com sucesso ou diferente de Zero no caso de erro

9.3.46 PlayDate

Permite reproduzir data por extenso.

Declarações:

Delphi:

```
function PlayDate(Port: Smallint; Value: string; Mask:
string;
```

```
                                    TermDigits: string;
                                    PauseBefore: integer):
                                    smallint;
```

Visual Basic:

```
Function PlayDate (Port As Integer, Value as String,
                   Mask as string, TermDigits as String,
                   PauseBefore as integer) as Integer
```

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* milisegundos antes de iniciar a reprodução

Valor de Retorno:

Retorna zero se foi iniciado com sucesso ou diferente de Zero no caso de erro

9.3.47 PlayFile

Inicia a reprodução de um arquivo através da placa.

Declarações:

Delphi:
```
function PlayFile(Port: Smallint; const File_,
           TermDigits: WideString; Origin: integer):
           Smallint;
```
Visual Basic:
```
Function PlayFile(Port As Integer, File As String,
         TermDigits As String, Origin as Integer) As
Integer
```

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 é gerado e o evento OnPlayStrop 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.

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: Retorna zero se foi *iniciado* com sucesso ou 1 no caso de erro.

9.3.48 PlayList

Inicia a reprodução de uma lista de mensagens de um determinado canal.

Declarações:

Delphi:
```
function PlayList(Port: Smallint; TermDigits:string):
smallint;
```

Visual Basic:
```
Function PlayList(Port As Integer,TermDigits as String)
as
                                                        i
                                                        n
                                                        t
                                                        e
                                                        g
                                                        e
                                                        r
```

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:

0 – OK
1 – Lista de mensagens vazia
3 – Driver desabilitado

9.3.49 PlayListAdd

Permite adicionar itens a serem reproduzidos na lista do canal.

Declarações:

Delphi:
```
function PlayListAdd(Port: Smallint; ItemType Smallint,
        Value: string; Mask: string; PauseBefore:
        integer): Smallint;
```

Visual Basic:
```
Function PlayListAdd(Port As Integer, ItemType as
integer,
                  Value As String, Mask as String,
                  PauseBefore as Integer) As Integer
```

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* milisegundos antes de iniciar a reprodução

Valor de Retorno:

Retorna zero se foi *iniciado* com sucesso ou 1 no caso de erro.

9.3.50 PlayListClear

Elimina todos os itens na lista do canal.

Declarações:

Delphi:
```
procedure PlayListClear(Port: Smallint)
```

Visual Basic:
```
Sub PlayListClear(Port As Integer)
```

Descrição:

Este método deve ser chamado antes do método PlayListAdd para eliminar todos os elementos que porventura estejam lá.

Parâmetros:

Port – Indica o canal da Placa.

Valor de Retorno:

A função não retorna nada.

9.3.51 PlayListGetCount

Retorna a quantidade de elementos na lista daquele canal.

Declarações:

Delphi:
```
function PlayListGetCount(Port: Smallint): smallint;
```

Visual Basic:
```
Function PlayListGetCount(Port As Integer) as integer
```

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

9.3.52 PlayListRemoveItem

Remove o item especificado da lista de mensagens do canal

Declarações:

Delphi:
```
function PlayListRemoveItem(Port, Index: Smallint):
smallint;
```

Visual Basic:
```
Function PlayListRemoveItem (Port, Index As Integer) as
                                    integer
```

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.

9.3.53 PlayNumber

Permite reproduzir números dígito a digito.

Declarações:

Delphi:
```
function PlayNumber(Port: Smallint; Value: string;
                            TermDigits: string;
                            PauseBefore: integer):
                            smallint;
```

Visual Basic:
```
Function PlayNumber (Port As Integer, Value as String,
                  TermDigits as String, PauseBefore as
                  integer) as Integer
```

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* milisegundos antes de iniciar a reprodução

Valor de Retorno:

Retorna zero se foi iniciado com sucesso ou diferente de Zero no caso de erro

9.3.54 PlayTime

Permite reproduzir hora por extenso.

Declarações:

Delphi:
```
function PlayTime(Port: Smallint; Value: string;
                              TermDigits: string;
                              PauseBefore: integer):
                              smallint;
```

Visual Basic:
```
Function PlayTime(Port As Integer, Value as String,
                  TermDigits as String, PauseBefore as
                  integer) as Integer
```

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* milisegundos antes de iniciar a reprodução

Valor de Retorno:

Retorna zero se foi iniciado com sucesso ou diferente de Zero no caso de erro

9.3.55 PromptAbort

Interrompe a execução de uma função de PromptStart.

Declarações:

Delphi:
```
function PromptAbort(Port: Smallint): Smallint;
```

Visual Basic:
```
Function PromptAbort (Port As Integer) As Integer
```

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:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: PromptStart

9.3.56 PromptSettings

Configura as opções de prompt

Declarações:

Delphi:

```
function PromptSettings(Port:
Smallint;PlaybackPhrase:string;
       ConfirmationPhrase: string;RetryDigit: string;
       ConfirmationDigit:string; CancelDigit:
       string):smallint
```

Visual Basic:

```
Function PromptSettings(Port As Integer, PlaybackPhrase
as
              string, ConfirmationPhrase as string,
              RetryDigit as string, ConfirmationDigit
              as string, CancelDigit as string) as
              integer
```

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.: *"Tecle * para confirmar, # para cancelar ou 9 para digitar de*

novo"
RetryDigit – Digito que indicará redigitação dos dados
ConfirmationDigit - Digito que indicará aceitação dos dados
CancelDigit - Digito que indicará cancelamento da entrada de dados

Retorno:

Retorna Zero se executado com sucesso.

9.3.57 PromptStart

Inicia a função especial de entrada de dados

Declarações:

Delphi:
```
function PromptStart(Port: Smallint; Message: string;
                  MinDigit, MaxDigit,Timeout: smallint;
                  InterdigitTimeout: smallint;
                  TermDigits: string; WithConfirmation,
                  WithPlayback: boolean; Retries:
                  smallint):
                                          Smallin
                                          t;
```
Visual Basic:
```
Function PromptAbort(Port As Integer, Message as
string,
                  MinDigit, MaxDigit,Timeout as
                  integer;
                  InterdigitTimeout as integer;
                  TermDigits as string;
                  WithConfirmation,
                  WithPlayback as boolean;
                  Retries as integert) as integer
```

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 digito 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 – Timout 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:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.58 PutSamples

Envia amostras para um canal da placa.

Declarações:

Delphi:

```
     Function PutSamples(Port: integer, VARIANT
SamplesArray)
```

Visual Basic:

```
    Sub PutSamples(Port As Integer, SamplesArray)
```

Descrição:

Este método envia o conteúdo de SamplesArray para um determinado canal da placa, permitindo que estas amostras sejam reproduzidas pela placa.

Parâmetros:
Port – Indica o canal da Placa.
SamplesArray – Vetor Variant de 33 amostras

Valor de Retorno:
0 – OK
> 0 - Falha

9.3.59 ReadDigits

Lê o conteúdo do buffer de dígitos do canal especificado

Declarações:

Delphi:

```
function ReadDigits(Port: Smallint): string;
```

Visual Basic:

```
Function ReadDigits(Port As Integer) As String
```

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

Valor de Retorno:

Retorna uma string com os dígitos ou nulo caso não haja nada

9.3.60 ReadSecurityWord

Permite ler a string de segurança da memória da placa.

Declarações:

Delphi:
```
function ReadSecurityWord(Card: smallint; Param:
                                  WideString):string;
```

Visual Basic:
```
Function WriteSecurityWord(Card as Integer,
                                Param as String) as
String
```

Descrição:

Esta função permite ler 10 caracteres da memória da placa. Sempre são lidos os 10 caracteres em uma única chamada da função. Esta função é relativamente lenta e, por isso, não deve ser utilizada durante algum processamento mais intenso.

Parâmetros:

Card – Indica a **placa** a ser lida, indo de 1 a **n**.
Param – String reservada. Passar sempre nula.

OBS.: *Esta função só está disponível em placas equipadas com memória EEPROM (v1.6 em diante) da VoicerPhone e em todas as VoicerBox PCI/4*

9.3.61 RecordFile

Inicia a gravação de um arquivo através da placa.

Declarações:

Delphi:
```
function RecordFile(Port: Smallint; const File_,
                    TermDigits: WideString):
                    Smallint;
```
Visual Basic:
```
Function RecordFile(Port As Integer, File As String,
                    TermDigits As String) As Integer
```

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.

Parâmetros:

Port – Indica o canal da Placa.
File – String contendo o nome e caminho completo do arquivo SIG que será gravado.
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:

Retorna zero se foi *iniciado* com sucesso ou 1 no caso de erro.

Veja Também: OnRecordStart, OnRecordStop, StopRecordFile

9.3.62 RecordPause

Interrompe temporariamente uma gravação, permitindo sua continuação no mesmo arquivo

Declarações:

Delphi:
```
function RecordPause(Port: Smallint;Paused: boolean):
Smallint;
```

Visual Basic:
```
Function RecordFile(Port As Integer, Paused as Boolean)
As Integer
```

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 – *True* coloca em pausa e *False* retira.

9.3.63 SetAnswerSensitivity

Permite aumentar ou diminuir a sensibilidade da detecção de atendimento.

Declarações:

Delphi:
```
procedure SetAnswerSensitivity(Factor: smallint);
```

Visual Basic:
```
Function SetAnswerSensitivity(Factor as integer)
```

Descrição:

Este método configura o nível de sensibilidade da detecção de atendimento. Deve ser chamado no inicio da aplicação e afeta todos os canais ao mesmo tempo.

Parâmetros:

Factor – Fator de sensibilidade que varia de 1 (menos sensível) até 10 (mais sensível). No inicio, o valor padrão é 6

Retorno:

Retorna Zero se executado com sucesso.

9.3.64 SetAnswerThreshold

Permite alterar o limiar para deteção de atendimento.

Declarações:

Delphi:
```
procedure SetAnswerThreshold(Port:smallint, Threshold:
smallint);
```

Visual Basic:
```
Function SetAnswerThreshold(Port as integer, Threshlod
as integer);
```

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 dificuldada pelas condições da rede pública.

Parâmetros:

Port - Canal

Threshold - Pode variar de 1 a 30. O valor padrao 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.

9.3.65 SetCallAfterAnswer

Configura as opções após a deteção de atendimento do método MakeCall

Declarações:

Delphi:
```
procedure SetCallAfterAnswer(Port: Smallint; FileName:
string;
                                Pause: SmallInt;
```

```
AutoHangUp: Boolean);
```

Visual Basic:

```
Function SetCallAfterAnswer(Port As Integer,Filename as
                            String, Pause as Integer,
                            AutoHangUp as Boolean) As
                            Integer
```

Descrição:

Este método configura uma frase para ser reproduzida após a deteção 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:

Retorna Zero se executado com sucesso.

9.3.66 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:

Retorna Zero se executado com sucesso.

9.3.67 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:

Retorna Zero se executado com sucesso.

9.3.68 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 – Digito após o flash (se existir)

PauseAfter – Pausa após o flash

Retorno:

Retorna Zero se executado com sucesso.

9.3.69 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 milisegundos.

Retorno:

Retorna Zero se executado com sucesso.

9.3.70 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:

Retorna Zero se executado com sucesso.

9.3.71 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 – Digito após o flash (se existir)
PauseAfter – Pausa após o flash

Retorno:

Retorna Zero se executado com sucesso.

9.3.72 SetCallNoAnswerRingCount

Configura a quantidade de rings a serem considerados como não atendimento

Declarações:

Delphi:
```
procedure SetCallNoAnswerRingCount(Port: Smallint;
                          RingCount:smallint): smallint;
```

Visual Basic:
```
Function SetCallNoAnswerRingCount (Port As Integer,
```

```
RingCount
                                                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.
RingCount – Númerod de rings.

Retorno:

Retorna Zero se executado com sucesso.

9.3.73 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:

Retorna Zero se executado com sucesso.

9.3.74 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. È utilizado quando o método MakeCall é chamado com o formato *ctWithFlash*.

Parâmetros:

Port – Indica o canal da Placa.

Count – Número de flashes
Digit – Digito após o flash (se existir)
PauseAfterFlash – Pausa após o flash
PauseAfterDigit - Pausa após o dígito

Retorno:

Retorna Zero se executado com sucesso.

9.3.75 SetCallWaitForDialTone

Indica se o tom de discagem deverá ser aguardado.

Declarações:

Delphi:
```
procedure SetCallWaitForDialTone(Port: Smallint;
                          WaitForDialTone: boolean);
```

Visual Basic:
```
Function SetCallWaitForDialTone(Port As Integer,
           WaitForDialTone as boolean) as integer
```

Descrição:

Este método indica se o método MakeCall deverá esperar pelo tom de discagem antes de discar. Se não receber este tom, gera o evento OnAfterMakeCall indicando a situação.

Parâmetros:

Port – Indica o canal da Placa.
WaitForDialTone – True/False

Retorno:

Retorna Zero se executado com sucesso.

9.3.76 SetDetectionType

Determina o tipo de sinalização que a placa deverá interpretar para fins de identificação de assinante A ou tons de opções de menu em cada canal.

Declarações:

Delphi:
```
procedure SetDetectionType(Port: Smallint;
                                DetectionType:
SmallInt);
```

Visual Basic:
```
Function SetDetectionType (Port As Integer,
                      DetectionType as Integer) As
Integer
```

Descrição:

As companhias telefônicas sinalização a Identificação do Assinante "A" (quem está discando) através de dois tipos de sinal: DTMF e MFP. Esta sinalização ocorre antes do primeiro Ring.

No caso de deteção dos tons do telefone para um atendedor com menu, utiliza-se a deteção DTMF.

9.3.77 SetDTMFAttenuatingHigh

Indicar o fator de atenuação para a freqüência alta na geração de tons DTMF.

Declarações:

Delphi:
```
procedure SetDTMFAttenuatingHigh(Value: integer);
```

Visual Basic:
```
Function SetDTMFAttenuatingHigh(Value as Integer) As
Integer
```

Descrição:

Este método pode receber valores entre 0 e 60 dB (*decibéis*) para atenuação da freqüencia alta. 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:

Duration – Duração em milissegundos

Valor de Retorno:

Retorna zero se for executada com sucesso.

9.3.78 SetDTMFAttenuatingLow

Indicar o fator de atenuação para a freqüência baixa na geração de tons DTMF.

Declarações:

Delphi:
```
procedure SetDTMFAttenuatingLow(Value: integer);
```

Visual Basic:
```
Function SetDTMFAttenuatingLow(Value as Integer) As
Integer
```

Descrição:

Esta propriedade pode receber valores entre 0 e 60 dB (*decibéis*) para atenuação da freqüencia baixa. 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:

Duration – Duração em milissegundos

Valor de Retorno:

Retorna zero se for executada com sucesso.

9.3.79 SetDTMFDuration

Determina a duração do tom DTMF gerado pela biblioteca

Declarações:

Delphi:
```
procedure SetDTMFDuration(Duration: integer);
```

Visual Basic:
```
Function SetDTMFDuration(Duration as Integer) As
Integer
```

Descrição:

O tom tem a duração de 100ms como padrão mas poderá ser alterado conforme a necessidade. Quando um novo valor é atribuído, todos os canais de todas as placas são afetados.

Parâmetros:

Duration – Duração em milissegundos

Valor de Retorno:

Retorna zero se for executada com sucesso.

9.3.80 SetDTMFPause

Determina a duração, em milisegundos, da pausa interdigital em uma discagem por tom.

Declarações:

Delphi:
```
procedure SetDTMFPause(Duration: integer);
```

Visual Basic:
```
Function SetDTMFPause(Duration as Integer) As Integer
```

Descrição:

Se o metodo Dial tiver mais de um dígito como parâmetro, uma pausa entre os dígitos acontece, respeitando o tempo determinado por esta propriedade.

Quando um novo valor é atribuído, todos os canais de todas as placas são afetados.

Parâmetros:

Duration – Duração em milissegundos

Valor de Retorno:

Retorna zero se for executada com sucesso.

9.3.81 SetDTMFTwist

Permite alterar fatores de sensibilidade para a detecção de dígitos.

Declarações:

Delphi:
```
procedure SetDTMFTwist(Twist1: integer; Twist2:
integer);
```

Visual Basic:
```
Function SetDTMFTwist(Twist1 as Integer, Twist2 as
integer) As Integer
```

Descrição:

Este método deve ser utilizado somente quando for constatado que em situações normais de operação, a detecção de DTMF esteja, ou muito sensível ou pouco sensível.

O padrão DTMF é composto de um par de freqüências (tons) para cada dígito discado que chamaremos respectivamente de F1 e F2.

Devido às características da linha telefônica e dos equipamentos a elas ligados, as amplitudes destes tons podem ter atenuações diferentes na propagação pela rede telefônica.

A presença de áudio juntamente com o par de tons do DTMF (quando a placa está falando uma mensagem) pode prejudicar sua detecção pois gera novos tons F3, junto com F1 e F2.

Parâmetros:

O parâmetro **TWIST1** diz respeito à tolerância na diferença de amplitude entre as duas freqüências do DTMF – F1 e F2. Quanto **maior** o valor de TWIST, maior a tolerância a

variações de amplitude entre as duas freqüências ou mais fácil será a detecção de digitos *(podendo até chegar em situações de talk-off)*. Quando necessário, o ajuste desta variável deve ser feito sem que a placa esteja falando. O valor padrão é 10, podendo variar de 0 a 99.

O parâmetro **TWIST2** diz respeito à seletividade de uma terceira freqüência F3 com relação às duas freqüências, F1 e F2 presentes no áudio recebido pela placa, isto quer dizer que quanto **menor** o valor de TWIST2, **maiores** poderão ser as amplitudes das outras freqüências presentes no áudio sem que seja rejeitada a detecção de DTMF, ou seja, os valores menores aumentam a sensibilidade de DTMF podendo chegar até situações de talk-off. O valor padrão é 0, podendo variar de 0 a 99.

Os dois valores deverão ser ajustados empiricamente em situações que os dígitos não estejam sendo detectados a contento ou, no outro extremo, em situações de *talk-off*.

Valor de Retorno:

Retorna zero se for executada com sucesso.

9.3.82 SetFirstFaxFrequency

Modificar a freqüência a ser detectada em caso de fax.

Declarações:

Delphi:
```
function SetFirstFaxFrequency(Frequency: Smallint):
Smallint;
```

Visual Basic:
```
Function SetFirstFaxFrequency(Frequency As Integer) As
Integer
```

Descrição:

Esta frequencia deve ser alterada caso seja necessário detectar alguma frequencia fora de padrão. Neste caso o fax não será mais detectado.

Parâmetros:

Frequency – Valor da freqüência. O padrão é 1100Hz.

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.83 SetFrequency

Modificar a freqüência padrão de supervisão

Declarações:

Delphi:
```
function SetFrequency(Frequency: Smallint): Smallint;
```

Visual Basic:

```
Function SetFrequency(Frequency As Integer) As Integer
```

Descrição:

A freqüência base para detecção dos tons de chamada, ocupado, etc... é 425hz. Algumas centrais trabalham com valores diferentes do padrão.

Alterando esta freqüência as placas Digivoice passarão a monitorar outra faixa de tons. Ao atribuir este valor, todos os canais de todas as placas são afetados.

Parâmetros:

Frequency – Valor da freqüência. O padrão é 400hz.

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.84 SetFrequencyTime

Altera o tempo mínimo para detecção do tom de discagem.

Declarações:

Delphi:
```
function SetFrequencyTime(Duration: Smallint):
Smallint;
```

Visual Basic:
```
Function SetFrequencyTime(Duration As Integer) As
Integer
```

Descrição:

Especifica o tempo de mínimo de duração que o tom deverá ocorrer para que a placa possa detectá-lo.
O valor padrão é 1500ms, tempo normalmente utilizado para centrais públicas e privadas, por isso só deverá ser alterado em casos de sinalizações específicas do PABX.

Parâmetros:

Duration – Tempo de duração. O padrão é 1500ms.

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.85 SetImpedance

Atribui um valor de impedância para as placas VoicerPhone ISA/PCI

Declarações:

Delphi:
```
function SetImpedance(Impedance: Smallint): smallint;
```

Visual Basic:
```
Function SetImpedance(Impedance As Integer) As Integer
```

Descrição:

As centrais PABX trabalham com impedância de 600W ou 900W, dependendo do fabricante e modelo. As placas VoicerPhone deverão utilizar a mesma impedância do PABX para melhor qualidade de áudio.

Quando o método é chamado, todas as placas ISA habilitadas serão afetadas. Não tem efeito algum sobre as placas VoicerBox PCI

Parâmetros:

Impedance - Permite os valores **it600** ou **it900**.

Valor de Retorno:

Retorna zero se for executada com sucesso.

9.3.86 SetPlayFormat

Especifica o formato de reprodução de um canal independente.

Declarações:

Delphi:

```
function SetPlayFormat(Port: Smallint; FileFormat:
Smallint):
                                                    Sma
                                                    lli
                                                    nt;
```

Visual Basic:

```
Function SetPlayFormat(Port As Integer, FileFormat as
Integer)
                                                As
Integer
```

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 audio 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 agora 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 - ffWave
1 - ffSig
2 - ffWavePCM
3 - ffGsm610

Valores de Retorno:

0 - Executado com sucesso
1 - Parâmetro inválido.
2 - Não altera o formato caso o canal esteja reproduzindo uma mensagem.

Veja Também: SetRecordFormat.

9.3.87 SetRecordFormat

Especifica o formato de gravação de um canal independente.

Declarações:

Delphi:

```
function SetRecordFormat(Port: Smallint; FileFormat:
Smallint):
                                                    Sma
                                                    lli
                                                    nt;
```

Visual Basic:

```
Function SetRecordFormat(Port As Integer, FileFormat as
Integer)
                                                As
Integer
```

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 audio 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.

Parâmetros:

Port – Indica o canal da Placa.
FileFormat – Formato para gravação.
0 - ffWave
1 - ffSig
2 - ffWavePCM
3 - ffGsm610

Valores de Retorno:

0 - Executado com sucesso
1 - Parâmetro inválido.
2 - Não altera o formato caso o canal esteja gravando uma mensagem.

Veja Também: SetPlayFormat.

9.3.88 SetRecordGain

Especifica o ganho para a gravação.

Declarações:

Delphi:

```
function SetRecordGain(Port: Smallint; Ganho:
Smallint):
                                                    Sma
                                                    lli
                                                    nt;
```

Visual Basic:

```
Function SetRecordFormat(Port As Integer, Ganho as
Integer)
                                                As
Integer
```

Descrição:

Especifica o ganho para a gravação de um canal independente. O ganho pode variar de 1 a 10 e o padrão é 4. Caso o valor especificado seja menor que 1, o valor assumido será 1 e caso o valor especificado seja maior que 10 o valor assumido será 10.

Parâmetros:

Port – Indica o canal da Placa.
Ganho – Valores de 1 a 10, sendo o padrao 4

Valores de Retorno:

0 - Executado com sucesso

9.3.89 SetSecondFaxFrequency

Modificar a freqüência a ser detectada em caso de fax.

Declarações:

Delphi:
```
function SetSecondFaxFrequency(Frequency: Smallint):
Smallint;
```

Visual Basic:
```
Function SetSecondFaxFrequency(Frequency As Integer) As
Integer
```

Descrição:

Esta frequencia deve ser alterada caso seja necessário detectar alguma frequencia fora de padrão. Neste caso o fax não será mais detectado.

Parâmetros:

Frequency – Valor da freqüência. O padrão é 2100Hz.

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.90 SetToneTwist

Permite alterar fatores de sensibilidade para a detecção de tons.

Declarações:

Delphi:
```
procedure SetToneTwist(Twist: integer);
```

Visual Basic:
```
Function SetToneTwist(Twist as Integer) As Integer
```

Descrição:

Este método deve ser utilizado somente quando for constatado que em situações normais de operação, a detecção de tons esteja, ou muito sensível ou pouco sensível, principalmente em situações que o tom é gerado em cima da voz.

Parâmetro:

TWIST:

Na detecção de tons 425Hz e FAX a variável TWIST3 diz respeito a rejeição de da freqüência do tom F1 a um segundo tom presente no áudio. Quanto maior a variável TWIST3 maior será a rejeição. Em outras palavras, quanto **maior** o valor deste parâmetro, **menor** será a sensibilidade de detecção de tons em cima da voz. O valor padrão é 5 e poderá variar de 0 a 99.

Valor de Retorno:

Retorna zero se for executada com sucesso.

9.3.91 SetVolume

Modificar o volume do headset da placa VoicerPhone (ISA/PCI).

Declarações:

Delphi:
```
function SetVolume(Volume: Smallint): Smallint;
```

Visual Basic:
```
Function SetVolume(Volume As Integer) As Integer
```

Descrição:

A placa VoicerPhone permite três níveis de ganho (volume). Utilize esta propriedade para aumentar ou diminuir o ganho. Este volume pode ser alterado o qualquer momento durante a operação do sistema. O volume pode assimir os valores **voOne**, **voTwo** e **voThree** representando 3 níveis de ganho possíveis.

Parâmetros:

Volume – O ganho a ser atribuído (**voOne**, **voTwo** e **voThree)**

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

9.3.92 ShutdownVoicerLib

Finaliza a comunicação da placa com a aplicação.

Declarações:

Delphi:
```
function ShutdownVoicerLib: Integer;
```

Visual Basic:
```
Function ShutdownVoicerLib() As Long
```

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. Então, NUNCA FECHE A APLICAÇÃO ANTES DE CHAMAR ESTE MÉTODO.

Valores de Retorno:

Retorna 99 se executado com sucesso e diferente disso em caso de erro.

Veja Também: StartVoicerLib

9.3.93 Sig2Wave

Converte do formato SIG para o formato Wave.

Declarações:

Delphi:
```
function Sig2Wave(const Source:WideString;const
Destination:WideString; Gain:Double):Smallint;
```

Visual Basic:
```
Function Sig2Wave(Source As String, Destination As
String,
Gain as Double) As Integer
```

Descrição:

A placa VoicerPhone trabalha com arquivos no formato SIG (Lei ***μ*** - 8Khz). Para poder reproduzir um arquivo gerado pela placa em uma placa de som, é necessário converter o arquivo para um formato Wave conhecido. O método Sig2Wave faz esta conversão, bastando passar o nome do arquivo de origem, do arquivo de destino e o ganho em relação à gravação original em sig. O ganho pode ser 0 indicando que não há ganho.

Parâmetros:

Source – String contendo o nome e caminho completo do arquivo SIG que será convertido.
File – String contendo o nome e caminho completo do arquivo WAVE que será gravado.
Gain - Fator de ganho sobre o arquivo original em SIG. Pode ser utilizado valores maiores que zero para melhorar gravações baixas.

Valores de Retorno:

0 - Executado com sucesso
1 – Não foi encontrado o arquivo de origem (Source)
2 – Não foi possível criar o arquivo de destino (Destination)
3 - Erro indeterminado

Veja Também: Wave2Sig

Obs.: As versões atuais trabalham também com formato nativo Wave (8bits, 8khz, mono) portanto dê preferência a trabalhar sempre com wave já que tem o mesmo tamanho do SIG.

9.3.94 StartVoicerLib

Inicializa a comunicação da placa com a aplicação.

Declarações:

Delphi:
```
function StartVoicerLib: Integer;
```

Visual Basic:
```
Function StartVoicerLib() As Long
```

Descrição:

Este método faz com que se inicie a comunicação da placa com a aplicação. Logo, a sua chamada deverá ser a primeira coisa que o programador deverá fazer na aplicação, pois até então, a placa estará totalmente "morta", e nada irá funcionar. As placas ISA deverão ser configuradas previamente através do programa utilitário GeraINI.

Valores de Retorno:

99 - executado com sucesso
0 - Erro Genérico – Provavemente causado por falha na instalação. Recomenda-se desinstalar e repetir o processo de instalação.
1 - Driver já está inicializado
2 - Arquivo de configuracao inexistente (somente para placas ISA)
3 - Placas PCI nao foram encontradas
4 - Numero de Placas ISA invalido (arquivo de configuração com dados incorretos)
5 - Irq de placa ISA inválido (arquivo de configuração com dados incorretos)
6 - IO de placa ISA invalido (arquivo de configuração com dados incorretos)
7 - Erro de conferência - provavelmente um conflito de endereços de I/O
8 - Falta do arquivo KPVPISA.VXD (Win9x) ou KPVPISA.SYS (NT/2000)
9 - Erro na abertura dos recursos da placa
10 - Erro na carga do programa - provavelmente um conflito de endereços de I/O

Veja Também: ShutdownVoicerLib

9.3.95 StopPlayFile

Interrompe a reprodução de um arquivo SIG.

Declarações:

Delphi:
```
function StopPlayFile(Port: Smallint): Smallint;
```

Visual Basic:
```
Function StopPlayFile(Port As Integer) As Integer
```

Descrição:

Este método interrompe a reprodução de um arquivo SIG. Neste caso, o evento OnPlayStop é gerado e o parâmetro Status receberá o valor ssStopped.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: PlayFile

9.3.96 StopRecordFile

Interrompe a gravação de um arquivo SIG.

Declarações:

Delphi:
```
function StopRecordFile(Port: Smallint): Smallint;
```

Visual Basic:
```
Function StopRecordFile(Port As Integer) As Integer
```

Descrição:

Este método interrompe a gravação de um arquivo SIG. Neste caso, o evento OnRecordStop é gerado e o parâmetro Status receberá o valor ssStopped.

Parâmetros:

Port – Indica o canal da Placa

Valor de Retorno:

Retorna zero se foi executado com sucesso ou 1 no caso de erro.

Veja Também: RecordFile

9.3.97 Wave2Sig

Converte do formato Wave para o formato SIG.

Declarações:

Delphi:
```
function Wave2Sig(const Source, Destination:
WideString):
                                                    Sma
                                                    lli
                                                    nt;
```

Visual Basic:

```
Function Wave2Sig(Source As String, Destination As
String)
                                                    As
                                                    Int
                                                    ege
                                                    r
```

Descrição:

A placa VoicerPhone trabalha com arquivos no formato SIG (Lei µ - 8Khz). Para poder reproduzir um arquivo gerado por um placa de som (tipo SoundBlaster), é necessário converter o arquivo WAVE para o formato SIG reconhecido. O método Wave2Sig faz esta conversão, bastando passar o nome do arquivo de origem e do arquivo de destino

Parâmetros:

Source – String contendo o nome e caminho completo do arquivo WAVE que será convertido.
File – String contendo o nome e caminho completo do arquivo SIG que será gravado.

Valores de Retorno:

- **0** - Executado com sucesso
- **1** - Não foi encontrado o arquivo de origem (Source)
- **2** - Não foi possível criar o arquivo de destino (Destination)
- **3** - Erro indeterminado

Veja Também: Sig2Wave.

O arquivo Wave deverá ser 8khz, 16bits, Mono para que a conversão seja possível

9.3.98 WaveToGsm

Converte do formato Wave para o formato GSM.

Declarações:

Delphi:

```
function WaveToGsm(const Source:WideString; const
Destination: WideString):
                                                    Sma
                                                    lli
                                                    nt;
```

Visual Basic:

```
Function WaveToGsm(Source As String, Destination As
String)
                                                    As
                                                    Int
                                                    ege
                                                    r
```

Parâmetros:

Source – String contendo o nome e caminho completo do arquivo GSM que será convertido.
Destination – String contendo o nome e caminho completo do arquivo Wave que será gravado.

Valores de Retorno:

0 - Executado com sucesso
1 - Não foi encontrado o arquivo de origem (Source)
2 - Não foi possível criar o arquivo de destino (Destination)
3 - Erro indeterminado na conversão

Veja Também: GsmToWave.

O arquivo Wave deverá ser 8khz,

16bits, Mono para que a conversão seja possível

9.3.99 WriteSecurityWord

Permite gravar string de segurança na memória da placa.

Declarações:

Delphi:
```
function WriteSecurityWord(Card: smallint;
                                Data,
Param:String):smallint;
```

Visual Basic:
```
Function WriteSecurityWord(Card as Integer, Data As
String
                                   Param as String) as
Integer
```

Descrição:

A função WriteSecurityWord permite gravar na memória da placa até 10 caracteres. É sempre gravado do primeiro endereço em diante, ou seja, o programador não tem acesso a um endereço específico de memória. Caso seja enviado mais de 10 caracteres, a string será truncada e o restante será descartado. ***OBS.: Esta função só está disponível em placas equipadas com memória EEPROM.***

Parâmetros:

Card – A **placa** que deverá ser gravada. De 1 a ***n***.
Data - O dado a ser gravado
Param - Uso reservado, passar sempre nulo ("")

Valores de Retorno:

0 – OK
1 - Erro de I/O
2 - String maior que 10 caracteres
3 - Driver desabilitado - Necessita executar StartVoicerLib

Índice Remissivo

- N -

- O -

- P -

- R -

- S -

- V -