Voice Software Reference: Features Guide

for Windows

05-1457-001

COPYRIGHT NOTICE

Publication Date: March, 2000

Part Number: 05-1457-001

Dialogic, an Intel Company
1515 Route 10
Parsippany NJ 07054
U.S.A.

For **Technical Support**, visit the Dialogic support website at:
http://support.dialogic.com

For **Sales Offices** and other contact information, visit the main Dialogic website at:
http://www.dialogic.com

OPERATING SYSTEM SUPPORT

The term *Windows* refers to both the Windows NT® and Windows® 2000 operating systems. For a complete list of supported Windows operating systems, refer to the *Release Guide* that came with your Dialogic System Release for Windows, or to the Dialogic support site at *http://support.dialogic.com/releases*.

Table of Contents

List of Tables

List of Figures

1. Introduction to Voice Features

1.1. Overview

This chapter provides a brief description of the major voice software features for Windows. The remainder of this *Voice Features Guide* is divided into chapters based on each major feature. The major features of the voice software for Windows are:

- Call Analysis
- Caller ID
- Global Dial Pulse Detection (GDPD)
- Global Tone Detection and Global Tone Generation; Cadence Tone Generation
- R2MF Signaling
- Speed and Volume Control
- Echo Cancellation Resource
- Analog Display Services Interface (ADSI)
- Syntellect Patent Protection: Automated Attendant
- Transaction Record
- PerfectCall Disconnect Tone Supervision
- Demonstration Programs

1.2. Call Analysis Features

Call Analysis is used to monitor the progress of a call after dialing into the Public Switched Telephone Network (PSTN). There are two forms of Call Analysis: Basic Call Analysis and PerfectCall Call Analysis. PerfectCall Call Analysis uses an improved method of signal identification and can detect fax machines and answering machines. Basic Call Analysis provides backward compatibility for older applications written before PerfectCall Analysis became available.

Call Analysis is initiated using the **dx_dial( )** function which uses input from the Call Analysis Parameter (DX_CAP) data structure.

Call Analysis is available on all voice boards.

1.2.1. Call Progress Features

The intelligent network interface boards support a combination of the following call progress features:

- PerfectCall (call progress analysis)
- Positive Voice Detection (PVD)
- Positive Answering Machine Detection (PAMD)
- Silence Detection

1.2.2. Tone Features

The intelligent network interface boards support a combination of the following tone features:

- PerfectDigit (DTMF signaling and MF signaling)
- Global Tone Detection (GTD)
- Global Tone Generation (GTG)
- Compelled Tone Protocol (R2MF)
- Dial Pulse Detection

1.2.3. PerfectCall Disconnect Tone Supervision

PerfectCall provides positive disconnect supervision by detecting either a loop current drop or the disconnect tone that occurs after a party hangs up to end a connected call. In both cases, when a disconnect is detected, a loop current drop event is generated. In this way, disconnect tones can be used easily in an application that processes loop current drop events.

Disconnect supervision can be enabled using the Advanced Tone Features utility.

When enabled, the default disconnect tone definition is used. The default tone definition is as follows:

Dual Tone with cadence:

- frequency 1 = 500 ± 200 Hz.
- frequency 2 = 525 ± 175 Hz.
- cadence on-time = 550 ± 400 ms.

- cadence off-time = 550 ± 400 ms.

1.2.4. Using PBXpert Tone Set Files with PerfectCall

PBXpert is a utility program designed to facilitate the management of unique call progress tones produced by PBXs, key systems (KSU), and PSTNs. A Tone Set File, or TSF, is the way in which PBXpert stores call progress tone definitions.

The Advanced Tone Features utility allows you to select and activate a TSF that will be used by your Dialogic voice board for all progress analysis. When a TSF is active, its tone definitions replace the default tone definitions already in the Dialogic voice DLL, where they are automatically used by any applications using PerfectCall.

The **dx_TSFstatus( )** function enables your application to determine the outcome when activating a TSF.

1.3. Signaling Features

1.3.1. Global Tone Detection (GTD)

Global Tone Detection allows a voice board to detect single or dual frequency tones other than DTMF tones 0-9, a-d, *, and #. Through GTD, a user can define the characteristics of a tone in order to detect a tone with the same characteristics. The characteristics of a tone can be defined and tone detection can be enabled using GTD functions provided in the Voice Library.

Global Tone Detection (GTD) is available on all Dialogic voice boards.

See *Chapter5.2. Global Tone Detection (GTD)* for detailed information about Global Tone Detection.

1.3.2. Global Tone Generation (GTG)

Global Tone Generation permits the creation of user-defined tones using the TN_GEN template data structure and allows the user to play the tone using the

dx_playtone( ) function. The **dx_bldtngen( )** can be used to build the TN_GEN template.

Global Tone Generation (GTG) is available on all Dialogic voice boards.

See *Section 5.3. Global Tone Generation (GTG)* for detailed information about Global Tone Generation.

1.3.3. Cadence Tone Generation (CTG)

CTG is an enhancement to Global Tone Generation. It allows you to generate a tone with up to 4 single- or dual-tone elements, each with its own on/off duration, which creates the tone pattern or cadence. The data structure TN_GENCAD defines a cadenced tone and the function **dx_playtoneEx( )** plays the cadenced tone defined by TN_GENCAD. You can define your own custom cadenced tone or take advantage of the built-in set of standard PBX Call Progress Signals, such as dial tone, ringback, busy, etc.

1.3.4. Global Dial Pulse Detection (GDPD)

Global Dial Pulse Detection (GDPD) allows applications to detect dial pulses from rotary or pulse phones and use them as if they were DTMF digits. This software-based dial pulse detection method uses use country-specific parameters for extremely accurate performance. Unlike other DPD solutions on the market, GDPD does not require callers to train the detection algorithm by dialing zero before any other dialing response.

GDPD supports both digital and analog (using the loop start telephone interface) applications on selected boards.

Global DPD will work only on DPD-enabled boards. Boards with the "IDPD" suffix are enabled. If your board does not have an IDPD suffix, you must order a separate Global DPD enablement package from Dialogic. GDPD is not supported on the VFX/40, VFX/40E, VFX/40SC or VFX/40ESC boards. It is recommended that you retain the serial number of the DPD-enabled board for your records.

1.3.5. R2MF Signaling

R2MF signaling is an international signaling system that is used in Europe and Asia to permit the transmission of numerical and other information relating to the called and calling subscriber's lines.

1.4. Play and Record Features

1.4.1. Change Length Of Built-In Beep Tone

The **dx_settonelen( )** function can be used to modify the duration of the built-in beep tone that some application programs use to indicate the start of a recording (sometimes referred to as the pre-record beep) or a playback.

1.4.2. Speed and Volume Control

The voice software speed and volume support contains functions and data structures to control the speed and volume of play on a channel. For example, an end user may control the speed or volume of a message by entering a previously established DTMF tone.

NOTE: Speed can be controlled on playbacks using 24 kbps or 32 kbps ADPCM only. Volume can be controlled on all playbacks regardless of the encoding algorithm.

Speed and Volume control are available on D/21D, D/21E, D/41D, D/41E, D/41ESC, D/81A, D/121B, D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1 and D/320SC boards only. Do not use the Speed and Volume control functions to control speed on the D/120, D/121 or D/121A boards.

1.4.3. Transaction Record

Transaction Record enables the recording of a two-party conversation by allowing two SCbus time slots from a single channel to be recorded. This feature is useful for Call Center applications where it is necessary to archive a verbal transaction or record a live conversation. A live conversation requires two timeslots on the SCbus. Dialogic voice boards today can only record one timeslot at a time. No loss of channel density is realized. A D/160SC-LS can still record 16 simultaneous conversations. Voice activity on two channels can be summed and stored in a single file, device and/or memory.

The transaction record feature is compatible with the following Dialogic hardware: D/80SC, D/160SC, D/240SC, D/320SC, D/160SC-LS, D/240SC-T1, D/300SC-E1, D/240SC-2T1, D/300SC2E1, D/480SC-2T1, D/600SC-2E1.

The function **dx_mreciottdata( )** was added to the Dialogic Voice Library to implement the transaction record feature. It is an extension of the **dx_reciottdata( )** function.

1.4.4. Offset Playback of WAVE Files

The **dx_playiottdata( )** function can now be used to play WAVE files using offsets specified in the DX_IOTT data structure.

1.4.5. Silence Compressed Record (SCR)

SCR enables recording of a caller's message with silent pauses eliminated. This results in smaller size voice files with no loss of intelligibility. Parameters in the *voice.prm* file are used to control this feature.

1.4.6. Echo Cancellation Resource (ECR)

The Echo Cancellation Resource (ECR) feature is a voice channel mode that reduces the echo component on an external SCbus time slot signal. ECR provides the following features to an application:

- Adds the ability to utilize the echo cancellation function on signals external to the voice channels
- Enables the use of the voice channel to perform echo cancellation on non-voice channel signals
- Improves voice quality during full-duplex conversations
- Provides the ability to modify the size of the voice board firmware buffers to reduce latency for Internet telephony applications
- Provides the option to modify the characteristics of the echo canceller, such as enabling or disabling non-linear processing (NLP) during echo cancellation

1.4.7. GSM and G.726 Voice Coders

The voice software supports the GSM and G.726 voice coders through the I/O functions that use a **DX_XPB** data structure. For more detailed information, refer to the DX_XPB Data Structure in the *Voice Programmer's Guide*.

- WAVE GSM 6.10 full-rate voice coder
- G.726 bit-exact voice coder

1.5. Caller ID

Caller Identification (Caller ID) is a service provided by local telephone companies enabling the subscriber to receive the caller's phone number (Directory Number) and other information about the call. The Caller ID information is transmitted using Frequency Shift Keying (FSK) to the subscriber from the service provider, the telephone company Central Office (CO), at 1200 baud. Caller ID is available on selected boards.

1.6. SCbus Routing

The SCbus is a real-time, high speed, time division multiplexed (TDM) communications bus connecting Signal Computing System Architecture (SCSA), voice, telephone network interface and other technology resource boards together. SCbus boards are treated as board devices with on-board voice and/or telephone network interface devices that are identified by a board and channel (time slot for digital network channels) designation, such as a voice channel, analog channel, or digital channel.

For more information on the SCbus and SCbus routing, refer to the *SCbus Routing Guide* and the *SCbus Routing Software Reference*.

1.7. Syntellect™ Patent Protection: Automated Attendant

As a result of the Dialogic patent license agreement with Syntellect Technology Corporation (STC), you may purchase products that are licensed for specific telephony patents held by Syntellect directly from Dialogic. These patents cover a range of common functions used in computer telephony such as automated attendant, automated access and call processing to facilitate call completions.

One way to protect against potential patent infringement is by purchasing specific Dialogic hardware and software that include a license for the Syntellect Technology Corporation patent portfolio. These products have a part number designation of "STC." Any Dialogic product that does not contain the "STC" designation in its part number is not licensed under the STC patent portfolio.

For additional information on the Dialogic and Syntellect patent license agreement refer to the topic "Company" and "Press Room" at the Dialogic World Wide Web site at *http://support.dialogic.com*

Location on Dialogic Web Site http://www.dialogic.com/company	**Topic**
/syntel/over.htm	Overview of Syntellect/Dialogic Patent Agreement (includes information on products supported)

Location on Dialogic Web Site http://www.dialogic.com/company	Topic
/syntel/patents.htm	Syntellect Technology Corporation. U.S. Patents
/syntel/intellct.htm	Computer Telephony Intellectual Property Rights
/pressroom/pressrel/412web.htm	Dialogic and Syntellect Announce Patent License Agreement

The Syntellect software included on the *Dialogic System Release Software and SDK for Windows* is designed to be incorporated into any type of application. If your application requires patented Syntellect technology, you can use the API function calls in the Syntellect software to assure that licensed STC-enabled hardware is in the system, and if so, you may implement the patented functions.

The Syntellect hardware and software package offers a superset of features not available on non-STC boards. They include:

- A new library of API function calls.
- A sample automated attendant application that can be integrated in your voice processing application. The automated attendant:
  - checks for an incoming call
  - answers the call and plays a voice file
  - receives digit input and transfers the call to the proper extension
- The source code and demonstration code for the automated attendant application.

The following Dialogic boards support the Syntellect software included with the Dialogic System Software and SDK for Windows. Note that these boards have part numbers designated STC.

- ProLine/2V-STC
- DIALOG/4-STC
- D/21D-STC
- D/41D-STC
- D/21D-DE-STC
- D/41D-DE-STC

- D/21H-STC
- D/41H-STC
- D/41ESC-STC
- D/41ESC-DE-STC
- VFX/40SC-STC
- VFX/40ESC-STC
- VFX/40ESCplus-STC
- D/42XX-STC
- DVM/400-STC
- D/240SC-T1-STC
- D/160SC-LS-STC

1.8. Analog Display Services Interface (ADSI)

Analog Display Services Interface (ADSI) is a Bellcore standard that defines a protocol used to transmit data to display-based, ADSI-compliant telephones.

For many years, Dialogic provided one-way ADSI support through the **dx_play( )** and **dx_playf( )** functions. This ADSI support enabled developers to use Dialogic boards to make ADSI servers that work with ADSI phones and to support ADSI features such as visual Voice Mail. This is referred to as the "older" implementation of one-way ADSI (for details on this older method, see the **dx_play( )** function in the *Voice Software Reference: Programmer's Guide*).

The newer implementation of ADSI offers several enhancements and is supported through the **dx_RxIottData( )**, **dx_TxIottData( )**, and **dx_TxRxIottData( )** functions. This implementation is referred to simply as "**ADSI Support**" or "**Two-Way ADSI**" and includes the following features:

- Transmit to On-Hook ADSI phones: Allows messages to be sent to an ADSI phone when the phone is either On-Hook or Off-Hook.
- Two-Way Frequency Shift Keying (FSK): Allows users to send and receive character or binary data at 1200 bits/second between the server and compatible devices, such as certain ADSI phones with keyboards. The two-way FSK feature supports applications such as off-line Email editing and sending FSK Caller ID data to a customer premise equipment (CPE) device through an MSI/SC board.

This newer Dialogic ADSI support provides for both two-way and one-way ADSI transmission and is the recommended method for implementing either one-way or two-way ADSI in an application program. The older one-way ADSI support can be used but is not the recommended method. See *Section 9.5.3. Modifying Older One-Way ADSI Applications* for information on converting from the older to the newer method for using ADSI.

1.9. Demonstration Programs

A Multi-threaded Text-Based Application program and a Multi-threaded GUI-Based Application program are provided.

2. Call Analysis

2.1. Overview

The following Call Analysis features are available on all boards.

- Positive Voice Detection and Positive Answering Machine Detection
- Tri-Tone Frequency Detection (SIT tones)
- PerfectCall Analysis

Call Analysis is used to determine the progress of a call after dialing into the Public Switched Telephone Network (PSTN), where a wide variety of signal possibilities can occur.

By using Call Analysis you can determine the following:

Call Analysis determines the outcome of the call from among the following possibilities:

Intercept	A Special Information Tone (SIT) was detected; an invalid number was dialed or there was a problem completing the call. This is also known as an *operator intercept*.
No Ringback	No discernable signal pattern was detected.
Connect	The phone was answered.
No Answer	The line was ringing but was not answered.
Busy	A busy signal was detected.

The outcome of the call is returned to the application when Call Analysis has completed.

There are two forms of Call Analysis: Basic Call Analysis and PerfectCall Call Analysis. PerfectCall Call Analysis uses an improved method of signal identification, and can also detect fax machines and answering machines. Basic

Call Analysis provides backward compatibility for older applications; any application which was written before PerfectCall Analysis became available will continue to work unchanged. However, it is recommended that all new applications be designed for PerfectCall Call Analysis.

Call Analysis is initiated when a call is dialed using the **dx_dial( )** function. This function uses input from the Call Analysis Parameter structure (DX_CAP). You can adjust the DX_CAP parameters to fit the needs of your application. When the Voice Driver determines the outcome of the call, information is returned using Extended Attribute functions.

2.2. How Does Call Analysis Work?

Call Analysis uses the following techniques to determine the progress of the call:

- Cadence Detection
- Frequency Detection
- Loop Current Detection
- Positive Voice Detection and Positive Answering Machine Detection

The following figures illustrate these processes.

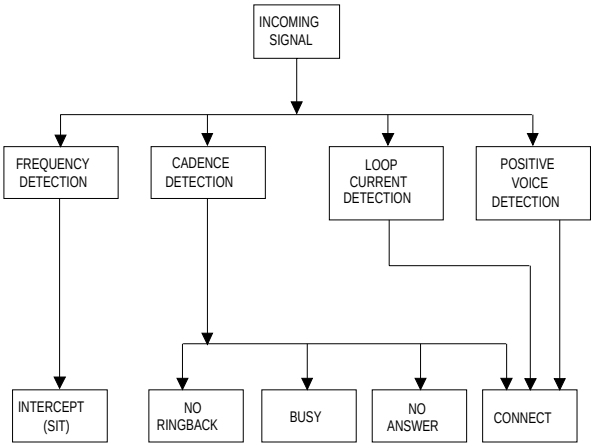


Figure 1. Basic Call Analysis Components

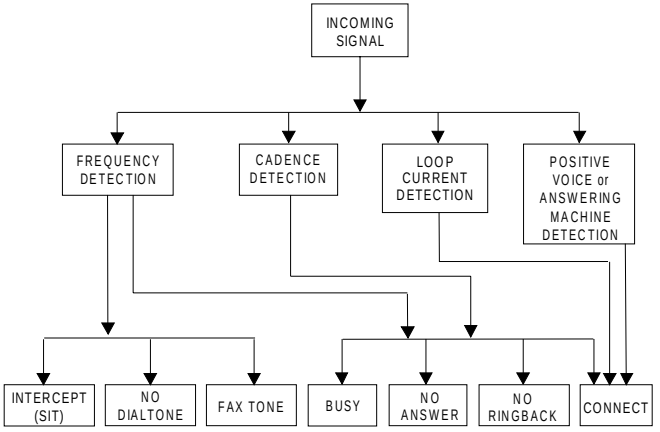


Figure 2. PerfectCall Call Analysis Components

Frequency Detection, Cadence Detection, Loop Current Detection, Positive Voice Detection, and Positive Answering Machine Detection can all operate

simultaneously during Call Analysis. See *Section 2.6. How the DX_CAP Controls Call Analysis* for more information.

The figures show that Cadence Detection is the sole means of detecting a no ringback, busy, or no answer when using Basic Call Analysis. PerfectCall Call Analysis uses Cadence Detection plus Frequency Detection to identify all of these signals plus fax machine tones. A connect can be detected through the complementary methods of Cadence Detection, Frequency Detection, Loop Current Detection, Positive Voice Detection, and Positive Answering Machine Detection (Positive Answering Machine Detection is available with PerfectCall Call Analysis only).

2.3. How to Enable PerfectCall Call Analysis

To enable PerfectCall Call Analysis on a specified channel, perform the following steps:

NOTE: This procedure needs to be followed only once per channel; thereafter, any outgoing calls made using **dx_dial( )** will benefit from PerfectCall Call Analysis.

1. Make any desired modifications to the default dial tone, busy tone, fax tone, and ringback signal definitions using the **dx_chgfreq( )**, **dx_chgdur( )**, and **dx_chgrepcnt( )** functions.
2. Execute the **dx_initcallp( )** function to activate PerfectCall Call Analysis. PerfectCall Call Analysis stays active until **dx_deltones( )** is called.

   NOTE: To disable PerfectCall Call Analysis, call the **dx_deltones( )** function.

Call **dx_deltones( )** prior to calling **dx_initcallp( )** to clear all tone templates remaining on the channel. However, **dx_deltones( )** deletes GTD tones added to the channel.

The **dx_initcallp( )** function initializes PerfectCall Call Analysis on the specified channel, using the current tone definitions for local dial tone, international dial tone, extra dial tone, two busy signals, ringback, and two fax tones. Once the channel is initialized with these tone definitions, this initialization cannot be

altered. The only way to change the tone definitions in effect for a given channel is to issue a **dx_deltones( )** call for that channel, then invoke another **dx_initcallp( )** with different tone definitions.

Refer to the Voice Software Reference: *Programmer's Guide* for more information on **dx_initcallp( )** and **dx_deltones( )**.

2.4. Modifying the Default PerfectCall Tone Definitions

PerfectCall Call Analysis makes use of Global Tone Detection (GTD) tone definitions for three different types of dial tones, two busy tones, one ringback tone, and two fax tones. The tone definitions specify the frequencies, durations and repetition counts necessary to identify each of these signals. Each signal may consist of a single tone or a dual tone.

The Voice Driver contains default definitions for each of these tones. The default definitions will allow applications to identify the tones correctly in most countries and for most switching equipment. If, however, a situation arises in which the default tone definitions are not adequate, three functions are provided to modify the standard tone definitions:

- **dx_chgfreq( )** specifies frequencies and tolerances for one or both frequencies of a single frequency or dual-frequency tone.
- **dx_chgdur( )** specifies the cadence (on time, off time, and acceptable deviations) for a tone.
- **dx_chgrepcnt( )** specifies the repetition count required to identify a tone.

These functions only change the tone definitions, they do not alter the process of PerfectCall Call Analysis. When the **dx_initcallp( )** function is invoked to activate PerfectCall Call Analysis on a particular channel, it uses the current tone definitions to initialize that channel. Multiple calls to **dx_initcallp( )** may therefore use varying tone definitions, and several channels can operate simultaneously with different tone definitions.

Details on these functions may be found in the *Voice Software Reference: Programmer's Guide.*

2.5. How to Use Call Analysis

The following procedure describes how to initiate an outbound call with Call Analysis:

1. Set up the Call Analysis Parameter structure (DX_CAP), which contains parameters that control the operation of Call Analysis.
2. Execute the **dx_dial( )** function to initiate Call Analysis.
   - **If running dx_dial( ) asynchronously**, use the Event Management functions to determine when dialing with Call Analysis is complete (TDX_CALLP termination event).
   - **If running dx_dial( ) synchronously**, wait for **dx_dial( )** to return a value greater than 0 to indicate successful completion.

   See the **dx_dial( )** function description in the *Voice Software Reference: Programmer's Guide* for information about asynchronous and synchronous operation.
3. Use **ATDX_CPTERM( )** to determine the outcome of the call.

an intercept	CR_CEPT
no ringback	CR_NORB
busy signal	CR_BUSY
no answer	CR_NOANS
fax machine	CR_FAXTONE
no dial tone	CR_NODIALTONE
connect	CR_CNCT
Call Analysis stopped	CR_STOPD
Call Analysis error	CR_ERROR

NOTE: When running **dx_dial( )** synchronously, these results are also returned by **dx_dial( )**.

4. Obtain additional termination, frequency, or cadence information (such as the length of the salutation) as desired using Extended Attribute functions.

Each of these steps is described in detail in the following pages.

NOTE: For information about **dx_dial( )**, **ATDX_CPTERM( )**, and the DX_CAP data structure, see the *Voice Software Reference: Programmer's Guide*:

2.5.1. Set Up the Call Analysis Parameter Structure (DX_CAP)

The **dx_dial( )** function enables Call Analysis after dialing.

If you want to customize the parameters for your environment, you must set up the Call Analysis Parameter structure before calling **dx_dial( )**. By adjusting the DX_CAP parameters, you can:

- Eliminate Call Analysis functions that do not pertain to your environment.
- Optimize performance of the required functions.
- Support non-standard system configurations.
- Perform additional functions, such as determining whether the called party is a business, residence, or answering machine.

To set up the DX_CAP structure for Call Analysis: Execute the **dx_clrcap( )** function to clear the DX_CAP and initialize the parameters to 0. The value 0 indicates that the default value will be used for that particular parameter. **dx_dial( )** can also be set to run with default Call Analysis parameter values, by specifying a NULL pointer to the DX_CAP structure.

For more detailed information on the DX_CAP block parameters, refer to *Section 2.6. How the DX_CAP Controls Call Analysis.*

2.5.2. To Initiate Call Analysis Use the dx_dial( ) Function

Enable Call Analysis by calling **dx_dial( )** with the **mode** function argument set to DX_CALLP. Termination of dialing with Call Analysis is indicated differently depending on whether the function is running asynchronously or synchronously. More information on **dx_dial( )** can be found in the *Voice Software Reference: Programmer's Guide*.

2.5.3. Determine the Outcome of the Call

Once **dx_dial( )** with Call Analysis has terminated, use the Extended Attribute function **ATDX_CPTERM( )** to determine the outcome of the call. **ATDX_CPTERM( )** will return one of the following Call Analysis Termination results:

CR_BUSY	Called line was busy.
CR_CNCT	Called line was connected.
CR_FAXTONE	Called line was answered by a fax machine or a modem (PerfectCall Call Analysis only).
CR_NOANS	Called line did not answer.
CR_NODIALTONE	Called line failed to produce a dial tone (PerfectCall Call Analysis only).
CR_NORB	Called line did not ring.
CR_CEPT	Called line received operator intercept (SIT). The Extended Attribute functions provide information on the detected frequencies and durations.
CR_STOPD	Call Analysis stopped due to **dx_stopch( )**.
CR_ERROR	Call Analysis error occurred. **ATDX_CPERROR( )** returns the type of the Call Analysis error.

Figure 3 illustrates the possible outcomes of Call Analysis.

Figure 4 illustrates the possible outcomes of PerfectCall Call Analysis.

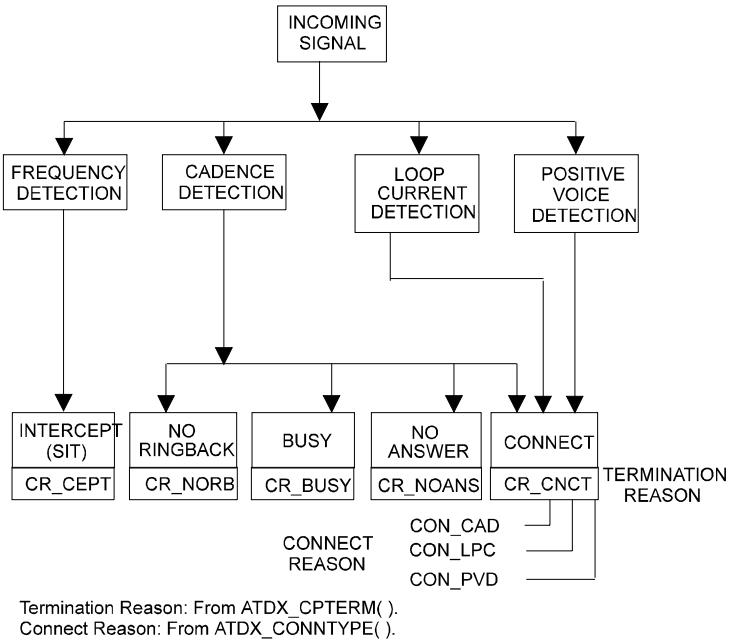


Figure 3. Call Analysis Outcomes for Basic Call Analysis

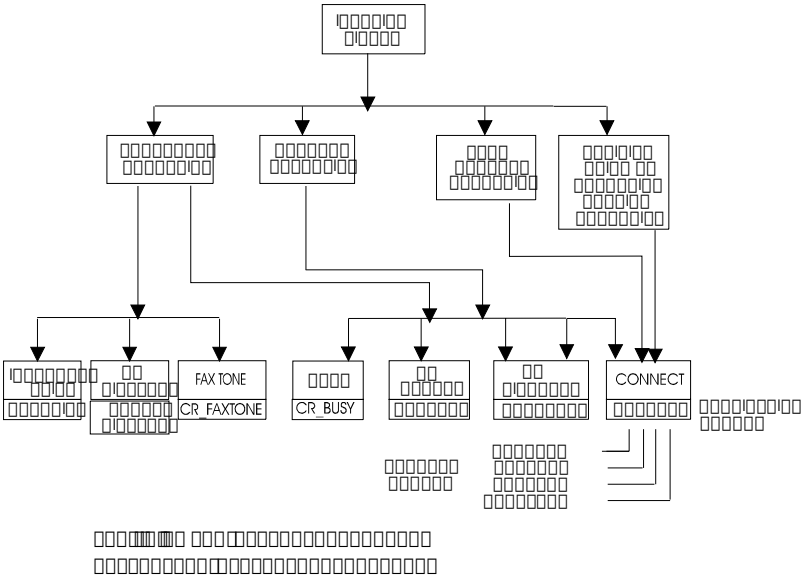


Figure 4. Call Analysis Outcomes for PerfectCall Call Analysis

2.5.4. Obtain Additional Call Outcome Information

Additional Call Analysis information can be retrieved using the following Extended Attribute functions:

ATDX_ANSRSIZ( )	• Returns duration of answer
ATDX_CPERROR( )	• Returns call analysis error
ATDX_CPTERM( )	• Returns last call analysis termination
ATDX_CONNTYPE( )	• Returns connection type
ATDX_CRTNID( )	• Returns the identifier of the tone that caused the most recent Call Analysis termination

ATDX_ANSRSIZ( )	• Returns duration of answer
ATDX_DTNFAIL( )	• Returns the dial tone character that indicates which dial tone Call Analysis failed to detect
ATDX_FRQDUR( )	• Returns duration of first frequency detected
ATDX_FRQDUR2( )	• Returns duration of second frequency detected
ATDX_FRQDUR3( )	• Returns duration of third frequency detected
ATDX_FRQHZ( )	• Returns frequency detected in Hz of first detected tone
ATDX_FRQHZ2( )	• Returns frequency of second detected tone
ATDX_FRQHZ3( )	• Returns frequency of third detected tone
ATDX_FRQOUT( )	• Returns percent of frequency out of bounds
ATDX_LONGLOW( )	• Returns duration of longer silence
ATDX_SHORTLO( )	• Returns duration of shorter silence
ATDX_SIZEHI( )	• Returns duration of non-silence

For a discussion of how frequency and cadence information returned by these Extended Attribute functions relate to the DX_CAP parameters, refer to *Section 2.6. How the DX_CAP Controls Call Analysis*.

2.6. How the DX_CAP Controls Call Analysis

The following sections describe the DX_CAP parameters that control Frequency Detection, Cadence Detection, Loop Current Detection, Positive Voice Detection, and Positive Answering Machine Detection.

The DX_CAP structure, as defined in the header file, is listed in the *Voice Software Reference: Programmer's Guide*.

2.6.1. Selecting SIT Frequency Detection, Positive Voice and Positive Answering Machine Detection

The Call Analysis Parameter structure (DX_CAP) parameter **ca_intflg** (intercept mode flag) is used to enable or disable Frequency Detection, Positive Voice Detection, and/or Positive Answering Machine Detection for Call Analysis.

The **ca_intflg** parameter also determines when SIT Frequency Detection should terminate and return an *intercept*. This can occur immediately upon detection of the specified frequencies or after waiting for a connect indicated by Cadence Detection, Loop Current Detection, Positive Voice Detection, or Positive Answering Machine Detection.

The following definitions are provided for use with the **ca_intflg** parameter:

Parameter	Description	
ca_intflg	Intercept Mode Flag: This parameter enables or disables SIT Frequency Detection, Positive Voice Detection (PVD), and/or Positive Answering Machine Detection (PAMD), and selects the mode of operation for Frequency Detection. Default: 1 (DX_OPTEN). The following modes are possible:	
	DX_OPTEN	Obsolete. It applied only to non_DSP boards such as the D/41B and D/21B voice boards; use DX_OPTNOCON for DSP boards.
	DX_OPTDIS	Disable SIT Detection, PAMD and PVD.
	DX_OPTNOCON	Enable Frequency Detection (terminate Call Analysis and return an *intercept* immediately upon detecting tones). Frequency Detection returns an *intercept* immediately after detecting a valid frequency.
	DX_PVDENABLE	Enable PVD.
	DX_PVDOPTEN	Obsolete. It applied only to non-DSP boards such as the D/41B and D/21B voice boards; use the DX_PVDOPTNOCON for DSP boards.
	DX_PVDOPTNOCON	Enable PVD and DX_OPTNOCON.
	DX_PAMDENABLE	Enable PAMD and PVD.
	DX_PAMDOPTEN	Enable PAMD, PVD and DX_OPTNOCON.

NOTE: Please note that the data structures described "Obsolete" are NOT obsolete for PEB.

2.6.2. SIT Frequency Detection

SIT Frequency Detection operates simultaneously with all other Call Analysis detection methods. The purpose of Frequency Detection is to detect the tri-tone Special Information Tone (SIT) sequences and other single-frequency tones. Detection of an SIT sequence indicates an operator intercept or other problem in completing the call.

SIT Frequency Detection can detect virtually any single-frequency tone below 2100 Hz and above 300 Hz.

Tri-Tone SIT Sequences

Tone information for the 4 SIT sequences is provided in *Table 1*. The frequencies are represented in Hz and the length of the signal is in 10 ms units.

Table 1. Special Information Tone Sequences

SIT Name	SIT Description	1st Tone Freq.	1st Tone Len.	2nd Tone Freq.	2nd Tone Len.	3rd Tone Freq.	3rd Tone Len.
NC	No Circuit Found	985	38	1429	38	1777	38
IC	Operator Intercept	914	27	1371	27	1777	38
VC	Vacant Circuit	985	38	1370	27	1777	38
RO	Reorder (system busy)	9148	27	1429	38	1777	38

The length of the first tone is not dependable; often it is shortened or cut.

Setting Tri-Tone Frequency Detection Parameters

Frequency Detection on voice boards is designed to detect all three tones in the tri-tone SIT sequence. To detect all three tones in the SIT sequence, you must specify the frequency detection parameters in the DX_CAP for all three tones in the sequence.

To detect all four tri-tone SIT sequences:

- Set an appropriate frequency detection range in the DX_CAP to detect each tone across all four SIT sequences. Set the first frequency detection range to detect the first tone for all four SIT sequences (approximately 900 to 1000 Hz). Set the second frequency detection range to detect the second tone for all four SIT sequences (approximately 1350 to 1450 Hz). Set the third frequency detection range to detect the third tone for all four SIT sequences (approximately 1725 to 1825 Hz).
- Set an appropriate detection time using the ca_timefrq and ca_mxtimefrq parameters to detect each tone across all four SIT sequences. For each tone, set ca_timefrq to 5 and ca_mxtimefrq to 50 to detect all SIT tones. The tones range in length from 27 to 38 (in 10 ms units), with some tones occasionally cut short by the central office.

NOTE: Occasionally, the first tone can also be truncated by a delay in the onset of Call Analysis due to the setting of ca_stdely.

- After an SIT sequence is detected, **ATDX_CPTERM( )** will return CR_CEPT to indicate an operator intercept, and you can determine which SIT sequence was detected by obtaining the actual detected frequency and duration for the tri-tone sequence using Extended Attribute functions.

See the following fields in DX_CAP to be used for Frequency Detection on voice boards. Frequencies are specified in Hz, and time is specified in 10 ms units. To enable detection of the second and third tones, you must set the frequency detection range and time for each tone.

General

Parameter	Description
ca_stdely	Start Delay: The delay after dialing has been completed and before starting Frequency Detection. This parameter also determines the start of Cadence Detection and Positive Voice Detection. Default: 25 (10 ms units). Note that this can affect detection of the first element of an operator intercept tone.

First Tone

Parameter	Description
ca_lowerfrq	Lower Frequency: Lower bound for first tone in Hz. Default: 900.
ca_lowerfrq	Lower Frequency: Lower bound for first tone in Hz. Default: 900.
ca_upperfrq	Upper Frequency: Upper bound for first tone in Hz. Default: 1000. Adjust higher for additional operator intercept tones.
ca_timefrq	Time Frequency: Minimum time for first tone to remain in bounds. The minimum amount of time required for the audio signal to remain within the frequency detection range for it to be detected. The audio signal must not be greater than **ca_upperfrq** or lower than **ca_lowerfrq** for at least the time interval specified in **ca_timefrq**. Default: 5 (10 ms units)
ca_mxtimefrq	Maximum Time Frequency: Maximum allowable time for first tone to be present. Default: 0 (10 ms units).

Second Tone

NOTE: This tone is disabled initially and must be activated by the application using these variables

Parameter	Description
ca_lower2frq	Lower Bound for second Frequency: Lower bound for second tone in Hz. Default: 0
ca_upper2frq	Upper Bound for second Frequency: Upper bound for second tone in Hz. Default: 0.
ca_time2frq	Time for second Frequency: Minimum time for second tone to remain in bounds. Default: 0 (10 ms units).
ca_mxtime2frq	Maximum Time for second Frequency: Maximum allowable time for second tone to be present. Default: 0 (10 ms units).

Third Tone

NOTE: This tone is disabled initially and must be activated by the application using these variables.

Parameter	Description
ca_lower3frq	Lower Bound for third Frequency: Lower bound for third tone in Hz. Default: 0.
ca_upper3frq	Upper Bound for third Frequency: Upper bound for third tone in Hz. Default: 0.
ca_time3frq	Time for third Frequency: Minimum time for third tone to remain in bounds. Default: 0 (10 ms units).
ca_mxtime3frq	Maximum Time for third Frequency: Maximum allowable time for third tone to be present. Default: 0 (10 ms units)

Tri-Tone Frequency Information Returned by Extended Attribute Functions

Upon detection of the specified sequence of frequencies, Extended Attribute functions can be used to provide the exact frequency and duration of each tone in the sequence. The frequency and duration information will allow exact determination of all four SIT sequences.

Extended Attribute functions are used to provide information on the frequencies detected by Call Analysis:

Tri-Tone Frequency Information First Tone (ca_lowerfrq and ca_upperfrq)

ATDX_FRQHZ( )	Frequency Hertz: Frequency in Hz of the tone detected in the tone detection range specified by the DX_CAP **ca_lowerfrq** and **ca_upperfrq** parameters; usually the first tone of an SIT sequence. This function can be called on non-DSP boards.
ATDX_FRQDUR( )	Frequency Duration: Duration of the tone detected

ATDX_FRQHZ( )	Frequency Hertz: Frequency in Hz of the tone detected in the tone detection range specified by the DX_CAP **ca_lowerfrq** and **ca_upperfrq** parameters; usually the first tone of an SIT sequence. This function can be called on non-DSP boards. in the tone detection range specified by the DX_CAP **ca_lowerfrq** and **ca_upperfrq** parameters; usually the first tone of an SIT sequence (10 ms units).

Tri-Tone Frequency Information Second Tone (ca_lower2frq and ca_upper2frq)

ATDX_FRQHZ2( )	Frequency Hertz 2: Frequency in Hz of the tone detected in the tone detection range specified by the DX_CAP **ca_lower2frq** and **ca_upper2frq** parameters; usually the second tone of an SIT sequence.
ATDX_FRQDUR2( )	Frequency Duration 2: Duration of the tone detected in the tone detection range specified by the DX_CAP **ca_lowerfrq** and **ca_upperfrq** parameters; usually the second tone of an SIT sequence (10 ms units).

Tri-Tone Frequency Information Third Tone (ca_lower3frq and ca_upper3frq)

ATDX_FRQHZ3( )	• Frequency Hertz 3: Frequency in Hz of the tone detected in the tone detection range specified by the DX_CAP **ca_lower3frq** and **ca_upper3frq** parameters; usually the third tone of an SIT sequence.
ATDX_FRQDUR3( )	• Frequency Duration 3: Duration of the tone detected in the tone detection range specified by the DX_CAP **ca_lower3frq** and **ca_upper3frq** parameters; usually the third tone of an SIT sequence (10 ms units).

Global Tone Detection Tone Memory Usage

If you use Call Analysis to identify the tri-tone SIT sequences, Call Analysis will create tone detection templates internally, and this will reduce the number of tone templates that can be created using Global Tone Detection functions. See *Chapter 5. Global Tone Detection/Generation* for information relating to memory usage for Global Tone Detection.

Call Analysis will create one tone detection template for each single-frequency tone with a 100 Hz detection range. For example, if detecting the set of tri-tone SIT sequences (three frequencies) on each of four channels, the number of allowable user-defined tones will be reduced by three per channel.

If you initiate Call Analysis and there is not enough memory to create the SIT tone detection templates internally, you will get a CR_MEMERR error. This indicates that you are trying to exceed the maximum number of tone detection templates. The tone detection range should be limited to a maximum of 100 Hz per tone to reduce the chance of exceeding the available memory.

Frequency Detection Errors

The frequency detection range specified by the lower and upper bounds for each tone cannot overlap with each other, otherwise an error will be produced when the driver attempts to create the internal tone detection templates. For example, if **ca_upperfrq** is 1000 and **ca_lower2frq** is also 1000, an overlap occurs and will result in an error. Also the lower bound of each frequency detection range must be

less than the upper bound (for example, **ca_lower2frq** must be less than **ca_upper2frq**).

Setting Single Tone Frequency Detection Parameters

Setting single tone frequency detection parameters allows you to identify that a SIT sequence was encountered because 1 of the tri-tones in the SIT sequence was detected. But Frequency Detection cannot determine exactly which SIT sequence was encountered, because it is necessary to identify 2 tones in the SIT sequence to distinguish among the 4 possible SIT sequences.

The default frequency detection range is 900 to 1000 Hz, which is set to detect the first tone in any SIT sequence. Because the first tone is often truncated, you may want to increase **ca_upperfrq** to 1800 Hz so that it includes the third tone. If this results in too many false detections, you can set Frequency Detection to detect only the third tone by setting **ca_lowerfrq** to 1750 and **ca_upperfrq** to 1800.

The following fields in the DX_CAP are used for Frequency Detection. Frequencies are specified in Hz, and time is specified in 10 ms units.

Parameter	Description
ca_stdely	Start Delay: The delay after dialing has been completed and before starting Frequency Detection. This parameter also determines the start of Cadence Detection. Default: 25 (10 ms units).
ca_infltr	Not used.
ca_lowerfrq	Lower Frequency: Lower bound for tone in Hz. Default: 900.
ca_upperfrq	Upper Frequency: Upper bound for tone in Hz. Default: 1000.
ca_rejctfrq	Not used.

Single Tone Frequency Information Returned

Upon detection of a frequency in the specified range, the Extended Attribute functions can be used to provide the exact frequency that was detected.

The following Extended Attribute functions return information on the single tone detected in the tone detection range specified by the DX_CAP **ca_lowerfrq** and **ca_upperfrq** fields.

ATDX_FRQOUT( )	• Not used.
ATDX_FRQHZ( )	• Frequency Hertz: Frequency in Hz of the tone detected in the tone detection range specified by the DX_CAP ca_lowerfrq and ca_upperfrq parameters; usually the first tone of an SIT sequence.

2.6.3. Cadence Detection in Basic Call Analysis

The Cadence Detection algorithm has been optimized for use in the United States standard network environment.

Cadence Detection as discussed here, and the Call Progress Characterization (CPC) Utility, are relevant to Basic Call Analysis only. For PerfectCall Call Analysis, refer to *Section 2.6.4. Tone Detection in PerfectCall Call Analysis*. The CPC program, which runs under MS-DOS, can be used to collect call progress data in both Basic Call Analysis and PerfectCall Call Analysis.

If your system is operating in another type of environment (such as behind a PBX), you can customize the Cadence Detection algorithm to suit your system through the adjustment of the Cadence Detection parameters. The Call Progress Characterization (CPC) Utility, which runs under MS-DOS can be used to determine the Cadence Detection parameter requirements for your system.

The following section discusses Cadence Detection and some of the most commonly adjusted Cadence Detection parameters. An entire listing of the DX_CAP, including all Cadence Detection parameters, can be found in the *Voice Software Reference: Programmer's Guide*.

Cadence Detection analyzes the audio signal on the line to detect a repeating pattern of sound and silence, such as the pattern produced by a ringback or a busy signal. These patterns are called *audio cadences*. Once a cadence has been established, it can be classified as a single ring, a double ring or a busy signal by comparing the periods of sound and silence to established parameters.

NOTES: **1.** Sound is referred to as *nonsilence*.

2. The algorithm used for cadence detection is disclosed and protected under U.S. patent 4,477,698 of Melissa Electronic Labs, and other patents pending for Dialogic Corporation.

Typical Cadence Patterns

The following figures show some typical cadence patterns.

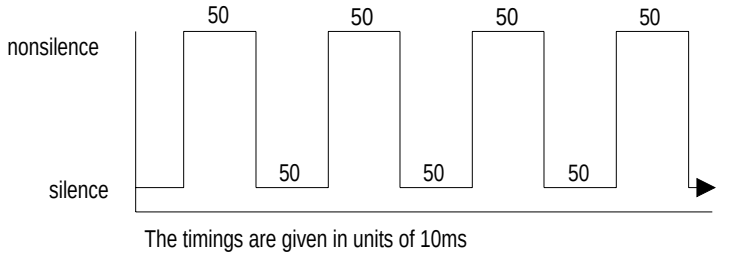


Figure 5. A Standard Busy Signal

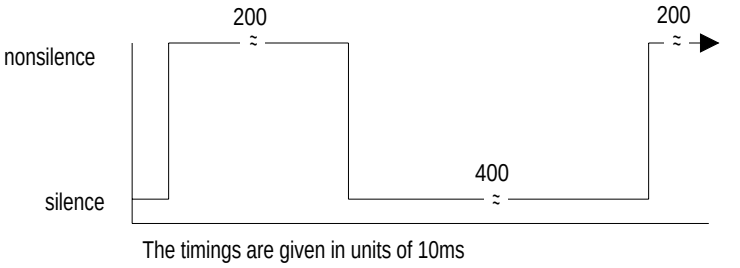


Figure 6. A Standard Single Ring

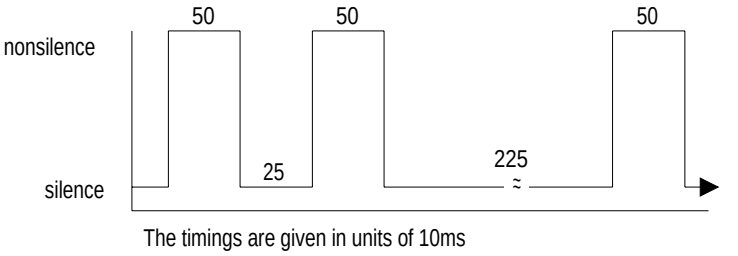


Figure 7. A Type of Double Ring

Elements of a Cadence

From the preceding cadence examples, you can see that a given cadence may contain 2 silence periods with different durations, such as for a double ring, but in general, the nonsilence periods have the same duration. To identify and distinguish between the different types of cadences, the Voice Driver must detect two silence and two nonsilence periods in the audio signal. *Figure 8* illustrates cadence detection.

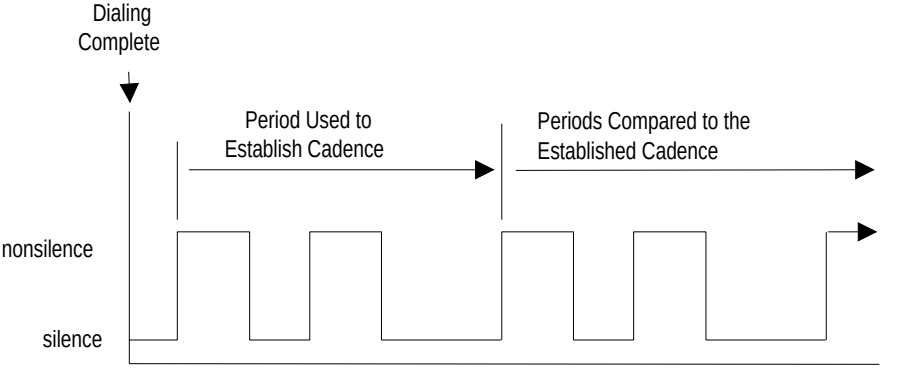


Figure 8. Cadence Detection

Once the cadence is established, the cadence values can be retrieved using the following Extended Attribute functions:

ATDX_SIZEHI( )	• Length of the nonsilence period (in 10 ms units) for the detected cadence.
ATDX_SHORTLOW( )	• Length of the shortest silence period for the detected cadence (in 10 ms units).
ATDX_LONGLOW( )	• Length of the longest silence period for the detected cadence (in 10 ms units).

Only one nonsilence period is used to define the cadence because the nonsilence periods have the same duration. *Figure 9* shows the elements of an established cadence.

Figure 9. Elements of Established Cadence

The durations of subsequent states are compared with these fields to see if the cadence has been broken.

Outcomes of Cadence Detection

Cadence Detection can identify the following conditions during the period used to establish the cadence or after the cadence has been established:

- No Ringback
- Connect
- Busy
- No Answer

Although Loop Current Detection and Positive Voice Detection provide complementary means of detecting a connect, Cadence Detection provides the only way in Basic Call Analysis to detect a no ringback, busy, or no answer.

Cadence Detection can identify the following conditions during the period used to establish the cadence:

No Ringback While the cadence is being established, Cadence Detection determines whether the signal is continuous silence or nonsilence, in which case, Cadence Detection returns a **no ringback**, indicating there is a problem in completing the call.

Connect While the cadence is being established, Cadence Detection determines whether the audio signal departs from acceptable network standards for busy or ring signals. In this case, Cadence Detection returns a *connect*, indicating that there was a break from general cadence standards.

Cadence Detection can identify the following conditions after the cadence has been established:

Connect After the cadence has been established, Cadence Detection determines whether the audio signal departs from the established cadence. In this case, Cadence Detection returns a *connect*, indicating that there was a break in the established cadence.

No Answer After the cadence has been established, Cadence Detection determines whether the cadence belongs to a single or double ring. In this case, Cadence Detection can return a *no answer*, indicating there was no break in the ring cadence for a specified number of times.

Busy After the cadence has been established, Cadence Detection determines whether the cadence belongs to a slow busy signal. In this case, Cadence Detection can return a *busy*, indicating that the busy cadence was repeated for a specified number of times.

To determine whether the ring cadence is a double or single ring, compare the value returned by the **ATDX_SHORTLOW( )** function to the DX_CAP field

ca_lo2rmin. If the **ATDX_SHORTLOW( )** value is less than **ca_lo2rmin**, the cadence is a double ring; otherwise it is a single ring.

Setting Selected Cadence Detection Parameters

Only the most commonly adjusted Cadence Detection parameters are discussed here. A listing of the DX_CAP function can be found in the *Voice Software Reference: Programmer's Guide*.

You should only need to adjust Cadence Detection parameters for network environments that do not conform to the U.S. standard network environment (such as behind a PBX).

The Call Progress Characterization Utility (CPC) Utility, which runs under MS-DOS, can be used to determine the correct cadence detection parameter settings for your system.

Call Progress Characterization General Cadence Detection Parameters

Parameter	Description
ca_stdely	Start Delay: The delay after dialing has been completed and before starting Cadence Detection. This parameter also determines the start of Frequency Detection and Positive Voice Detection. Default: 25 (10 ms units) = 0.25 seconds. Be careful with this variable. Setting this variable too small may allow switching transients or, if too long, miss critical signaling.

To eliminate audio signal glitches over the telephone line, the parameters **ca_logltch** and **ca_higltch** are used to determine the minimum acceptable length of a valid silence or nonsilence duration. Any silence interval shorter than **ca_logltch** is ignored, and any nonsilence interval shorter than **ca_higltch** is ignored.

Parameter	Description
ca_higltch	High Glitch: The maximum nonsilence period to ignore. Used to help eliminate spurious nonsilence intervals. Default: 19 (in 10 ms units)
ca_logltch	Low Glitch: The maximum silence period to ignore. Used to help eliminate spurious silence intervals. Default: 15 (in 10 ms units).

Cadence Detection Parameters Affecting a No Ringback

After Cadence Detection begins, it waits for an audio signal of nonsilence. The maximum waiting time is determined by the parameter **ca_cnosig** (continuous no signal). If the length of this period of silence exceeds the value of **ca_cnosig**, a **no ringback** is returned (see *Figure 10. No Ringback Due to Continuous No Signal*). This usually indicates a dead or disconnected telephone line or some other system malfunction.

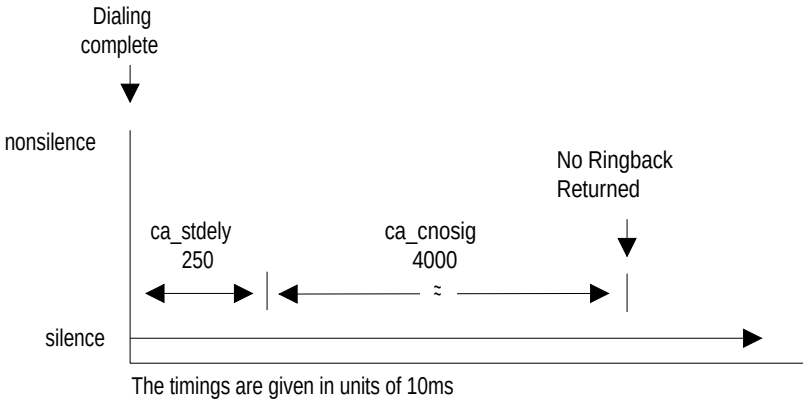


Figure 10. No Ringback Due to Continuous No Signal

Parameter	Description
ca_cnsig	Continuous No Signal: The maximum time of silence (no signal) allowed immediately after Cadence Detection begins. If exceeded, a *no ringback* is returned. Default: 4000 (in 10 ms units), or 40 seconds.

If the length of any period of nonsilence exceeds the value of **ca_cnosil** (continuous nonsilence), a *no ringback* is returned (see *Figure 11. No Ringback Due to Continuous Nonsilence*).

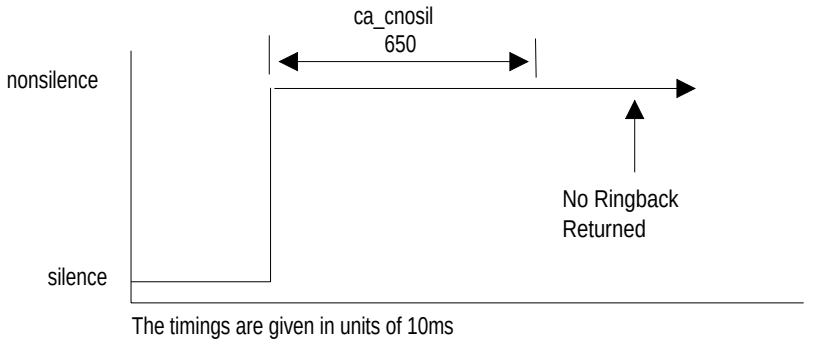


Figure 11. No Ringback Due to Continuous Nonsilence

Parameter	Description
ca_cnosil	Continuous Nonsilence: The maximum length of nonsilence allowed. If exceeded, a *no ringback* is returned. Default: 650 (in 10 ms units), or 6.5 seconds.

Cadence Detection Parameters Affecting a No Answer or Busy

By using the **ca_nbrdna** parameter, you can set the maximum number of ring cadence repetitions that will be detected before returning a *no answer*.

By using the **ca_nbrdna** and **ca_nsbusy** parameters, you can set the maximum number of busy cadence repetitions.

Parameter	Description
ca_nbrdna	Number of Rings Before Detecting No Answer: The number of single or double rings to wait before returning a *no answer*. Default: 4.
ca_nsbusy	Nonsilence Busy: The number of nonsilence periods in addition to **ca_nbrdna** to wait before returning a *busy*. Default: 0. **ca_nsbusy** is *added* to **ca_nbrdna** to give the actual number of busy cadences at which to return busy. Note that even though ca_nsbusy is declared as an unsigned variable, it can be a small negative number. Do not allow **ca_nbrdna + ca_nsbusy** to equal 2. This is a foible of the 2's complement bit mapping of a small negative number to an unsigned variable.

Cadence Detection Parameters Affecting a Connect

Cadence Detection parameters are used to measure the length of the salutation when the phone is answered. The salutation is the greeting when a person answers the phone, or an announcement when an answering machine or computer answers the phone.

By examining the length of the greeting or salutation you receive when the phone is answered, you may be able to distinguish between an answer at home, at a business, or by an answering machine.

The length of the salutation is returned by the **ATDX_ANSRSIZ( )** function. By examining the value returned, you can estimate the kind of answer that was received.

Normally, a person at home will answer the phone with a brief salutation that lasts about 1 second, such as “Hello” or “Smith Residence.” A business will usually answer the phone with a longer greeting that lasts from 1.5 to 3 seconds, such as “Good afternoon, Dialogic Corporation.” An answering machine or computer will usually play an extended message that lasts more than 3 or 4 seconds.

This method is not 100% accurate, for the following reasons:

- The length of the salutation can vary greatly.

- A pause in the middle of the salutation can cause a premature connect event.
- If the phone is picked up in the middle of a ringback, the ringback tone may be considered part of the salutation, making the **ATDX_ANSRSIZ( )** return value inaccurate.

In the last case, if someone answers the phone in the middle of a ring and quickly says "Hello," the nonsilence of the ring will be indistinguishable from the nonsilence of voice that immediately follows, and the resulting **ATDX_ANSRSIZ( )** return value may include both the partial ring and the voice. In this case, the return value may deviate from the actual salutation by 0 to +1.8 seconds. The salutation would appear to be the same as when someone answers the phone after a full ring and says two words.

NOTE: A return value of 180 to 480 may deviate from the actual length of the salutation by 0 to +1.8 seconds.

Cadence Detection will measure the length of the salutation when the **ca_hedge** (hello edge) parameter is set to 2 (the default).

Parameter	Description
ca_hedge	Hello Edge: The point at which a *connect* will be returned to the application, either the rising edge (immediately when a connect is detected) or the falling edge (after the end of the salutation). 1 = rising edge. 2 = falling edge. Default: 2 (*connect* returned on falling edge of salutation). Try changing this if the called party has to say "Hello" twice to trigger the answer event.

Because a greeting might consist of several words, Call Analysis waits for an specified period of silence before assuming the salutation is finished. The **ca_ansrdgl** (answer deglitcher) parameter determines when the end of the salutation occurs. This parameter specifies the maximum amount of silence allowed in a salutation before it is determined to be the end of the salutation. To use **ca_ansrdgl**, set it to approximately 50 (in 10 ms units).

Parameter	Description
ca_ansrdgl	Answer Deglitcher: The maximum silence period (in 10 ms units) allowed between words in a salutation. This parameter should be enabled only when you are interested in measuring the length of the salutation. Default: -1 (disabled).

The **ca_maxansr** (maximum answer) parameter determines the maximum allowable answer size before returning a *connect*.

Parameter	Description
ca_maxansr	Maximum Answer: The maximum allowable length of **ansrsize**. When **ansrsize** exceeds **ca_maxansr**, a *connect* is returned to the application. Default: 1000 (in 10 ms units), or 10 seconds.

Figure 12 shows how the **ca_ansrdgl** parameter works.

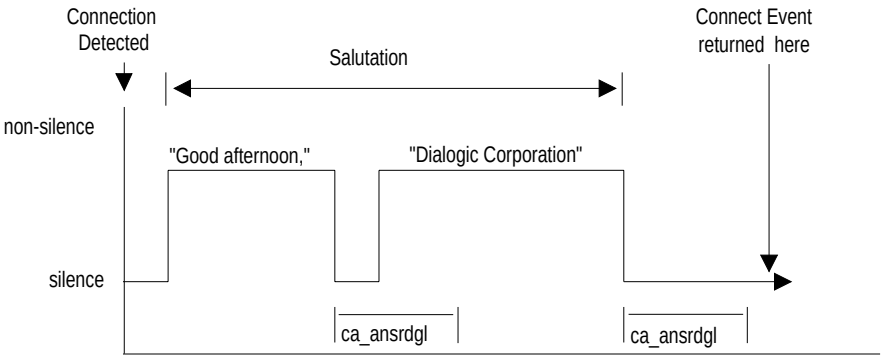


Figure 12. Cadence Detection Salutation Processing

When **ca_hedge** = 2, Cadence Detection waits for the end of the salutation before returning a *connect*. The end of the salutation occurs when the salutation contains a period of silence that exceeds **ca_ansrdgl** or the total length of the salutation exceeds **ca_maxansr**. When the *connect* event is returned, the length of the salutation can be retrieved using the **ATDX_ANSRSIZ( )** function.

After Call Analysis is complete, check call **ATDX_ANSRSIZ( )**. If the return value is less than 180 (1.8 seconds), you have probably contacted a residence. A return value of 180 to 300 is probably a business. If the return value is larger than 480, you have probably contacted an answering machine. A return value of 0 means that a *connect* was returned because excessive silence was detected. This can vary greatly in practice.

NOTE: When a connect is detected through Positive Voice Detection or Loop Current Detection, the DX_CAP parameters **ca_hedge**, **ca_ansrdgl**, and **ca_maxansr** are ignored.

Cadence Information Returned Using Extended Attribute Functions

ATDX_SIZEHI( )	• Size High: Duration of the cadence non-silence period (in 10 ms units).
ATDX_SHORTLOW( )	• Short Low: Duration of the cadence shorter silence period (in 10 ms units).
ATDX_LONGLOW( )	• Long Low: Duration of the cadence longer silence period (in 10 ms units).
ATDX_ANSRSIZ( )	• Answer Size: Duration of answer if a connect occurred (in 10 ms units) (ignored for what for PVD).
ATDX_CONNTYPE( )	• Connection type. If ATDX_CONNTYPE( ) returns CON_CAD, the connect was due to Cadence Detection.

2.6.4. Tone Detection in PerfectCall Call Analysis

PerfectCall Call Analysis uses a combination of Cadence Detection and Frequency Detection to identify certain signals during the course of an outgoing call. Cadence detection identifies repeating patterns of sound and silence, and Frequency Detection determines the pitch of the signal. Together, the cadence and frequency of a signal make up its *tone definition*.

Unlike Basic Call Analysis, which uses fields in the DX_CAP structure to store signal cadence information, PerfectCall Call Analysis uses tone definitions which

are contained in the Voice Driver itself. Functions are available to modify these default tone definitions.

Types of PerfectCall Tones

Tone definitions are used to identify several kinds of signals. The following list shows the defined tones and their tone identifiers. Tone identifiers are returned by the **ATDX_CRTNID( )** function.

#define	Tone Description
TID_DIAL_LCL	Local dial tone
TID_DIAL_INTL	International dial tone
TID_DIAL_XTRA	Special (or "extra") dial tone
TID_BUSY1	Single tone busy signal
TID_BUSY2	Dual tone busy signal
TID_RNGBK1	Ringback tone
TID_FAX1	A fax CNG tone
TID_FAX2	A fax CED or modem tone
TID_DISCONNECT	Disconnect tone (post-connect)

The tone identifiers are used in calls to the functions **dx_chgfreq( )**, **dx_chgdur( )**, and **dx_chgrepcnt( )** to change the tone definitions. Refer to these functions in the *Voice Software Reference: Programmer's Guide.* Also see *Section 2.3. How to Enable PerfectCall Call Analysis*.

Dial Tone Detection

Wherever PerfectCall Call Analysis is in effect, a dial string for an outgoing call may specify special ASCII characters that instruct the system to wait for a certain kind of dial tone. The following additional special characters may appear in a dial string:

L Wait for a local dial tone

I Wait for an international dial tone

X Wait for a special (extra) dial tone

The tone definitions for each of these dial tones is set for each channel at the time of the **dx_initcallp( )** function. In addition, the following DX_CAP fields identify how long to wait for a dial tone, and how long the dial tone must remain stable.

ca_dtn_pres Dial Tone Present: The length of time that the dial tone must be continuously present (in 10 ms units). If a dial tone is present for this amount of time, dialing of the dial string proceeds. Default value: 100 (one second).

ca_dtn_npres Dial Tone Not Present: The length of time to wait before declaring the dial tone not present (in 10 ms units). If a dial tone of sufficient length (**ca_dtn_pres**) is not found within this period of time, Call Analysis terminates with the reason CR_NODIALTONE. The dial tone character (**L**, **I**, or **X**) for the missing dial tone can be obtained using **ATDX_DTNFAIL( )**. Default value: 300 (three seconds).

ca_dtn_deboff Dial Tone Debounce: The maximum duration of a break in an otherwise continuous dial tone before it is considered invalid (in 10 ms units). This parameter is used for ignoring short drops in dial tone. If a drop longer than **ca_dtn_deboff** occurs, then dial tone is no longer considered present, and another dial tone must begin and be continuous for **ca_dtn_pres**. Default value: 10 (100 msec).

Ringback Detection

PerfectCall Call Analysis uses the tone definition for ringback to identify the first ringback signal of an outgoing call. At the end of the first ringback (that is, normally, at the beginning of the second ringback), a timer goes into effect. The system continues to identify ringback signals (but does not count them). If a break occurs in the ringback cadence, the call is assumed to have been answered, and Call Analysis terminates with the reason CR_CONN (connect); the connection type returned by the **ATDX_CONNTYPE( )** function will be CON_CAD (cadence break).

However, if the timer expires before a connect is detected, then the call is deemed unanswered, and Call Analysis terminates with the reason CR_NOAN.

The following DX_CAP fields govern this behavior:

ca_stdely	Start Delay: The delay after dialing has been completed before starting Cadence Detection, Frequency Detection, and Positive Voice Detection (in 10 ms units). Default: 25 (0.25 seconds).
ca_cnosig	Continuous No Signal: The maximum length of silence (no signal) allowed immediately after the **ca_stdely** period (in 10 ms units). If this duration is exceeded, Call Analysis is terminated with the reason CR_NORNG (no ringback detected). Default value: 4000 (40 seconds).
ca_noanswer	No Answer: The length of time to wait after the first ringback before deciding that the call is not answered (in 10 ms units). If this duration is exceeded, Call Analysis is terminated with the reason CR_NOAN (no answer). Default value: 3000 (30 seconds).
ca_maxintering	Maximum Inter-ring: The maximum length of time to wait between consecutive ringback signals (in 10 ms units). If this duration is exceeded, Call Analysis is terminated with the reason CR_CONN (connected). Default value: 800 (8 seconds).

Busy Tone Detection

There are two busy tones defined in PerfectCall Call Analysis, TID_BUSY1 and TID_BUSY2. If either of them is detected while Frequency Detection and Cadence Detection are active, then Call Analysis is terminated with the reason CR_BUSY. **ATDX_CRTNID( )** identifies which busy tone was detected.

No DX_CAP fields affect busy tone detection.

Positive Answering Machine Detection (PAMD)

The recommended setting for the Call Analysis Parameter structure (DX_CAP) **ca_pamd_spdval** field is PAMD_ACCU. Previously, the **ca_pamd_spdval** field could be set to PAMD_FULL or PAMD_QUICK. This field can now also be set to PAMD_ACCU, which does the most accurate evaluation; it detects live voice

as accurately as PAMD_FULL but is more accurate than PAMD_FULL (although slightly slower) in detecting an answering machine. Use the setting PAMD_ACCU when accuracy is more important than speed.

Positive Answering Machine Detection (PAMD) is available only with PerfectCall Call Analysis. Whenever PAMD is enabled, Positive Voice Detection (PVD) is also enabled. PAMD is enabled by setting the **ca_intflg** field of the DX_CAP structure to one of the following values:

- DX_PAMDENABLE: Enable PAMD and PVD without enabling SIT Frequency Detection.
- DX_PAMDOPTEN: Enable PAMD and PVD, and enable SIT frequency detection.

When enabled, detection of an answering machine will result in the termination of Call Analysis with the reason CR_CONN (connected). The connection type returned by the **ATDX_CONNTYPE( )** function will be CON_PAMD.

To distinguish between a greeting by a live human and one by an answering machine, one of the following methods is used:

- PAMD_QUICK: The quick method examines only the events surrounding the connect time and makes a rapid judgment as to whether or not an answering machine is involved.
- PAMD_FULL: The long method looks at the full greeting to determine whether it came from a human or a machine.
- PAMD_ACCU: The most accurate detection. Provides the same accuracy as PAMD_FULL for detecting live voice, but is more accurate (though slightly slower) than PAMD_FULL for detecting an answering machine. Use this setting when accuracy is more important than speed.

The slower method gives a very accurate determination. However in situations where a fast decision is more important than accuracy, PAMD_QUICK might be preferred.

The following DX_CAP fields govern positive answering machine detection:

ca_pamd_spdval PAMD Speed Value: Whether to make a quick decision

	on Positive Answering Machine Detection. Possible values:	
	PAMD_FULL	Look at greeting (long method). Default.
	PAMD_QUICK	Look at connect only (quick method)
	PAMD_ACCU	Look at greeting (long method) and use most accuracy for detecting an answering machine.
ca_pamd_qtemp	PAMD Qualification Template: The algorithm to use in PAMD. At present there is only one template: PAMD_QUAL1TMP. This parameter must be set to this value.	
ca_pamd_failtime	PAMD fail time: Maximum time to wait for Positive Answering Machine Detection or Positive Voice Detection after a cadence break. Default Value: 400 (in 10 ms units).	
ca_pamd_minring	Minimum PAMD ring: Minimum allowable ring duration for Positive Answering Machine Detection. Default Value: 190 (in 10 ms units).	

Fax or Modem Tone Detection

Detection of fax and modem tones is available only with PerfectCall Call Analysis. Two tones are defined, TID_FAX1 and TID_FAX2. If either of them is detected while Frequency Detection and Cadence Detection are active, then Call Analysis is terminated with the reason CR_FAXTONE. **ATDX_CRTNID( )** identifies which fax or modem tone was detected.

No DX_CAP fields affect fax or modem tone detection.

2.6.5. Loop Current Detection

Some telephone systems return a momentary drop in loop current when a connection has been established (*answer supervision*). Loop Current Detection returns a *connect* when a transient loop current drop is detected.

In some environments, including most PBXs, answer supervision is not provided. In these environments, Loop Current Detection will not function. Check with your CO or PBX supplier to see if answer supervision based on loop current changes is available.

In some cases, the application may receive one or more transient loop current drops before an actual connection occurs. This is particularly true when dialing long-distance numbers, when the call may be routed through several different switches. Any one of these switches may be capable of generating a momentary drop in loop current.

To disable Loop Current Detection, set **ca_lcdly** to -1.

Loop Current Detection Parameters Affecting a Connect

To prevent detecting a connect prematurely or falsely, due to a spurious loop current drop, you can delay the start of Loop Current Detection by using the parameter **ca_lcdly**.

Loop Current Detection returns a *connect* after detecting a loop current drop. To allow the person who answered the phone to say "hello" before the application proceeds, you can delay the return of the *connect* by using the parameter **ca_lcdly1**.

Parameter	Description
ca_lcdly	Loop Current Delay: The delay after dialing has been completed and before beginning Loop Current Detection. To disable Loop Current Detection, set to -1. Default: 400 (10 ms units).
ca_lcdly1	Loop Current Delay 1: The delay after Loop Current Detection detects a transient drop in loop current and before Call Analysis returns a *connect* to the application. Default: 10 (10 ms units).

If the **ATDX_CONNTYPE( )** function returns CON_LPC, the connect was due to Loop Current Detection.

NOTE: When a connect is detected through Positive Voice Detection or Loop Current Detection, the DX_CAP parameters ca_hedge, ca_ansrdgl, and ca_maxansr are ignored.

2.6.6. Positive Voice Detection

Positive Voice Detection (PVD) can detect when a call has been answered by determining whether an audio signal is present that has the characteristics of a live or recorded human voice. This provides a very precise method for identifying when a connect occurs.

PVD is especially useful in those situations where answer supervision is not available for Loop Current Detection to identify a connect, and where the cadence is not clearly broken for Cadence Detection to identify a connect (for example, when the nonsilence of the cadence is immediately followed by the nonsilence of speech).

If the **ATDX_CONNTYPE( )** function returns CON_PVD, the connect was due to Positive Voice Detection.

NOTE: When a connect is detected through Positive Voice Detection or Loop Current Detection, the DX_CAP parameters ca_hedge, ca_ansrdgl, and ca_maxansr are ignored.

2.7. Call Analysis Errors

If **ATDX_CPTERM( )** returns CR_ERROR, you can use **ATDX_CPERROR( )** to determine the Call Analysis error that occurred.

#define	Error Description
CR_MEMERR	Out of memory trying to create temporary SIT tone templates (exceeds maximum number of templates).
CR_TMOUTON	Time-out waiting for an SIT tone.
CR_TMOUTOFF	SIT tone too long (exceeds a ca_mxtimefrq parameter).
CR_UNEXPTN	Unexpected SIT tone (the sequence of detected tones did not correspond to the SIT sequence).
CR_MXFRQERR	Invalid ca_mxtimefrq field in DX_CAP. If the

#define	Error Description
CR_MEMERR	Out of memory trying to create temporary SIT tone templates (exceeds maximum number of templates). ca_mxtimefrq parameter for each SIT tone is non-zero, it must have a value greater than or equal to the ca_timefrq parameter for the same SIT tone.
CR_UPFRQERR	Invalid upper frequency selection. This value must be non-zero for detection of any SIT tone.
CR_LGTUERR	Lower frequency greater than upper frequency.
CR_OVRLPERR	Overlap in selected SIT tones.

3. Caller ID

3.1. Caller ID Formats

An application can enable the Caller ID feature on specific channels to process Caller ID information as it is received with an incoming call. Caller ID is supported on selected boards only.

Caller ID formats currently supported are:

- Custom Local Area Signaling Services (CLASS) is a standard published by Bellcore:
  - Single Data Message (SDM) format
  - Multiple Data Message (MDM) format
- Analog Calling Line Identity Presentation (ACLIP) is a standard used in Singapore published by the Telecommunications Authority of Singapore:
  - Single Data Message Format (SDMF)
  - Multiple Data Message Format (MDMF)

  NOTE: Throughout the discussion of Caller ID, ACLIP SDMF is referred to as SDM, and ACLIP MDMF is referred to as MDM.
- Calling Line Identity Presentation (CLIP) is a standard used in the United Kingdom published by British Telecommunications (BT)
- Japanese Calling Line Identity Presentation (JCLIP) is a standard for "Number Display" used in Japan published by Nippon Telegraph and Telephone Corporation (NTT).

  NOTE: JCLIP operation requires that the Japanese country-specific parameter file be installed and configured (select Japan in the Dialogic country configuration).

Caller ID information is received from the CO between the first and second ring for CLASS and ACLIP, and before the first ring for CLIP. CLIP and ACLIP are supported by the D/41ESC boards only.

NOTE: One or more Caller ID information features may be available from the service provider or Central Office, whom you must contact to determine the available Caller ID options.

Dialogic supports the following CLASS and ACLIP Caller ID information:

- Single Data Message (SDM) format Caller ID information:
  - Frame header (indicating SDM format type)
  - Calling line's Directory Number (DN)
  - Date
  - Time
- Multiple Data Message (MDM) format Caller ID information:
  - Frame header (indicating MDM format type)
  - Calling line's Directory Number (DN)
  - Date
  - Time
  - Calling line's subscriber name
  - Calling line's DN (digits only)
  - Dialed Directory Number (digits only)
  - Reason why caller DN is not available
  - Indicate if the call is forwarded
  - Indicate if the call is long distance
  - Reason why calling subscriber name is not available

Dialogic supports the following CLIP Caller ID information::

- Calling line's Directory Number (DN)
- Date
- Time
- Calling line's subscriber name
- Calling line's DN (digits only)
- Dialed Directory Number (digits only)
- Reason why caller DN is not available
- Type of call (for example, voice, ring back when free, message waiting call)
- Network Message System status (number of messages waiting)
- Reason why calling subscriber name is not available

Dialogic supports the following JCLIP Caller ID information:

- Frame header (indicating MDM format type)
- Calling line's Directory Number (DN) (digits only)
- Dialed Directory Number (digits only)
- Reason why caller DN is not available
- Reason why calling subscriber name is not available

3.2. Accessing Caller ID Information

Applications using the Caller ID feature can process Caller ID information in the following ways:

- For CLASS or ACLIP, the Caller ID information is received from the service provider between the first and second ring. Set the ring event in the application to occur on or after the second ring. The ring event indicates reception of the CLASS or ACLIP Caller ID information from the CO.
- For CLIP or JCLIP, the Caller ID information is received from the service provider before the first ring. Set the ring event in the application to occur on or after the first ring. The ring event indicates reception of the CLIP Caller ID information from the CO.

The Caller ID information is available for the call from the moment the ring event is generated (if the ring event is set in your application as stated above) until one of the following occurs:

- If the call is answered (the application channel goes off-hook), the Caller ID information is available to the application until the call is disconnected (the application channel goes on-hook).
- If the call is not answered (the application channel remains on-hook), the Caller ID information is available to the application until rings are no longer received from the CO (signaled by ring off event, if enabled).

To determine if Caller ID information has been received from the CO, before issuing a **dx_gtcallid( )** or **dx_gtextcallid( )** Caller ID function, check the event data in the event block. When the ring event is received, the event data field in the event block is bitmapped and indicates that Caller ID information is available

when bit 0 (LSB) is set to 1. For details on the event block and function code examples, refer to the *Voice Software Reference: Programmer's Guide*.

NOTE: If the call is answered before the Caller ID information has been received from the CO, Caller ID information will not be available to the application.

If the call is not answered and the ring event is received before the Caller ID information has been received from the CO, Caller ID information will not be available until the beginning of the second ring (CLASS, ACLIP) or the beginning of the first ring (CLIP, JCLIP).

Based on the Caller ID options provided by the CO and for applications that require only the calling line Directory Number (DN), issue the **dx_gtcallid( )** function to get the calling line DN.

Based on the Caller ID options provided by the CO and for applications that require additional Caller ID information, issue the **dx_gtextcallid( )** function for each type of Caller ID message required. As an argument in the **dx_gtextcallid( )** function, the type of Caller ID message to access is specified (**infotype**).

The **dx_wtcallid( )** function is a Caller ID Convenience function provided to allow applications to wait for a specified number of rings (as set for the ring event) and returns the calling station's Directory Number (DN).

The **dx_wtcallid( )** function combines the functionality of the following:

- **dx_setevtmsk( )** voice function
- **dx_getevt( )** voice function
- **dx_gtcallid( )** Caller ID function

3.3. Error Handling

When the Caller ID function completes, check the return code.

- If the Caller ID function completes successfully, the buffer contains the Caller ID information.
- If the Caller ID function completes unsuccessfully, an error code is returned that indicates the reason for the error.

- When using the **dx_gtextcallid( )**, error codes depend upon the Message Type ID argument (**infotype**) passed to the function. All Message Types can produce an EDX_CLIDINFO error. Message Type CLIDINFO_CALLID can also produce EDX_CLIDOOA and EDX_CLIDBLK errors.

NOTE: The call is still active when an error is returned for a Caller ID function.

When using the **dx_gtcallid( )** Caller ID function, if an error is returned indicating the caller's phone number (DN) is blocked or out of area, other information such as date and time may be available by issuing the **dx_gtextcallid( )** Caller ID function. The information available, other than the caller's phone number, is determined by the CO.

3.4. Enabling Channels to Use the Caller ID Feature

During initialization, before the initial use of Caller ID functions, the application must enable the Caller ID feature on the channels requiring Caller ID. Caller ID is enabled by setting the following channel-based parameter using the Dialogic Voice library function **dx_setparm( )**.

Caller ID parameter for **dx_setparm( )**:

Parameter	Description
DXCH_CALLID	Enable or disable Caller ID for the channel as specified in **dx_setparm( )**. Valid values are: DX_CALLIDDISABLE (default) DX_CALLIDENABLE

NOTE: If Caller ID is enabled, on-hook detection (DTMF, MF and Global Tone Detection) will not function.

Refer to the *Voice Software Reference: Programmer's Guide* for function details.

3.5. Caller ID Demonstration Programs

The Caller ID feature is supported in the following Dialogic demonstration programs under the Dialogic home directory (normally *C:\Program Files\Dialogic*):

- SAMPLES\VOICE\SAMPLE.EXE (using the Dialogic API)
- SAMPLES\VOICE\TALKER32\TALKER32.EXE (using TAPI)

These demonstration programs require that the board support Caller ID and that the channel be connected to a PBX emulator that provides Caller ID simulation or to a CO line that has Caller ID services enabled.

3.6. Caller ID Technical Specifications

Before developing an application that requires CLASS, ACLIP or CLIP Caller ID information, contact your service provider and request the following appropriate detailed specifications:

Bellcore Documents for CLASS:
Information Exchange Management
Bellcore
445 South St., Room 2J-125
P. O. Box 1910
Morristown, NJ 07962-1910
Phone: 201-829-4785

- TR-NWT-000031 (issue 4)
  CLASS Feature: Calling Number Delivery
- TR-NWT-001188
  CLASS Feature: Calling Name Delivery Generic Requirements
- TR-NWT-000030 (issue 2)
  Voice Data Transmission Interface Generic Requirement

Telecommunications Authority of Singapore Document for ACLIP:
Telecommunications Authority of Singapore
TAS Building 1F
35 Robinson Road
Singapore 068876

- TAS TS PSTN1 A-CLIP: 1994
- Bellcore specification TR-NWT-000030 (see Bellcore address)

British Telecommunications Documents for CLIP:
British Telecommunications
Regulatory Services Unit
Room 134
2 City Forum
250-258 City Road
LONDON
EC1V 2TL

- SIN (Supplier Information Note) 242 (issue 01)
- SIN (Supplier Information Note) 227 (issue 01)

Nippon Telegraph and Telephone Documents for JCLIP:
Service Development Promotion Group
Network Service Department
Network Service Promotion Department
Business Commuications Headquarters
Nippon Telegraph and Telephone Corporation
Phone: (03) 5354-1257
E-mail: gisanshi@nws.hqs.ntt.co.jp

- Telephone Service Interfaces, Edition 5, Technical Reference

4. Global Dial Pulse Detection

4.1. Global Dial Pulse Detection Overview

Global Dial Pulse Detection (GDPD or Global DPD) allows applications to detect dial pulses from rotary or pulse phones and use them as if they were DTMF digits. GDPD is a software-based dial pulse detection method that can use country-specific parameters for extremely accurate performance. It provides the following features and benefits:

- Global DPD does not require a leading "0" to train the DPD algorithm. The algorithm is adaptive and can train on any DPD digit it encounters, with the greatest accuracy produced from training on a digit that has 5 or more pulses.
- Global DPD can be performed simultaneously with DTMF detection. The application can determine whether the digit detected is a DTMF or DPD digit.
- Global DPD can be performed simultaneously with Global Tone Detection (GTD). For example, the application can use GTD to monitor for disconnect tones (dial tone or busy) simultaneously with DPD.
- Global DPD supports pulse-digit cut-through during a voice playback, with the correct digit returned in the digit buffer. Global DPD uses echo cancellation, which provides more accurate reporting of digits during voice playback.
- The application can enable Global DPD and Volume Control. (Previously, there was a restriction that DPD digits had to be sent to the event queue instead of the digit queue if Volume Control was enabled.)

NOTE: Speed Control is not supported while the GDPD feature is enabled on the D/160SC-LS-IDPD, D/240SC-T1-IDPD, D/300SC-E1-75-IDPD, D/300SC-E1-120-IDPD and D/320SC-IDPD boards. You may adjust the speed before or after placing or receiving a call that uses GDPD. If any speed control adjustments are attempted while GDPD is enabled, the function will return with a -1 indicating failure.

Global DPD is supported in the SAMPLE demonstration program located in the SAMPLES\VOICE directory under the Dialogic home directory (normally *C:\Program Files\Dialogic*).

GDPD is supported on the following boards:

- D/41ESC
- ProLine/2V
- D/21H, D/41H
- VFX/40ESCplus
- D/160SC-LS-IDPD
- D/320SC-IDPD
- D/240SC-T1-IDPD
- D/300SC-E1-120-IDPD

Global DPD works only on DPD-enabled boards. You must order a separate GDPD enablement package from Dialogic to enable GDPD on the boards listed above (except for boards with the "IDPD" suffix, which are already enabled).

To indicate that a board is DPD-enabled, apply the sticker provided with the GDPD enablement package to your board. Additionally, it is recommended that you write down the serial number of the DPD-enabled board for your records.

GDPD is available on all boards supported except the D/21D, D/41D, DIALOG/4, VFX/40, VFX/40E, VFX/40SC and VFX/40ESC boards.

GDPD is supported in the SAMPLE demonstration program located in the SAMPLES\VOICE directory under the Dialogic home directory (normally C:\Program Files\Dialogic).

Global DPD supported applications are: Analog applications using the loop start telephone interface on the D/41ESC (and other low density boards) or D/160SC-LS-IDPD voice boards; and digital applications using the D/300SC-E1-75-IDPD or D/320SC-IDPD voice boards.

4.2. Regional DPD Parameters

One reason for calling this Dial Pulse Detection implementation Global DPD is that the detection algorithm supports 8 pulse-per-second to 22 PPS telephones. Dialogic is continuously qualifying its Dial Pulse Detection algorithm against dial pulse data collected from different parts of the world to improve DPD accuracy for the telephone systems and telephones in different regions. As appropriate, Dialogic will issue download parameters from time to time to improve the

accuracy of DPD in a given region of the world, whether it is part of a country, an entire country, or a group of countries.

Customized Global DPD download parameters are provided for several countries: Argentina, Brazil, Colombia, India, Japan, Mexico and Venezuela, one of which can be selected during Installation (refer to the Country Specific Parameter File). As additional regions are qualified for customized Global DPD, relevant region-specific support will be released.

Support for a generic 10 pulse-per-second (PPS) Global DPD parameter file is provided for countries that use 10 PPS phones.

You must install the Country-Specific Parameters and select a country to obtain support for Global DPD.

Refer to the *Voice Software Reference: Programmer's Guide* for programming information and sample code.

4.3. Programming Considerations for Accurate Global DPD

The Global DPD algorithm will accurately detect digits in the supported regions without requesting a special training digit from the caller or requiring any other restrictions on the application. However, keep the following considerations in mind when designing the application:

- Talk-off rejection (the ability of the algorithm to distinguish between dial pulses and the human voice) will improve after the first digit is detected.
- Digit detection is slightly more accurate (about 2%) after detecting a digit of '5' or greater. It is not necessary to dial a special training digit to do this. The application may simply restrict the first menu to digits 5, 6, 7, 8, 9, and 0, and the training will be complete. Subsequent menus may be unrestricted.
- In general, detection accuracy is greater for higher digits than lower. While detection accuracy is very high, it may be further improved by restricting, whenever convenient, menu selections to digits greater than '3'.

4.4. Global DPD Application Programming Interface

Global Dial Pulse Detection uses the same Application Programming Interface (API) model as the standard DPD interface.

The Global DPD feature must be implemented on a call-by-call basis. Global DPD must be enabled for each call by calling **dx_setdigtyp( )** for the Global DPD feature to work correctly.

For any digit detected, you can determine the digit type, DTMF, MF, GTD (user-defined) or DPD, by using the DV_DIGIT data structure in the application. When a **dx_getdig( )** call is performed, the digits are collected and transferred to the user's digit buffer. The digits are stored as an array inside the DV_DIGIT structure. This method allows you to determine very quickly whether a pulse or DTMF telephone is being used.

4.4.1. Dial Pulse Detection Digit Type Reporting

As shown in the following table, two defines are provided for identifying the Dial Pulse Detection digit type, depending upon how the digit type is retrieved.

#define	Digit Type
DG_DPD	Dial Pulse Detection digit from the DX_EBLK event queue data (**cst_data)** through a DE_DIGITS Call Status Transition event
DG_DPD_ASCII	Dial Pulse Detection digit from the DV_DIGIT **dg_type** digit buffer using **dx_getdig( )**

[JI5]Obtaining the digit type for DPD digits is valid only in the case when the voice and DPD capabilities are both present on the same board. In the case where a voice board does not support DPD, you cannot detect DPD digits or obtain the DPD digit type even though you can enable DPD and digit type reporting without an error.

4.4.2. Defines for Digit Type Reporting

Several can be used to obtain the DV_DIGIT **dg_type** (digit type) from the digit buffer. These defines contain "_ASCII" extensions as shown in the following table. If you get the digit from the DV_DIGIT **dg_type** digit buffer using **dx_getdig( )**, you should use the digit type define that has the "_ASCII" suffix. Otherwise, if you get the digit from the DX_EBLK event queue data (**cst_data**) through a DE_DIGITS Call Status Transition event, you should use the digit type define without the "_ASCII" suffix

	Defines for dg_type from	
Digit Type	**Digit Buffer**	**Event Queue**
DTMF	DG_DTMF_ASCII	DG_DTMF
DPD	DG_DPD_ASCII	DG_DPD
MF	DG_MF_ASCII	DG_MF
GTD (user defined)	DG_USER1_ASCII DG_USER2_ASCII DG_USER3_ASCII DG_USER4_ASCII DG_USER5_ASCII	DG_USER1 DG_USER2 DG_USER3 DG_USER4 DG_USER5

NOTE: Use the defines as shown above to determine the digit type from the value returned in the **dg_type** digit type field. If you get the digit from the digit buffer by using **dx_getdig( )**, you should use the digit type define that has the "_ASCII" extension. Otherwise, if you get the digit from the event queue through a DE_DIGITS Call Status Transition event, you should use the digit type define without the "_ASCII" extension.

4.4.3. GDPD Programming Procedure

1. Define a data structure of type DV_DIGIT (the DV_DIGIT structure is specified in the *DXDIGIT.H* file).
2. Enable DPD on the desired channels using the **dx_setdigtyp( )** function. For new calls you must use the D_DPDZ mask that initializes the DPD detector for new calls.

3. Execute the **dx_getdig( )** function to collect and transfer the digits to the user's digit buffer. The digits are stored in the **dg_value** field of the DV_DIGIT structure with the corresponding digit types stored in the **dg_type** field of the DV_DIGIT structure.

4.4.4. GDPD Programming Example

```
/*$ dx_setdigtyp( )and dx_getdig( ) example for Global Dial Pulse Detection $*/
#include        <stdio.h>
#include        "srllib.h"
#include        "dxxxlib.h"

void main(int argc, char **argv)
{

   int     dev;                    /* Dialogic device handle */
   DV_DIGIT        dig;
   DV_TPT tpt;

   /*
    * Open device, make or accept call
    */

   /* setup TPT to wait for 3 digits and terminate */
   dx_clrtpt(&tpt, 1);
   tpt.tp_type =   IO_EOT;
   tpt.tp_termno = DX_MAXDTMF;
   tpt.tp_length = 3;
   tpt.tp_flags =  TF_MAXDTMF;

   /* enable DPD and DTMF digits */
   dx_setdigtyp(dev, D_DPDZ|D_DTMF);

   /* clear the digit buffer */
   dx_clrdigbuf(dev);

   /* collect 3 digits from the user */
   if (dx_getdig(dev, &tpt, &dig, EV_SYNC) == -1) {
      /* error, display error message */
      printf("dx_getdig error %d, %s\n", ATDV_LASTERR(dev), ATDV_ERRMSGP(dev));
   } else {
      /* display digits received and digit type */
      printf("Received \"%s\"\n", dig.dg_value);
      printf("Digit type is ");
      /*
       * digit types have 0x30 ORed with them strip it off
       * so that we can use the DG_xxx equates from the header files
       */
      switch ((dig.dg_type[0] & 0x000f)) {
         case DG_DTMF:
            printf("DTMF\n");
            break;
         case DG_DPD:
            printf("DPD\n");
            break;
         default:
            printf("Unknown, %d\n", (dig.dg_type[0] &0x000f));
      }
```

```
}

/*
 * continue processing call
 */
```

5. Global Tone Detection/Generation

5.1. Overview

This chapter provides a description of Global Tone Detection (GTD) and Global Tone Generation (GTG). These features provide the ability to generate and detect single or dual frequency tones.

5.2. Global Tone Detection (GTD)

Global Tone Detection is a feature that allows a user to define the characteristics of a tone in order to detect a tone with the same characteristics. The characteristics of a tone are defined using GTD tone templates. The tone templates contain parameters that allow the user to assign frequency bounds and cadence components. Single and dual frequency tones are detected by comparing all incoming sounds to the GTD tone templates.

GTD operates on a channel-by-channel basis and is active when the channel is off-hook, unless the system uses a DTI/xxx, in which case, GTD is always active. GTD works simultaneously with DTMF and MF tone detection.

The driver responds to a detected tone by producing either a *tone event* on the event queue or a *digit* on the digit queue. The particular response depends upon the GTD tone configuration.

Use the Global Tone Detection functions to access tone templates and enable detection of single and dual frequency tones that fall outside those automatically provided with the Voice Driver. This includes tones outside the standard DTMF set of 0-9, a-d, *, and #, and the standard MF tones 0-9, *, and a-c.

5.2.1. Defining GTD Tones

GTD tones can have an associated ASCII digit (and digit type) specified using the **digit** and **digtype** parameters in the **dx_addtone( )** function. When the tone is detected, the digit is placed in the DV_DIGIT buffer and can be retrieved using

the **dx_getdig( )** function. When the tone is detected, either the tone event or the digit associated with the tone can be used as a termination condition to terminate I/O functions.

Termination conditions are set using the DV_TPT data structure. See the *Voice Software Reference: Programmer's Guide*for information about the DV_TPT data structure.

NOTE: If you want to terminate on multiple tones (or digits), you need to specify the terminating conditions for each tone in a separate DV_TPT data structure.

5.2.2. Building Tone Templates

When creating the tone template you can define the following:

- single or dual frequency (300-3500 Hz)
- optional ASCII digit associated with the tone template
- cadence components

Adding a tone template to a channel enables detection of a tone on that channel. Although only one tone template can be created at a time, multiple tone templates can be added to a channel. Each channel can have a different set of tone templates. Once created, tone templates can be selectively enabled or disabled.

NOTE: A particular tone template cannot be changed or deleted. A tone template can be disabled on a channel, but to delete a tone template, all tone templates on that channel must be deleted.

The following functions are used to define tone templates:

dx_bldst( )	• build a single frequency tone description
dx_blddt( )	• build a dual frequency tone description
dx_bldstcad( )	• build a single frequency tone cadence description
dx_blddtcad( )	• build a dual frequency tone cadence description
dx_setgtdamp( )	• set the GTD amplitude

NOTE: GTD build functions define new tone templates, and **dx_addtone( )** adds the tone templates to a channel.

Use **dx_addtone( )** to enable detection of the tone template on a channel. After building a tone template using a **dx_bld...( )** function, **dx_addtone( )** must be called to add this tone template to a channel. If the template is not added, the next call to a **dx_bld...( )** function will overwrite the tone definition contained in the previous template.

dx_bldst( ) defines a simple single frequency tone. Subsequent calls to **dx_addtone( )** will use this tone until another tone is defined. Thus, you can build a tone template and add it to several different channels.

dx_blddt( ) defines a simple dual frequency tone. Subsequent calls to **dx_addtone( )** will use this tone until another tone is defined. Thus, you can build a tone template and add it to several different channels.

NOTE: The D/41ESC, D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1, D/320SC, DTI/240SC, DTI/241SC, DTI/300SC, DTI/301SC, LSI/81SC, and LSI/161SC boards cannot detect dual tones with frequency components closer than 65 Hz. Use a single tone description to detect dual tones that are closer together than the ranges specified above.

dx_bldstcad( ) defines a simple single frequency cadence tone. Subsequent calls to **dx_addtone( )** will use this tone until another tone is defined. Thus, you can build a tone template and add it to several different channels. A single frequency cadence tone has single frequency signals with specific on/off characteristics.

dx_blddtcad( ) defines a simple dual frequency cadence tone. Subsequent calls to **dx_addtone( )** will use this tone until another tone is defined. Thus, you can build a tone template and add it to several different channels. A dual frequency cadence tone has dual frequency signals with specific on/off characteristics.

The minimum on- and off-time for cadence detection is 40 ms on and 40 ms off.

dx_setgtdamp( ) sets the amplitudes used by GTD. The amplitudes set using **dx_setgtdamp( )** will be the default amplitudes that will apply to all tones built using the **dx_bld...( )** functions. The amplitudes will remain valid for all tones built until **dx_setgtdamp( )** is called again and the amplitudes are changed.

The following table lists some standard Bell System Network Call Progress Tones. The frequencies are useful when creating the tone templates.

Tone	Frequency(Hz)	On Time (ms)	Off Time (ms)
Dial	350 + 440	Continuous	
Busy	480 + 620	500	500
Congestion (Toll)	480 + 620	200	300
Reorder (Local)	480 + 620	300	200
Ringback	440 + 480	2000	4000

5.2.3. Working with Tone Templates

Use the following functions to add/delete tone templates or to enable/disable tone detection:

- **dx_addtone( )** • add a tone template
- **dx_deltones( )** • delete tone templates
- **dx_distone( )** • disable detection of a tone
- **dx_enbtone( )** • enable detection of a tone

dx_addtone( ) adds the tone template that was defined by the most recent GTD build-tone function call to the specified channel. Adding a tone template to a channel downloads it to the board and enables detection of tone-on and tone-off events for that tone template.

CAUTIONS

1. Each tone template must have a unique identification.

2. Errors will occur if you use **dx_addtone( )** to change a tone template that has previously been added.

dx_deltones( ) removes all tone templates previously added to a channel with **dx_addtone( )**. If no tone templates were previously enabled for this channel, the function has no effect.

NOTE: **dx_deltones( )** does not affect tones defined by build-tone template functions and tone templates not yet defined. If you have added tones for PerfectCall Call Analysis, these tones are also deleted.

dx_distone( ) disables the detection of DX_TONEON and/or DX_TONEOFF events on a channel. Detection capability for user-defined tones is enabled on a channel by default when **dx_addtone( )** is called.

dx_enbtone( ) enables the detection of DX_TONEON and/or DX_TONEOFF events on a channel. Detection capability for tones is enabled on a channel by default when **dx_addtone( )** is called. The function can re-enable tones disabled by **dx_distone( )**.

DX_TONEON and DX_TONEOFF events are Call Status Transition (CST) events.

5.2.4. Tone Event Retrieval

To retrieve events, perform the following:

1. Call **dx_addtone( )** or **dx_enbtone( )** to enable tone-on/off detection.
2. Call **dx_getevt( )** to wait for CST event(s). Events are returned in the DX_EBLK structure.

NOTE: These procedures are the same as the retrieval of any other CST event, except that **dx_addtone( )** or **dx_enbtone( )** are used to enable event detection instead of **dx_setevtmsk( )**.

You can optionally specify an associated ASCII digit (and digit type) with the tone template. In this case, the tone template is treated like DTMF tones. When the digit is detected, it is placed in the digit buffer and can be used for termination. When an associated ASCII digit is specified, tone events will not be generated for that tone.

5.2.5. Maximum Number of Tone Templates

Guidelines for maximum number of tone templates are listed below and in *Table 2*:

- The D/21D, D/21E, D/21H and ProLine/2V boards allow a maximum of 33 tone templates each, which results in 16 tone templates per channel if apportioned evenly among the 2 channels.
- The D/4xD, D/41E, D/41ESC, D/41H and Dialog/4 boards allow a maximum of 33 tone templates each, which results in 8 tone templates per channel if apportioned evenly among the 4 channels.
- The D/81A board allows a maximum of 100 tone templates per board, which results in 12 tone templates per channel if apportioned evenly among the 8 channels.
- The D/12x board allows a maximum of 166 tone templates per board, which results in 13 tone templates per channel if apportioned evenly among the 12 channels.
- The D/160SC-LS board has a maximum of 240 tone templates per board or 15 per channel if evenly apportioned.
- The D/240SC and D/240SC-T1 boards have a maximum of 300 tone templates per board or 15 per channel if evenly apportioned.
- The D/300SC-E1 and D/320SC boards have a maximum of 450 tone templates per board or 15 per channel if evenly apportioned.
- The DTI/240SC and DTI/241SC boards have a maximum of 300 tone templates per board or 15 per channel if evenly apportioned.
- The DTI/300SC and DTI/301SC boards have a maximum of 450 tone templates per board or 15 per channel if evenly apportioned.
- The LSI/81SC and LSI/161SC boards have a maximum of 240 tone templates per board or 15 per channel if evenly apportioned.
- The maximum number of tone templates is based on tone templates that define a *dual* tone with a frequency detection range (bandwidth) of 62 Hz. (The detection range is the difference between the minimum and maximum defined frequencies for the tone.)

- If you use Call Analysis to identify the tri-tone Special Information Tone (SIT) sequences, Call Analysis will create tone templates internally, and this will reduce the number of tone templates that can be created by the user. Call Analysis creates one tone template for each single-frequency tone that is defined using DX_CAP. For example, if detecting the SIT tri-tone sequences per channel, the number of tone templates that can be defined by the user will be reduced by 3 per channel.
- If you create R2MF user-defined tones using **r2_creatfsig( )**, the voice boards will usually be able to create all 15 R2MF user-defined tones due to the overlap in frequencies for the R2MF signals. If these boards do not have sufficient memory, they may be able to support R2MF signaling through a reduced number of R2MF user-defined tones.
- If you use **dx_blddt( )** (or one of the other build tone functions) or **r2_creatfsig( )** to define a user-defined tone that alone or with existing user-defined tones exceeds the available memory, you will get an EDX_MAXTMPLT error.
- If you initiate Call Analysis and there is not enough memory to create the SIT tones internally, you will get a CR_MEMERR.
- The tone detection range should be limited to a maximum of 200 Hz per tone to reduce the chance of exceeding the available memory.

  NOTE: The **dx_deltones( )** function deletes all tone templates from a specified channel and releases the memory that was used for those tone templates. When an associated ASCII digit is specified, tone events will not be generated for that tone.

Table 2. Maximum Memory and Tone Templates (for Dual Tones)

Hardware	Tone Templates Per Board	Tone Templates Per Channel
D/21D	33	16
D/4xD	33	8
D/21E	33	16
D/41E	33	8
D/41ESC	33	8
D/81A	100	12
D/121A	166	13
D/121B	166	13
D/160SC-LS	240	15
D/240SC	300	15
D/240SC-T1	300	15
D/300SC-E1	450	15
D/320SC	450	15
DTI/240SC	300	15
DTI/241SC	300	15
DTI/300SC	450	15
DTI/301SC	450	15
LSI/81SC	240	15
LSI/161SC	240	15

5.2.6. Applications

Two sample applications for Global Tone Detection are described in this section. The first application describes how to use GTD to detect a fast-busy signal to determine when a disconnect occurs. The second application describes how to use GTD for leading edge detection of a tone using debounce time.

5.2.7. PerfectCall Disconnect Tone Supervision

PerfectCall provides disconnect tone supervision, which detects a disconnect tone that occurs after a party hangs up to end a connected call. This has been implemented in such a way that it can be used easily in an application that processes loop current drop events.

The disconnect tone is a PerfectCall tone definition (TID_DISCONNECT, tone ID 257) that contains the same default values as the TID_BUSY2 tone: Frequency 1: 500 Hz (deviation 200 Hz), Frequency 2: 525 Hz (deviation 175 Hz), On-time: 550 ms (deviation 400 ms), Off-time: 550 ms (deviation 400 ms), Repetition Count: 4.

The disconnect tone definition can be customized to the needs of your application the same as any PerfectCall tone. To enable disconnect supervision and use it with PerfectCall, use the following procedure:

1. Use the TSF Configuration application to enable the disconnect tone. The tone is enabled if a check mark is shown. (The setting of this parameter is stored in the Registry, and any change to it takes effect when the application is executed rather than when the boards are started.)

   Optionally, you can customize the disconnect tone definition by enabling and specifying in the TSF Configuration application Advanced Options window a PBXpert Tone Set File (if one is available). Or you can customize the disconnect tone through the API functions provided for that purpose: **dx_chgfreq( )**, **dx_chgdur( )**, and **dx_chgrepcnt( )**

2. Use the **dx_initcallp( )** function to initialize PerfectCall in your application as normal.

3. Use the **dx_setevtmsk( )** function with the DM_LCOFF parameter to enable detection of loop current drop events. This step is essential to obtain disconnect supervision.
4. Use the **dx_dial( )** function with Call Progress Analysis enabled to establish a connected call as normal.

   The TID_DISCONNECT tone is not used during Call Progress Analysis; it is automatically disabled until Call Progress Analysis ends. If a fast busy tone occurs during Call Progress Analysis, it is detected by the TID_BUSY2 tone and will terminate Call Progress Analysis.

   If the party hangs up and a disconnect tone is present within the parameters of the TID_DISCONNECT tone definition, PerfectCall detects the tone and generates a single loop current drop event (DX_LCOFF).

NOTE: If you enable the disconnect tone and loop current drop events as described above and perform a dial **without** Call Progress Analysis, a loop current drop event occurs if a fast busy tone is encountered during the dial.

Global Tone Disconnect Supervision

Global Tone Detection can be used for disconnect supervision. When a telephone call terminates, the central office may send a momentary drop in loop current to signal the disconnect. In configurations where the voice board is connected to a Private Branch Exchange (PBX), it is likely that there will be no drop in loop current for the voice board to detect. Instead, the PBX may initiate a fast-busy signal to indicate the disconnect. Global Tone Detection can be used to detect this fast-busy signal. Perform the following to detect the signal:

1. Determine the frequencies of the signal.
2. Characterize the on/off durations and tolerances of the signal cadence.
3. Use a build-tone function to define the characteristics of a single or dual tone with cadence in a tone template.
4. Use the **dx_addtone( )** function to add the GTD tone template for Global Tone Detection on each channel.

Leading Edge Detection Using Debounce Time

Rather than detecting a signal immediately, an application may want to wait for a period of time (debounce time) before the DX_TONEON event is generated indicating the detection of the signal. The **dx_bldstcad( )** and **dx_blddtcad( )** functions can detect leading edge debounce on-time. A tone must be present at a given frequency and for a period of time (debounce time) before a DX_TONEON event is generated. The debounce time is specified using the ontime and ondev parameters in the **dx_bldstcad( )** or **dx_blddtcad( )** functions.

To use this application specify the following values for the **dx_bldstcad( )** or **dx_blddtcad( )** function parameters listed below:

Parameter	Description
ontime	1/2 of the desired debounce time
ondev	-1/2 of the desired debounce time
offtime	0
offdev	0
repcnt	0

NOTE: This application cannot work with the functions **dx_blddt( )** and **dx_bldst( )** since these functions do not have timing field parameters.

5.3. Global Tone Generation (GTG)

Global Tone Generation enables the creation of user-defined tones. The Tone Generation template, TN_GEN, is used to define the tones with the following information:

- Single or dual tone
- Frequency fields
- Amplitude for each frequency
- Duration of tone

5.3.1. Global Tone Generation Functions

The following functions are used to generate tones:

dx_bldtngen( )	• build a tone generation template
dx_playtone( )	• play a tone

dx_bldtngen( ) is a convenience function that sets up the tone generation template data structure (TN_GEN) by allowing the assignment of specified values to the appropriate fields. The tone generation template is placed in the user's return buffer and can then be used by the **dx_playtone( )** function to generate the tone.

dx_playtone( ) plays a tone specified by the tone generation template (pointed to by **tngenp**). Termination conditions are set using the DV_TPT structure. The reason for termination is returned by the **ATDX_TERMMSK( )** function. **dx_playtone( )** returns a 0 to indicate that it has completed successfully.

5.3.2. Building and Implementing a Tone Generation Template

The tone generation template defines the frequency, amplitude, and duration of a single or dual frequency tone to be played. You can use the convenience function **dx_bldtngen( )** to set up the structure. Use **dx_playtone( )** to play the tone.

The TN_GEN data structure is shown below:

```
typedef struct {
   unsigned short tg_dflag;        /* dual tone - 1, single tone - 0 */
   unsigned short tg_freq1;        /* frequency of tone 1 (in Hz) */
   unsigned short tg_freq2;        /* frequency of tone 2 (in Hz) */
   short int      tg_ampl1;        /* amplitude of tone 1 (in dB) */
   short int      tg_ampl2;        /* amplitude of tone 2 (in dB) */
   short int      tg_dur;          /* duration (in 10 ms) */
} TN_GEN;
```

After you build the TN_GEN data structure, there are two ways to define each tone template:

1. Include the values in the structure, or
2. Pass the values to TN_GEN using the **dx_bldtngen( )** function.

After defining the template, pass TN_GEN to **dx_playtone( )** to play the tone.

If you include the values in the structure, you must create a structure for each tone template. If you pass the values using the **dx_playtone( )** function, then you can reuse the structure. If you are only changing one value in a template with many variables, it may be more convenient to use several structures in the code instead of reusing just one.

5.4. Cadenced Tone Generation

Cadenced tone generation is an enhancement to Global Tone Generation that enables you to generate a signal with up to 4 single- or dual-tone elements, each with its own on/off duration creating the signal pattern or cadence.

Cadenced tone generation is accomplished with the **dx_playtoneEx( )** function and the TN_GENCAD data structure.

You can define your own custom cadenced tone or take advantage of the built-in set of standard PBX Call Progress Signals.

5.4.1. How To Generate a Custom Cadenced Tone

A custom cadenced tone is defined by specifying in a TN_GENCAD data structure the repeating elements of the signal (the cycle) and the number of desired repetitions.

The cycle can consist of up to 4 segments, each with its own tone definition and cadence. A segment consists of a TN_GEN single- or dual-tone definition (frequency, amplitude, & duration) followed by a corresponding off-time (silence duration) that is optional. The **dx_bldtngen( )** function can be used to set up the TN_GEN components of the TN_GENCAD structure. The tone duration, or on-time, from TN_GEN (**tg_dur**) and the **offtime** from TN_GENCAD are combined to produce the cadence for the segment. The segments are seamlessly concatenated in ascending order to generate the signal cycle.

Use the following procedure to generate a custom cadenced tone.

1. Identify the repeating elements of the signal (the cycle).
2. Use a TN_GENCAD structure to define the segments in the cycle:

- Start with the first tone element in the cycle and identify the single- or dual-tone frequencies, amplitudes, and duration (on-time).
- Use the **dx_bldtngen( )** function to specify this tone definition in **tone[0]** (the first TN_GEN tone array element) of the TN_GENCAD structure.
- Identify the off-time for the first tone element and specify it in **offtime[0]**. If the first tone element is followed immediately by a second tone element, set **offtime[0]** = 0.
- Define the next segment of the cycle in **tone[1]** and **offtime[1]** the same way as above, and so on, up to the maximum of 4 segments or until you reach the end of the cycle.

3. Use the TN_GENCAD to define the parameters of the cycle:
   - Specify the number of segments in the cycle (**numsegs**).
   - Specify the number of cycle repetitions (**cycles**).
4. Set the termination conditions in a DV_TPT structure.
5. Call the **dx_playtoneEx( )** function and use the **tngencadp** parameter to specify the custom cadenced tone that you defined in the TN_GENCAD.

For an illustration of this procedure, see *Figure 13. Example of Custom Cadenced Tone Generation*.

Define the Signal as a Dialogic Custom Cadence Tone

■ ***Set the TN_GENCAD Parameters***

```
tngencad.cycles = 255;
tngencad.numsegs = 2;
tngencad.offtime[0] = 0;
tngencad.offtime[1] = 300;
dx_bldtngen(&tngencad.tone[0],440,480,-16,-16,100)
dx_bldtngen(&tngencad.tone[1],440,0,-16,0,20)
```

■ ***Call the dx_playtoneEx( ) Function***

```
dx_playtoneEx(dxxxdev,&tngencad,tpt,EV_SYNC)
```

■ ***The TN_GENCAD Definition and Resulting Signal***

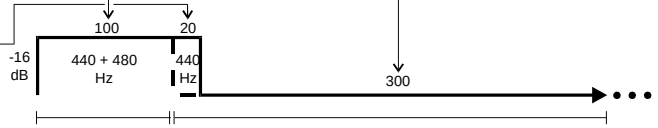
```
TN_GENCAD tngencad
 cycles      = 255
 numsegs     =   2
 offtime[0]  =   0
 offtime[1]  = 300
 offtime[2]  =   0
 offtime[3]  =   0

TN_GEN tone[0]
 dflag[0]    =   0
 tg_freq1[0] = 440
 tg_freq2[0] = 480
 tg_ampl1[0] = -16
 tg_ampl2[0] = -16
 tg_dur[0]   = 100

TN_GEN tone[1]
 dflag[1]    =   0
 tg_freq1[1] = 440
 tg_freq2[1] =   0
 tg_ampl1[1] = -16
 tg_ampl2[1] =   0
 tg_dur[1]   =  20

TN_GEN tone[2]
 dflag[2]     = 0
 tg_freq1[2]  = 0
 tg_freq2[2]  = 0
 tg_ampl1[2]  = 0
 tg_ampl2[2]  = 0
 tg_dur[2]    = 0

TN_GEN tone[3]
 dflag[3]     = 0
 tg_freq1[3]  = 0
 tg_freq2[3]  = 0
 tg_ampl1[3]  = 0
 tg_ampl2[3]  = 0
 tg_dur[3]    = 0
```



Segment 1

440 + 480 Hz dual tone at -16 dB with on-time of 100 (10 ms units) and *no off-time.*

Segment 2

Single tone of 440 Hz at -16 dB with on-time of 20 (10 ms units) and off-time of 300 (or 3 seconds).

Cycle: 2 segments repeating indefinitely, or until a **tpt** termination occurs.

Description of the Cadence Tone Used in this Example

Signal Description: Repetition of combined tones (440 + 480 Hz) ON for 0.8 to 1.2 seconds, followed by 440 Hz tone ON for 0.2 seconds, and tone OFF for 2.7 to 3.3 seconds applied within a power range of -14.5 to -17.5 dBm.

Note: Dialogic provides a predefined set of standard call progress signals that can be generated by **dx_playtoneEx( )**. This example shows how you would define the Special Audible Ring Tone 1 as a custom cadence tone if it were not already in the standard set as CP_RINGBACK1_CALLWAIT.

Figure 13. Example of Custom Cadenced Tone Generation

5.4.2. How To Generate a Non-Cadenced Tone

Either the **dx_playtoneEx( )** or the **dx_playtone( )** function can generate a non-cadenced tone.

- See the **dx_playtone( )** function for information on how to generate a non-cadenced signal. The following non-cadenced call progress signals could be generated by the **dx_playtone( )** function if you defined them in a TN_GEN: Dial Tone, Executive Override Tone, and Busy Verification Tone Part A.
- The **dx_playtoneEx( )** can also generate a non-cadenced tone by using a TN_GENCAD that defines a single segment.

  If you want to generate a continuous, non-cadenced signal, use a single segment and zero off-time, and specify 1) an infinite number of cycles, 2) an infinite on-time, or 3) both. (You must also specify the appropriate termination conditions in a DV_TPT structure or else the tone will never end). For example:

```
cycles = 255;
numsegs = 1;
offtime[0] = 0;
tone[0].tg_dur = -1
```

5.4.3. TN_GENCAD Data Structure - Cadenced Tone Generation

TN_GENCAD is a voice library data structure (*DXXXLIB.H*) that defines a cadenced tone that can be generated by using the **dx_playtoneEx( )** function.

The TN_GENCAD data structure contains a **tone** array with 4 elements that are TN_GEN data structures (Tone Generation Templates). For details on TN_GEN and TN_GENCAD, see the chapter on data structures in the *Voice Software Reference: Programmer's Guide*.

For examples of TN_GENCAD, refer to the Dialogic definitions of the standard call progress signals (*Table 4. TN_GENCAD Definitions for Standard PBX Call Progress Signals*).

5.4.4. How To Generate a Standard PBX Call Progress Signal

Use the following procedure to generate a standard PBX call progress signal from the predefined set of standard PBX call progress signals:

1. Select a call progress signal from ??Table 2 and note the signal ID (see also *Figure 14. Standard PBX Call Progress Signals*).
2. Set the termination conditions in a DV_TPT structure.
3. Call the **dx_playtoneEx( )** function and specify the signal ID for the **tngencadp** parameter. For example:

```
dx_playtoneEx(dxxxdev, CP_BUSY, tpt, EV_SYNC)
```

5.4.5. Predefined Set of Standard PBX Call Progress Signals

The following information is describes the predefined set of standard PBX call progress signals that are provided by Dialogic:

- *Table 3. Standard PBX Call Progress Signals* lists the predefined, standard, call progress signals and their signal IDs. The signal IDs can be used with the **dx_playtoneEx( )** function to generate the signal. (The #defines for the signal IDs are located in the *DXXXLIB.H* file.)
- *Figure 14. Standard PBX Call Progress Signals* illustrates the signals along with their tone specifications and cadences. The signals were divided into two parts so they could be illustrated to scale while providing sufficient detail. Part 1 uses a smaller scale than Part 2. (For this reason, the order of the signals is different than in the tables.)
- *Table 4. TN_GENCAD Definitions for Standard PBX Call Progress Signals* lists the TN_GENCAD definitions of the signal cycle and segment definitions for each predefined call progress signal. These definitions are located in the *DXGTD.C* file.
- *Section 5.4.6. Important Considerations for Using the Predefined Call Progress Signals* describes what standard was used, how the standard was implemented, information regarding the signal power levels, usage and other considerations.

Table 3. Standard PBX Call Progress Signals

Name	**Meaning**	**Signal ID (***tngencadp)*
Dial Tone	Ready for dialing	CP_DIAL
Reorder Tone (Paths-Busy, All-Trunks-Busy, Fast Busy)	Call blocked: resources unavailable	CP_REORDER
Busy Tone (Slow Busy)	Called line is busy	CP_BUSY
Audible Ring Tone 1 (Ringback Tone)	Called party is being alerted	CP_RINGBACK1
Audible Ring Tone 2 [1] (Slow Ringback Tone)	Called party is being alerted	CP_RINGBACK2
Special Audible Ring Tone 1 [1]	Called party has Call Waiting feature and is being alerted	CP_RINGBACK1_CALLWAIT
Special Audible Ring Tone 2 [1]	Called party has Call Waiting feature and is being alerted	CP_RINGBACK2_CALLWAIT
Recall Dial Tone	Ready for additional dialing on established connection	CP_RECALL_DIAL
Intercept Tone	Call blocked: invalid	CP_INTERCEPT
Call Waiting Tone 1 [2]	Call is waiting: single burst	C_CALLWAIT1
Call Waiting Tone 2 [2]	Call is waiting: double burst	CP_CALLWAIT2
Busy Verification Tone (Part A)	Alerts parties that attendant is about to enter connection	CP_BUSY_VERIFY_A
Busy Verification Tone (Part B)	Attendant remains connected	CP_BUSY_VERIFY_B

Table 3. Standard PBX Call Progress Signals - Continued

Name	**Meaning**	**Signal ID (***tngencadp)*
Executive Override Tone	Overriding party about to be bridged onto connection	CP_EXEC_OVERRIDE
Confirmation Tone	Feature has been activated or deactivated	CP_FEATURE_CONFIRM
Stutter Dial Tone (Message Waiting Dial Tone)	Message waiting; ready for dialing	CP_STUTTER_DIAL or CP_MSG_WAIT_DIAL

[1] Either of the two Audible Ring Tones can be used but are not intended to be intermixed in a system. When using the *Special* Audible Ring Tone (1 or 2), it should correspond to the Audible Ring Tone (1 or 2) that is used.

[2] The two Call Waiting Tones (1 & 2) can be used to differentiate between internally and externally originated calls. Call Waiting Tone 2 is defined as a double burst in this implementation, although “multiple” bursts are permissible.

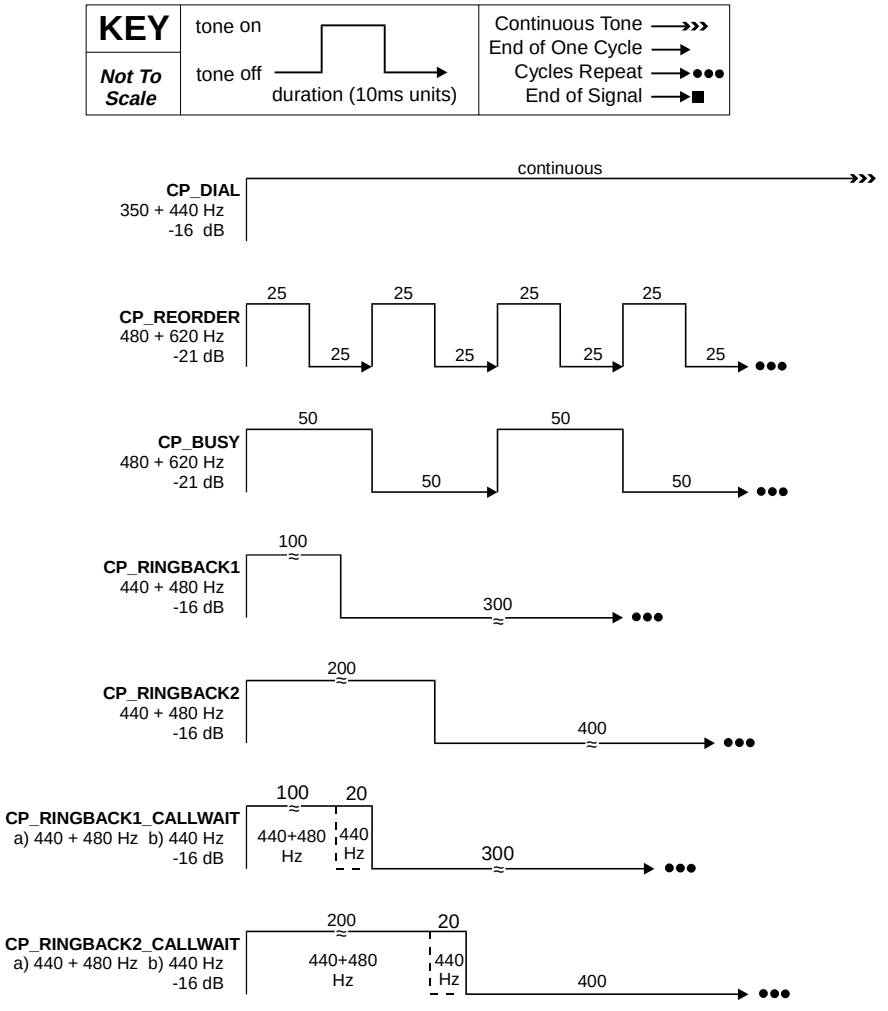


Figure 14. Standard PBX Call Progress Signals

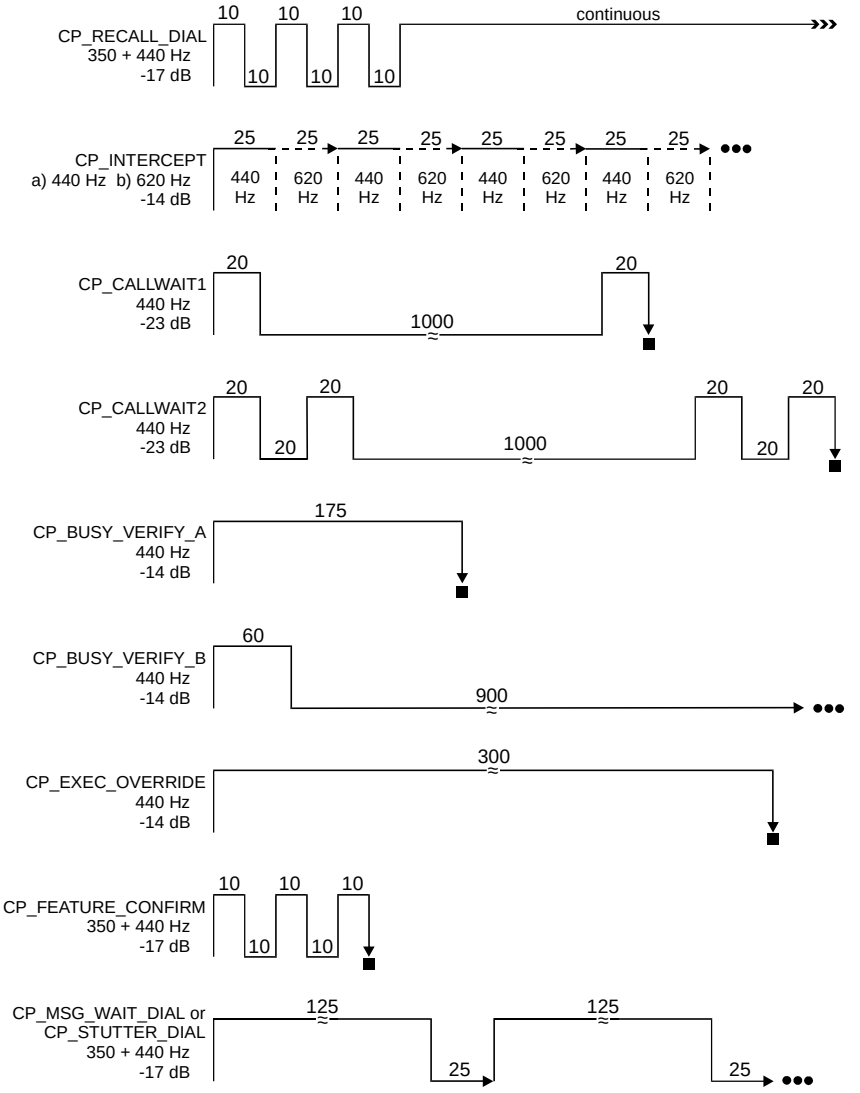


Figure 14. Standard PBX Call Progress Signals — continued

Table 4. TN_GENCAD Definitions for Standard PBX Call Progress Signals

SIGNAL_ID							
Cycle Definition		**Segment Definitions**					
Number of Cycles[1]	**Number of Segments in Cycle**	**Frequency #1 (Hz)**	**Frequency #2 (Hz)**	**Amplitude #1 (dB)**	**Amplitude #2 (dB)**	**On-Time[2] (10 ms)**	**Off-Time (10 ms)**
cycles	*numsegs*	*tg_freq1*	*tg_freq2*	*tg_ampl1*	*tg_ampl2*	*tg_dur*	*offtime*
CP_DIAL							
1	1	350	440	-17	-17	-1	0
CP_REORDER							
255	1	480	620	-21	-21	25	25
CP_BUSY							
255	1	480	620	-21	-21	50	50
CP_RINGBACK1							
255	1	440	480	-16	-16	100	300
CP_RINGBACK2							
255	1	440	480	-16	-16	200	400
CP_RINGBACK1_CALLWAIT							
255	2	440 440	480 0	-16 -16	-16 0	100 20	0 300
CP_RINGBACK2_CALLWAIT							
255	2	440 440	480 0	-16 -16	-16 0	200 20	0 400
CP_RECALL_DIAL							
1	4	350 350 350 350	440 440 440 440	-17 -17 -17 -17	-17 -17 -17 -17	10 10 10 -1	10 10 10 0
CP_INTERCEPT							
255	2	440 620	0 0	-14 -14	0 0	25 25	0 0

***Table 4. TN_GENCAD Definitions for Standard PBX Call Progress Signals* - Continued**

SIGNAL_ID							
Cycle Definition		**Segment Definitions**					
Number of Cycles[1]	**Number of Segments in Cycle**	**Frequency #1 (Hz)**	**Frequency #2 (Hz)**	**Amplitude #1 (dB)**	**Amplitude #2 (dB)**	**On-Time[2] (10 ms)**	**Off-Time (10 ms)**
cycles	*numsegs*	*tg_freq1*	*tg_freq2*	*tg_ampl1*	*tg_ampl2*	*tg_dur*	*offtime*
CP_CALLWAIT1							
1	2	440 440	0 0	-23 -23	0 0	20 20	1000 0
CP_CALLWAIT2							
1	4	440 440 440 440	0 0 0 0	-23 -23 -23 -23	0 0 0 0	20 20 20 20	20 1000 20 0
CP_BUSY_VERIFY_A							
1	1	440	0	-14	0	175	0
CP_BUSY_VERIFY_B							
255	1	440	0	-14	0	60	900
CP_EXEC_OVERRIDE							
1	1	440	0	-14	0	300	0
CP_FEATURE_CONFIRM							
1	3	350 350 350	440 440 440	-17 -17 -17	-17 -17 -17	10 10 10	10 10 0
CP_STUTTER_DIAL or **CP_MSG_WAIT_DIAL**							
255	1	350	440	-17	-17	125	25

[1] 255 specifies an infinite number of cycles (**cycles**)
[2] -1 specifies an infinite tone duration (**tg_dur**)

5.4.6. Important Considerations for Using the Predefined Call Progress Signals

1. Signal definitions are based on the TIA/EIA Standard: Requirements for Private Branch Exchange (PBX) Switching Equipment, TIA/EIA-464-B, April 1996 (Telecommunications Industry Association in association with the Electronic Industries Association, Standards and Technology Department, 2500 Wilson Blvd., Arlington, VA 22201). To order copies, contact Global Engineering Documents in the USA at 1-800-854-7179 or 1-303-397-7956.
2. A separate Line Lockout Warning Tone, which indicates that the station line has been locked out because dialing took too long or the station failed to disconnect at the end of a call, is not necessary and is not recommended. You can use the Reorder tone over trunks; or the Intercept, Reorder, or Busy tone over stations.
3. For signals that specify an infinite repetition of the signal cycle (**cycles** = 255) or an infinite duration of a tone (**tg_dur** = -1), you must specify the appropriate termination conditions in the DV_TPT structure used by **dx_playtoneEx( )**.
4. There may be more than one way to use TN_GENCAD to generate a given signal. For example, the 3 bursts of the Confirmation Tone can be created through one cycle containing 3 segments (as in the Dialogic implementation) or through a single segment that is repeated in 3 cycles.
5. To generate a continuous, non-cadenced signal , you can use **dx_playtoneEx( )** and TN_GENCAD to specify a single segment with zero off-time and with an infinite number of cycles and/or an infinite on-time.

   Alternatively, you could use **dx_playtone( )** and TN_GEN to generate a non-cadenced signal. The following non-cadenced call progress signals could be generated by the **dx_playtone( )** function if you defined them in a TN_GEN: 1) Dial Tone, 2) Executive Override Tone, and 3) Busy Verification Tone Part A.
6. Note that the Intercept Tone consists of alternating single tones.
7. Although the TIA/EIA Standard describes the Busy Verification Tone as one signal, the 2 segments are separate tones/events: Part A is a single burst almost 3 times longer than Part B and it alerts the parties before the attendant intrudes; Part B is a short burst every 9 seconds continuing as long as the interruption lasts. The TIA/EIA Standard does not define an off-time between

Part A and B. Therefore, the application developer is responsible for implementing the timing between the two parts of this signal.

8. The TIA/EIA Standard specifies the range of permissible power levels per frequency for 1) the Central Office trunk interface and 2) all other interfaces (including off-premise stations and tie trunks). The Dialogic implementation adopted the approximate middle value in the acceptable range of power levels for applying the signals to the CO trunk interface. These power levels were more restrictive than those for the other interfaces. According to the following statement in the TIA/EIA Standard, additional requirements and considerations may apply:

   "Studies have shown that the lower level tones that are transmitted over trunks should be 6 dB hotter at the trunk interface (than at the line interface) to compensate for increased loss on the end-to-end connection. In the case of tones used at higher levels, the 6-dB difference is not used since power at trunk interfaces must be limited to -13 dBm0 total when averaged over any 3-second interval to prevent carrier overload. Maximum permissible powers listed are consistent with this requirement taking into account the allowable interruption rates for the various tones. Uninterrupted tones, such as Dial Tone and Intercept Tone, shall be continuously limited to -13 dBm."

6. R2MF Signaling

6.1. Overview

This chapter provides a description of R2MF signaling protocol and the use of R2MF signaling with voice boards. R2MF support is available on all voice boards.

6.2. R2MF Overview

R2MF signaling is an international signaling system that is used in Europe and Asia to permit the transmission of numerical and other information relating to the called and calling subscribers' lines.

R2MF signals that control the call set-up are referred to as *interregister signals*. In the case of the signals sent between the central office (CO) and the customer premises equipment (CPE), the CO is referred to as the *outgoing register* and the CPE as the *incoming register*. Signals sent from the CO are *forward* signals; signals sent from the CPE are *backward* signals. The outgoing register (CO) sends forward interregister signals and receives backward interregister signals. The incoming register (CPE) receives forward interregister signals and sends backward interregister signals. See *Figure 15* .

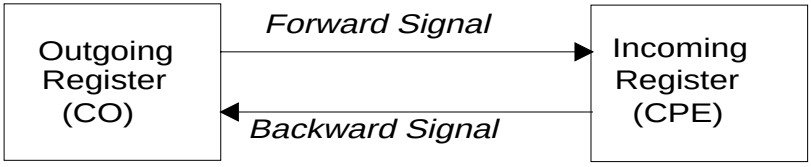


Figure 15. Forward and Backward Interregister Signals

The focus of this section is on the forward and backward interregister signals, and more specifically, the *address* signals, that provide the telephone number of the called subscriber's line. For national traffic, the address signals can also provide the telephone number of calling subscriber's line for automatic number identification (ANI) applications.

R2MF signals that are used for supervisory signaling on the network are called *line signals*. Line signals are beyond the scope of this document.

6.2.1. Direct Dialing-In Service

Since R2MF address signals can provide the telephone number of the called subscriber's line, the signals may be used by applications providing direct dialing-in (DDI) service, also known as direct inward dialing (DID), and dialed number identification service (DNIS).

DDI service allows an outside caller to dial an extension within a company without requiring an operator to transfer the call. The central office (CO) passes the last 2, 3, or 4 digits of the dialed number to the customer premises equipment (CPE) and the CPE completes the call.

6.2.2. R2MF Multifrequency Combinations

R2MF signaling uses a multifrequency code system based on six fundamental frequencies in the forward direction (1380, 1500, 1620, 1740, 1860, and 1980 Hz) and six frequencies in the backward direction (1140, 1020, 900, 780, 660, and 540 Hz).

Each signal is composed of two out of the six fundamental frequencies, which results in 15 different tone combinations in each direction. Although R2MF is designed for operation on international networks with 15 multifrequency combinations in each direction, in national networks it can be used with a reduced number of signaling frequencies (for example, 10 multifrequency combinations). See the following tables for a list of the signal tone pairs:

- *Table 5. Forward Signals CCITT Signaling System R2MF Tones*
- *Table 6. Backward Signals CCITT Signaling System R2MF Tones*

Table 5. Forward Signals CCITT Signaling System R2MF Tones

R2MF TONES			DIALOGIC INFORMATION		
Tone Number	**Tone Pair Frequencies (Hz)**		**Group I Define**	**Group II Define**	**Tone Detect. ID**
1	1380	1500	SIGI_1	SIGII_1	101
2	1380	1620	SIGI_2	SIGII_2	102
3	1500	1620	SIGI_3	SIGII_3	103
4	1380	1740	SIGI_4	SIGII_4	104
5	1500	1740	SIGI_5	SIGII_5	105
6	1620	1740	SIGI_6	SIGII_6	106
7	1380	1860	SIGI_7	SIGII_7	107
8	1500	1860	SIGI_8	SIGII_8	108
9	1620	1860	SIGI_9	SIGII_9	109
10	1740	1860	SIGI_0	SIGII_0	110
11	1380	1980	SIGI_11	SIGII_11	111
12	1500	1980	SIGI_12	SIGII_12	112
13	1620	1980	SIGI_13	SIGII_13	113
14	1740	1980	SIGI_14	SIGII_14	114
15	1860	1980	SIGI_15	SIGII_15	115

Table 6. Backward Signals CCITT Signaling System R2MF Tones

R2MF TONES			DIALOGIC INFORMATION	
Tone Number	**Tone Pair Frequencies (Hz)**		**Group A Define**	**Group B Define**
1	1140	1020	SIGA_1	SIGB_1
2	1140	900	SIGA_2	SIGB_2
3	1020	900	SIGA_3	SIGB_3
4	1140	780	SIGA_4	SIGB_4
5	1020	780	SIGA_5	SIGB_5
6	900	780	SIGA_6	SIGB_6
7	1140	660	SIGA_7	SIGB_7
8	1020	660	SIGA_8	SIGB_8
9	900	660	SIGA_9	SIGB_9
10	780	660	SIGA_0	SIGB_0
11	1140	540	SIGA_11	SIGB_11
12	1020	540	SIGA_12	SIGB_12
13	900	540	SIGA_13	SIGB_13
14	780	540	SIGA_14	SIGB_14
15	660	540	SIGA_15	SIGB_15

6.2.3. R2MF Signal Meanings

There are 2 groups of meanings associated with each set of signals. Group I meanings and Group II meanings are associated with the 15 forward signals. Group A meanings and Group B meanings are associated with the 15 backward signals. See *Figure 16* .

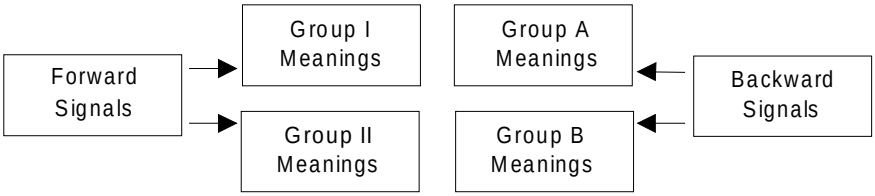


Figure 16. Multiple Meanings for R2MF Signals

In general, Group I forward signals and Group A backward signals are used to control the call set-up and to transfer address information between the outgoing register (CO) and the incoming register (CPE). The incoming register can then signal to the outgoing register to change over to the Group II and Group B meanings.

Group II forward signals provide the calling party's category, and Group B backward signals provide the condition of the called subscriber's line. For further information, see *Table 7* describing the purpose of the signal groups and the changeover in meanings.

Signaling must always begin with a Group I forward signal followed by a Group A backward signal that serves to acknowledge the signal just received and also has its own meaning. Each signal then requires a response from the other party. Each response becomes an acknowledgement of the event and an event for the other party to respond to.

Backward signals serve to indicate certain conditions encountered during call set-up or to announce switch-over to changed meanings of subsequent backward signals. Changeover to Group II and Group B meanings allows information about the state of the called subscriber's line to be transferred.

Table 7. Purpose of Signal Groups and Changeover in Meaning

Signal	Purpose	
Group I	Group I signals control the call set-up and provide address information.	
Group A	Group A signals acknowledge Group I signals (see exception under signal A-5 below) for call set-up, and can also request address and other information. Group A signals also control the changeover to Group II and Group B meanings through the following signals:	
	A-3	Address Complete — Changeover to Reception of Group B Signals: Indicates the address is complete and signals a changeover to Group II/B meanings; after signal A-3 is sent, signaling cannot change back to Group I/A meanings.
	A-5	Send Calling Party's Category: Requests transmission of a single Group II signal providing the calling party's category. Signal A-5 requests a Group II signal but does not indicate changeover to Group B signals. When the Group II signal requested by A-5 is received, it is acknowledged by a Group A signal; this is an exception to the rule that Group A signals acknowledge Group I signals.
Group II	Group II signals acknowledge signal A-3 or A-5 and provide the calling party category (national or international call, operator or subscriber, data transmission, maintenance or test call).	
Group B	Group B signals acknowledge Group II signals and provide the condition of the called subscriber's line. Before Group B signals can be transmitted, the preceding backward signal must have been A-3. Signals cannot change back to Group I/A.	

The Incoming Register Backward Signals Can Request:

- Transmission of address

  - Send next digit
  - Send digit previous to last digit sent
  - Send second digit previous to last digit sent
  - Send third digit previous to last digit sent
- Category of the call (the nature and origin)
  - National or international call
  - Operator or subscriber
  - Data transmission
  - Maintenance or test call
- Whether or not the circuit includes a satellite link
- Country code and language for international calls
- Information on use of an echo suppressor

The Incoming Register Backward Signals Can Indicate:

- Address complete — send category of call
- Address complete — put call through
- International, national, or local congestion
- Condition of subscriber's line
  - Send SIT to indicate long term unavailability
  - Line busy
  - Unallocated number
  - Line free — charge on answer
  - Line free — no charge on answer (only for special destinations)
  - Line out of order

NOTE: The meaning of certain forward multifrequency combinations may also vary depending upon their position in the signaling sequence. For example, with terminal calls the first forward signal transmitted in international working is a language or discriminating digit (signals I-1 through I-10). When the same signal is sent as other than the first signal, it usually means a numerical digit.

See the following tables for the signal meanings:

- *Table 8. Meanings for R2MF Group I Forward Signals*

Table 8. Meanings for R2MF Group I Forward Signals

Tone Number	Dialogic Define	(A) Primary Meaning (B) Secondary Meaning
1	SIGI_1	**(A)** Digit 1 **(B)** Language digit-French
2	SIGI_2	**(A)** Digit 2 **(B)** Language digit-English
3	SIGI_3	**(A)** Digit 3 **(B)** Language digit-German
4	SIGI_4	**(A)** Digit 4 **(B)** Language digit-Russian
5	SIGI_5	**(A)** Digit 5 **(B)** Language digit-Spanish
6	SIGI_6	**(A)** Digit 6 **(B)** Spare (language digit)
7	SIGI_7	**(A)** Digit 7 **(B)** Spare (language digit)
8	SIGI_8	**(A)** Digit 8 **(B)** Spare (language digit)
9	SIGI_9	**(A)** Digit 9 **(B)** Spare (discriminating digit)
10	SIGI_0	**(A)** Digit 0 **(B)** Discriminating digit
11	SIGI_11	**(A)** Access to incoming operator (Code 11) **(B)** Country code indicator: outgoing half-echo suppressor required
12	SIGI_12	**(A)** Access to delay operator (code 12); request not accepted **(B)** Country code indicator: no echo suppressor required
13	SIGI_13	**(A)** Access to test equipment (code 13); satellite link not included **(B)** Test call indicator
14	SIGI_14	**(A)** Incoming half-echo suppressor required; satellite link included **(B)** Country code indicator: outgoing half-echo suppressor inserted
15	SIGI_15	**(A)** End of pulsing (code 15); end of identification **(B)** Signal not used

Table 9. Meanings for R2MF Group II Forward Signals

Tone Number	Dialogic Define	Meaning
1	SIGII_1	National: Subscriber without priority
2	SIGII_2	National: Subscriber with priority
3	SIGII_3	National: Maintenance equipment
4	SIGII_4	National: Spare
5	SIGII_5	National: Operator
6	SIGII_6	National: Data transmission
7	SIGII_7	International: Subscriber, operator, or maintenance equipment (without forward transfer)
8	SIGII_8	International: Data transmission
9	SIGII_9	International: Subscriber with priority
10	SIGII_0	International: Operator with forward transfer facility
11	SIGII_11	Spare for national use
12	SIGII_12	Spare for national use
13	SIGII_13	Spare for national use
14	SIGII_14	Spare for national use
15	SIGII_15	Spare for national use

Table 10. Meanings for R2MF Group A Backward Signals

Tone Number	Dialogic Define	Meaning
1	SIGA_1	Send next digit (n+1)
2	SIGA_2	Send last but one digit (n-1)
3	SIGA_3	Address complete; change to Group B signals; no change back
4	SIGA_4	Congestion in the national network
5	SIGA_5	Send calling party's category; change to Group II; can change back
6	SIGA_6	Address complete; charge; set-up speech conditions
7	SIGA_7	Send last but two digit (n-2)
8	SIGA_8	Send last but three digit (n-3)
9	SIGA_9	Spare for national use
10	SIGA_0	Spare for national use
11	SIGA_11	Send country code indicator
12	SIGA_12	Send language or discriminating digit
13	SIGA_13	Send nature of circuit (satellite link only)
14	SIGA_14	Request for information on use of an echo suppressor
15	SIGA_15	Congestion in an international exchange or at its output

Table 11. Meanings for R2MF Group B Backward Signals

Tone Number	Dialogic Define	Meaning
1	SIGB_1	Spare for national use
2	SIGB_2	Send special information tone to indicate long-term unavailability
3	SIGB_3	Subscriber line busy
4	SIGB_4	Congestion encountered after change to Group B
5	SIGB_5	Unallocated number
6	SIGB_6	Subscriber line free; charge on answer
7	SIGB_7	Subscriber line free; no charge (only for calls to special destinations)
8	SIGB_8	Subscriber line out of order
9	SIGB_9	Spare for national use
10	SIGB_0	Spare for national use
11	SIGB_11	Spare for national use
12	SIGB_12	Spare for national use
13	SIGB_13	Spare for national use
14	SIGB_14	Spare for national use
15	SIGB_15	Spare for national use

6.2.4. R2MF Compelled Signaling

R2MF interregister signaling uses forward and backward compelled signaling. Simply put, with compelled signaling each signal is sent until it is responded to by a return signal, which in turn is sent until responded to by the other party. Each

signal stays on until the other party responds, thus compelling a response from the other party.

Reliability and speed requirements for signaling systems are often in conflict, the faster the signaling, the more unreliable it is likely to be. Compelled signaling provides a balance between speed and reliability because it adapts its signaling speed to the working conditions with a minimum loss of reliability.

The R2MF signal is composed of two significant events, tone-on and tone-off. Each tone event requires a response from the other party. Each response becomes an acknowledgement of the event and an event for the other party to respond to.

Compelled signaling must always begin with a Group I forward signal.

- The CO starts to send the first forward signal.
- As soon as the CPE recognizes the signal, it starts to send a backward signal that serves as an acknowledgement and at the same time has its own meaning.
- As soon as the CO recognizes the CPE acknowledging signal, it stops sending the forward signal.
- As soon as the CPE recognizes the end of the forward signal, it stops sending the backward signal.
- As soon as the CO recognizes the CPE end of the backward signal, it may start to send the next forward signal.

The CPE responds to a tone-on with a tone-on and to a tone-off with a tone-off. The CO responds to a tone-on with a tone-off and to a tone-off with a tone-on.

Refer to the following figures for more information:

- *Figure 17. R2MF Compelled Signaling Cycle*
- *Figure 18. Example of R2MF Signals for 4-digit DDI Application*

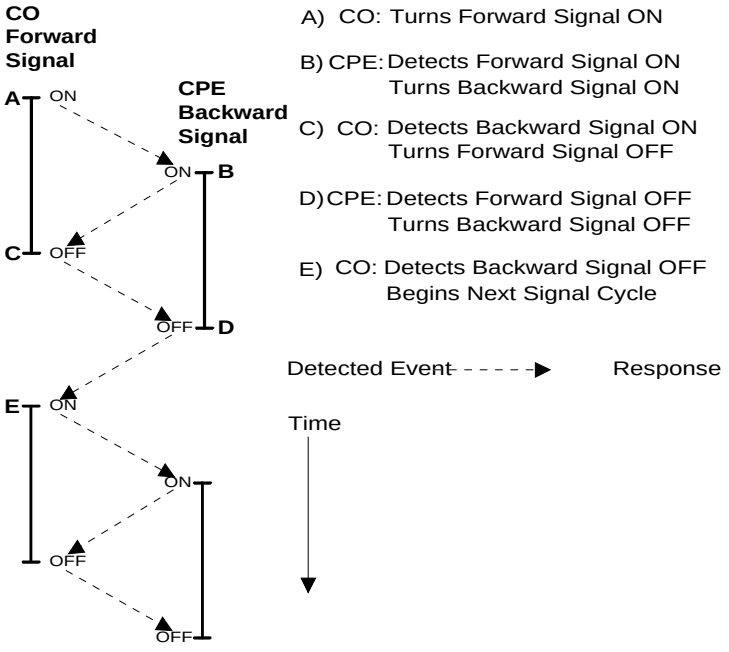


Figure 17. R2MF Compelled Signaling Cycle

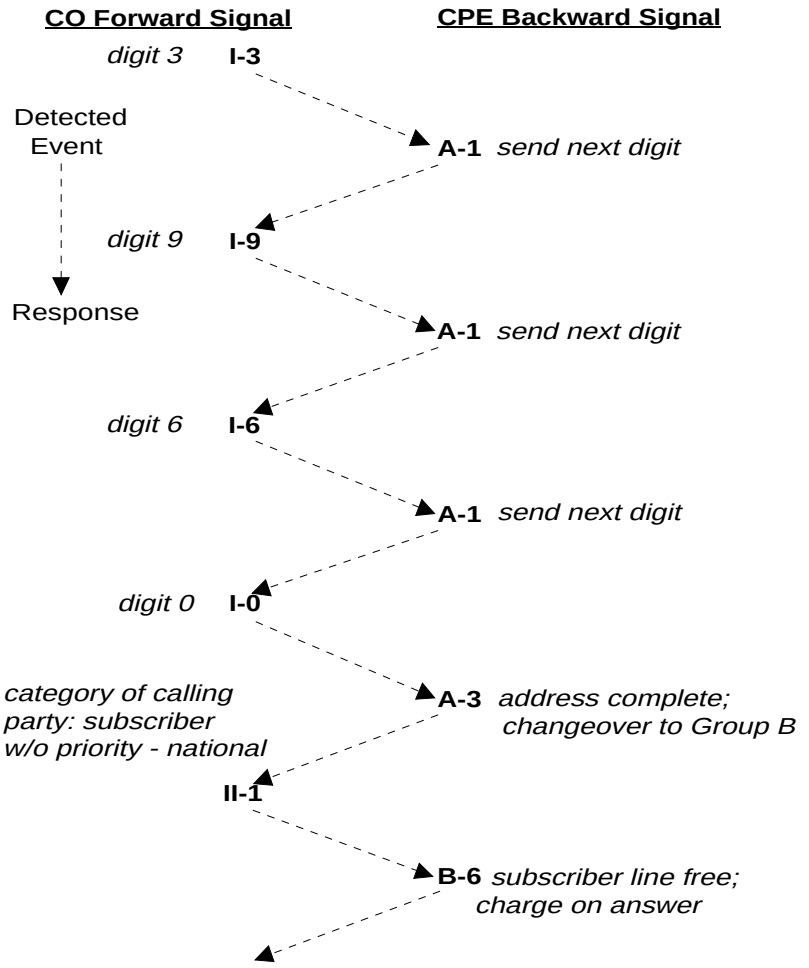


Figure 18. Example of R2MF Signals for 4-digit DDI Application

6.2.5. Related Publications

For more information on R2MF signaling, you can refer to the following publications:

Specifications of Signaling Systems R1 and R2, International Telegraph and Telephone Consultative Committee (CCITT), Blue Book Volume VI, Fascicle VI.4, ISBN 92-61-03481-0

General Recommendations on Telephone Switching and Signaling, International Telegraph and Telephone Consultative Committee (CCITT), Blue Book Volume VI, Fascicle VI.1, ISBN 92-61-03451-9

6.3. Using R2MF Signaling with Voice Boards

The Voice Software support for R2MF signaling is based upon Global Tone Detection and Global Tone Generation.

The following R2MF functions allow you to define R2MF tones for detecting the forward signals and to play the backward signals in the correct timing sequence required by the compelled signaling procedure:

r2_creatfsig( ): Create R2MF Forward Signal Tone
r2_playbsig( ): Play R2MF Backward Signal

See the *Voice Software Reference: Programmer's Guide* for a detailed description of these functions.

Four sets of defines are provided to specify the 15 Group I and 15 Group II forward signals, and the 15 Group A and 15 Group B backward signals. *Table 8*, *Table 9*, *Table 10*, and *Table 11* in *Section 6.2.3. R2MF Signal Meanings* provide a list of these defines.

6.4. R2MF Tone Detection Template Memory Requirements

To implement R2MF signaling the board must have sufficient memory blocks to create the number of user-defined tones required by your application. Your application may not need to detect all 15 forward signals, especially if you do not need to support R2MF signaling for international calls. If that is the case, your application can work with a reduced number of R2MF tones.

The D/12x, D/81A, and DIALOG/HD boards (for example, D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1, and D/320SC) normally contain sufficient memory to create the necessary R2MF tones. However, you should be aware of the maximum number of user-defined tones (including non-R2MF tones) allowed on the board. Refer to *Section 5.2.5. Maximum Number of Tone Templates*, for more information.

The D/21D, D/4xD, D/21E, D/21H, D/41E, D/41ESC, D/41H, Dialog/4 and ProLine/2V boards may also be able to create all 15 R2MF tones due to the overlap in frequencies for the R2MF signals. If creating these tones exceeds the maximum number of tones allowed, you may be able to support R2MF signaling through a reduced number of R2MF user-defined tones. Refer to *Section 5.2.5. Maximum Number of Tone Templates*.

7. Speed and Volume Control

7.1. Overview

This chapter describes how to control play-speed and play-volume.

The Voice Software contains functions and data structures to control the speed and volume of play on a channel. This allows an end user to control the speed or volume of a message by entering a DTMF tone, for example.

NOTE: Speed can be controlled on playbacks using 24 kbps or 32 kbps ADPCM only. Volume can be controlled on all playbacks regardless of the encoding algorithm.

7.2. Voice Software Speed and Volume Support

7.2.1. Speed and Volume Convenience Functions

The convenience functions set a digit that will adjust speed or volume, but do not use any data structures. These convenience functions will only function properly if you use the default settings of the Speed or Volume Modification Tables. These functions assume that the Modification Tables have not been modified. The speed or volume convenience functions are:

- **dx_addspddig( )** - adds a digit that will modify speed by a specified amount.
- **dx_addvoldig( )** - adds a digit that will modify volume by a specified amount.

7.2.2. Speed and Volume Adjustment Functions

Speed or volume can be adjusted explicitly or can be set to adjust in response to a pre-set condition, such as a specific digit. For example, speed could be set to increase a certain amount when "1" is pressed on the telephone keypad. The functions used for speed and volume adjustment are:

- **dx_setsvcond( )** - sets conditions that adjust speed or volume. Use this if you want to adjust speed or volume in response to a DTMF digit, or start of play.
- **dx_adjsv( )** - adjusts speed or volume explicitly. Use this if your adjustment condition is not a digit or start of play. For example, the application could call this function after detecting a spoken word (voice recognition) or a certain key on the keyboard.

See the *Voice Software Reference: Programmer's Guide* for detailed information about these functions.

7.2.3. Speed and Volume Modification Tables

Each channel has a Speed or Volume Modification Table for play speed or play volume adjustments. Except for the value of the settings, the table is the same for speed and volume.

Each Speed or Volume Modification table has 21 entries, 20 entries that allow for a maximum of 10 increases and decreases in speed or volume; the entry in the middle of the table is referred to as the "origin" entry that represents normal speed or volume. The normal speed or volume is how playback occurs when the Speed and Volume Control feature is not used. The normal speed or volume is the basis for all settings in the table.

Typically, the origin is set to 0. The normal speed or volume is the basis for all settings in the table. Speed and volume increases or decreases by moving up or down the tables.

Other entries in the table specify a speed or volume setting in terms of a deviation from normal. For example, if a Speed Modification Table (SMT) entry is -10, this value represents a 10% decrease from the normal speed.

Although the origin is typically set to normal speed/volume, changing the setting of the origin does not affect the other settings, because all values in the SVMT are based on a deviation from normal speed/volume.

Speed and Volume Control adjustments are specified by moving the current speed/volume pointer in the table to another SVMT table entry; this translates to

increasing or decreasing the current speed/volume to the value specified in the table entry.

A speed/volume adjustment stays in effect until the next adjustment on that channel or until a system reset.

The SVMT is like a 21-speed bicycle. You can select the gear ratio for each of the 21 speeds before you go for a ride (by changing the values in the SVMT). And you can select any gear once you are on the bike, like adjusting the current speed/volume to any setting in the SVMT.

The Speed or Volume Modification Table can be set or reset using the **dx_setsvmt( )** function which uses the DX_SVMT data structure. The current values of these tables can also be returned to the DX_SVMT structure using **dx_getsvmt( )**. The DX_SVCB data structure uses this table when setting adjustment conditions. See the *Voice Software Reference: Programmer's Guide* for information about the DX_SVMT and DX_SVCB data structures.

Adjustments to speed or volume are made by **dx_adjsv( )** and **dx_setsvcond( )** according to the Speed or Volume Modification Table settings. These functions adjust speed or volume to one of the following:

- a specified level (for example, to a specified absolute position in the speed table or volume table)
- a change in level (for example, by a specified number of steps up or down in the speed table or volume table)

For example: By default, each entry in the Volume Modification Table is equivalent to 2 decibels from the origin. Volume could be decreased by 2 decibels by specifying position 1 in the table, or by moving 1 step down.

The Speed Modification Table is shown in *Table 12*. Each entry in the table is a percentage deviation from the default play speed ("origin"). For example, the decrease[6] position reduces speed by 40%. This is four steps from the origin.

Table 12. Speed Modification Table

Table Entry	Default Value (%)	Absolute Position
decrease[0]	-128 (0x80)	-10
decrease[1]	-128 (0x80)	-9
decrease[2]	-128 (0x80)	-8
decrease[3]	-128 (0x80)	-7
decrease[4]	-128 (0x80)	-6
decrease[5]	-50	-5
decrease[6]	-40	-4
decrease[7]	-30	-3
decrease[8]	-20	-2
decrease[9]	-10	-1
origin	0	0
increase[0]	+10	1
increase[1]	+20	2
increase[2]	+30	3
increase[3]	+40	4
increase[5]	-128 (0x80)	6
increase[6]	-128 (0x80)	7
increase[7]	-128 (0x80)	8
increase[8]	-128 (0x80)	9
increase[9]	-128 (0x80)	10

NOTE: In this table, the lowest position utilized is the decrease[5] position. The remaining decrease fields are set to -128 (0x80). If these "non-adjustment" positions are selected, the default action is to play at the decrease[5] speed. These fields can be reset, as long as no values lower than -50 are used (for example, you could spread the 50% speed decrease over 10 steps rather than 5).

The Volume Modification Table is shown in *Table 13*. Each entry in the table is a deviation of *n* dB from the starting point or volume ("origin"). For example, the increase[1] position increases volume by 4 dB. This is two steps from the origin.

Table 13. Volume Modification Table

Table Entry	Default Value (dB)	Absolute Position
decrease[0]	-20	-10
decrease[1]	-18	-9
decrease[2]	-16	-8
decrease[3]	-14	-7
decrease[4]	-12	-6
decrease[5]	-10	-5
decrease[6]	-08	-4
decrease[7]	-06	-3
decrease[8]	-04	-2
decrease[9]	-02	-1
origin	0	0
increase[0]	+02	1
increase[1]	+04	2
increase[2]	+06	3
increase[3]	+08	4
increase[4]	+10	5
increase[5]	-128 (0x80)	6
increase[6]	-128 (0x80)	7
increase[7]	-128 (0x80)	8
increase[8]	-128 (0x80)	9
increase[9]	-128 (0x80)	10

NOTE: In this table, the highest position utilized is the increase[4] position. The remaining increase fields are set to -128 (0x80). If these "non-adjustment" positions are selected, the default action is to play at the

increase[4] volume. These fields can be reset, as long as no values higher than +10 are used (for example, you could spread the 10 dB volume increase over 10 steps rather than 5).

7.2.4. Play Adjustment Digits

The Voice Software processes play adjustment digits differently from normal digits:

- If a play adjustment digit is entered during playback, it causes a play adjustment only and has no other effect. This means that the digit is not added to the digit queue, it cannot be retrieved with the **dx_getdig( )** or **dx_getdigbuf( )** function, and it does not affect any termination condition.
- If the digit queue contains adjustment digits when a play begins and play adjustment is set to be level sensitive, the digits will affect the speed or volume and then be removed from the queue.

7.3. Using Speed and Volume Control

This section describes the steps for the following:

- Setting play adjustment conditions
- Adjusting play explicitly

7.3.1. Setting Adjustment Conditions

Adjustment conditions are set in the same way for speed or volume. The following steps describe how to set conditions upon which volume should be adjusted:

1. Set up the Volume Modification Table (if you do not want to use the defaults):
    - Set up the DX_SVMT structure to specify the size and number of the steps in the table.
    - Call the **dx_setsvmt( )** function, which points to the DX_SVMT structure, to modify the Volume Modification Table (**dx_setsvmt( )** can also be used to reset the table to its default values).

2. Set up the DX_SVCB structure to specify the condition, the size and type of adjustment.
3. Call **dx_setsvcond( )**, which points to an array of DX_SVCB structures. All subsequent plays will adjust volume as required whenever one of the conditions specified in the array occur.

See the *Voice Software Reference: Programmer's Guide* for more information on **dx_setsvcond( )** and **dx_setsvmt( )**. See the *Voice Software Reference: Programmer's Guide* for information on the DX_SVMT and DX_SVCB structures.

7.3.2. Explicitly Adjusting Speed and Volume

Speed and volume adjustments are made in the same way. The following is an example of the steps you should take to adjust speed, but you can use exactly the same procedure for volume:

1. Set up the Speed Modification Table (if you do not want to use the defaults):
    - Set up the DX_SVMT structure to specify the size and number of the steps in the table.
    - Call the **dx_setsvmt( )** function, which points to the DX_SVMT structure, to modify the Speed Modification Table (**dx_setsvmt( )** can also be used to reset the table to its default values).
2. When required, call **dx_adjsv( )** to adjust the Speed Modification Table, by specifying the size and type of the adjustment.

See the *Voice Software Reference: Programmer's Guide* for more information on **dx_adjsv( )** and **dx_setsvmt( )**. See the *Voice Software Reference: Programmer's Guide* for information about the DX_SVMT structure.

8. Echo Cancellation Resource

8.1. Echo Cancellation Resource Overview

This chapter describes the Echo Cancellation Resource feature, how it works, its modes of operation, as well as sample application models and sample code.

The Echo Cancellation Resource (ECR) feature is a functional component of a voice channel. In ECR mode, a voice channel can dynamically perform echo cancellation on any external SCbus time slot signal.

Prior to the implementation of the ECR feature in the Dialogic Voice Library, each voice channel device had a single transmit (TX) SCbus time slot assigned to it for data communication across the SCbus. To connect one device to another across the SCbus, an application would call **xx_listen( )** on one voice or network device to connect to a second device's transmit channel. Any signal transmitted by the second device on its transmit channel (TX channel) would be received by the first device's receive channel (RX channel). For a full duplex connection, the second device would then call **xx_listen( )** to connect its receive channel to the first device's transmit channel.

The Dialogic ECR feature provides the ability to utilize echo cancellation on signals external to the voice channel. The echo cancellation capability becomes a system-wide resource that may be applied to any SCbus PCM stream. The addition of the ECR feature allows the application to dynamically configure a voice channel as either an echo cancellation device (ECR mode) or as a standard voice processing channel (SVP mode). In ECR mode, a portion of the standard voice functionality remains available while another portion of it becomes unavailable. See section *8.2. How Echo Cancellation Resource Works* for details.

For technical information on ECR functions such as parameters, see the Voice Software Reference Volume 2 (Programmer's Guide).

NOTE: Throughout this chapter, reference is made to echo cancellation-specific terminology. See *Glossary* for definitions of ECR terminology.

8.2. How Echo Cancellation Resource Works

The Dialogic echo canceller accepts two SCbus input data streams.

One stream contains data that is identical to that which was transmitted to the echo-producing circuit (Transmit Signal in *Figure 19. Echo canceller with Relevant Input and Output Signals*).

The second stream, referred to as the echo- carrying stream, contains received data from this circuit. The received data typically contains a signal with two time-varying signals superimposed upon one another. One signal consists of a filtered version of the transmitted data (referred to as echo) and the other signal originates at the far end (referred to as 'far-end speech').

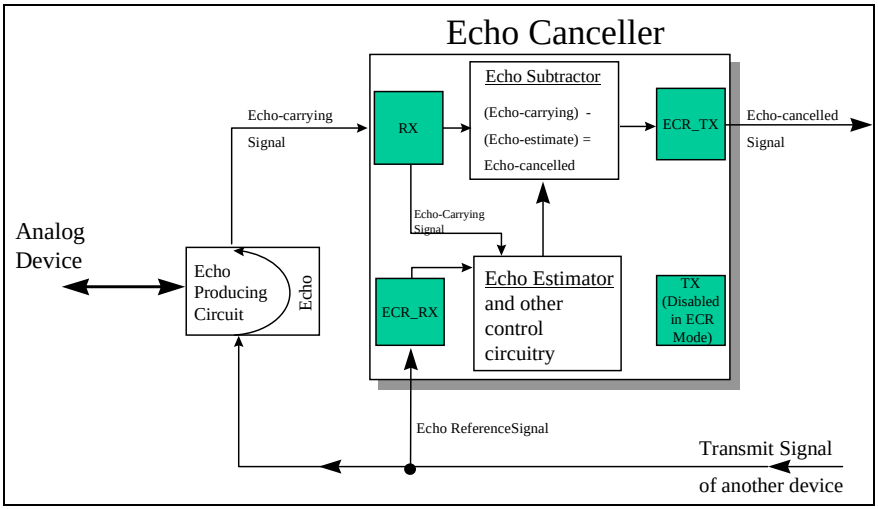


Figure 19. Echo canceller with Relevant Input and Output Signals

The purpose of the echo canceller is to reduce sufficiently the magnitude of the echo component, such that it does not interfere with further processing or analysis of the echo-cancelled data stream. The echo canceller performs this function by computing a model of the impulse response of the echo path using information in the echo-carrying signal. Then, given the impulse response model and access to the echo-reference signal, the echo canceller forms an estimate of the echo. This

estimate is then subtracted from the echo-carrying signal forming a third, echo-cancelled signal.

Figure 20. Echo canceller Operating over an SCbus illustrates the signals used in the echo canceller.

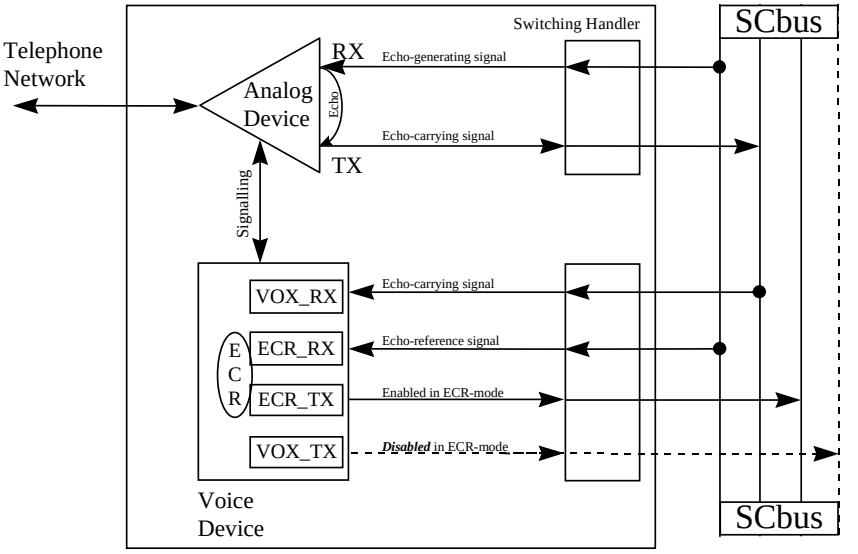


Figure 20. Echo canceller Operating over an SCbus

For echo cancellation, an extra SCbus time slot is assigned to each voice device for use by the ECR feature. In order to activate ECR mode, the application must route two receive time slots to the voice channel.

Once the ECR feature is enabled on a board, each voice channel is also permanently assigned two SCbus **transmit** time slots. These time slots are referred to as the voice-transmit time slot and the echo cancellation-transmit time slot. You can retrieve the time-slot numbers for each via the **dx_getxmitslot( )** and **dx_getxmitslotecr( )** functions, respectively. If the ECR feature is not enabled, the channels are not assigned the echo cancellation SCbus transmit time slots, and ECR mode is not possible on any voice channel of that board.

The function **dx_listen( )** routes the echo-carrying signal to the voice device. A call to **dx_listenecr( )** or **dx_listenecrex( )** routes the echo-reference signal to the voice channel and simultaneously activates ECR mode. The resulting echo canceller uses the echo-reference signal to estimate the echo component in the echo-carrying signal, and subtracts that estimate from the echo-carrying signal. This process results in an echo-cancelled signal with a greatly reduced echo component.

For another device to receive the echo-cancelled signal output by the echo canceller, it calls **dx_getxmitslotecr( )** to retrieve the echo canceller's transmit time-slot number, and calls **xx_listen( )** to connect its receive channel to the echo-cancelled signal.

To return the voice channel to Standard Voice Processing (SVP) mode, the application calls **dx_unlistenecr( )** on the voice channel to stop the echo canceller, disable ECR mode, and disconnect the echo canceller's receive channel.

For examples of echo cancellation configurations, see section *8.4. Application Models*.

8.3. Modes of Operation

When the ECR feature is enabled via the Dialogic Configuration Manager (DCM) on a supported board, there are two possible modes of operation: SVP and ECR.

Until ECR mode is activated, the board operates in the Standard Voice Processing (SVP) mode, which offers default echo cancellation. The ECR mode, which provides high performance echo cancellation, can be dynamically activated or deactivated on any voice channel of the enabled board.

8.3.1. Standard Voice Processing Mode (SVP)

All voice channels are initially in the SVP mode with the default echo cancellation for ECR feature enabled boards. The SVP mode utilizes a 48 tap (6 ms) echo canceller.

In SVP mode, all Dialogic voice functions operate as usual, with one exception. If a channel in SVP mode is playing a file and listening (via a **dx_listen( )** function),

then playback transmits data on both the standard voice-transmit time slot and the echo cancellation-transmit time slot. The standard voice-transmit time slot carries the play signal. The echo cancellation time slot carries an echo-cancelled version of the signal from the receive time slot. This echo-cancelled signal is derived from the original play signal (the echo reference) and the signal from the receive time slot specified in the **dx_listen( )** function (the echo carrying signal).

8.3.2. Echo Cancellation Resource Mode (ECR)

Any voice channel can be placed into ECR mode via the **dx_listenecr( )** or **dx_listenecrex( )** function on an ECR feature enabled board. When a voice channel is placed in ECR mode, the echo reference SCbus time slot is specified and the high performance echo canceller is activated. The ECR mode supplies 128 tap (16 ms) echo cancellation.

When an echo carrying signal is provided as an input to the ECR by an associated **dx_listen( )** function, an echo-cancelled version of that signal is produced on the echo cancellation SCbus time slot. If no echo carrying signal is defined, the contents of the echo cancellation transmit time slot are undefined and unpredictable. Other characteristics of the echo canceller can be set if the ECR mode is activated using the **dx_listenecrex( )** function.

NOTE: **dx_listen( )** may precede or follow the **dx_listenecr( )** or **dx_listenecrex( )** function. If multiple **dx_listen( )** and **dx_listenecr( )** or **dx_listenecrex( )** function calls are issued against a single channel, the echo cancellation operates on the last two issued. Successive **dx_listenecr( )** or **dx_listenecrex( )** functions can be issued without requiring any **dx_unlistenecr( )** between them.

While a channel is in ECR mode, a number of standard voice operations are not available. The unavailable operations include the following:

Table 14. Unavailable Voice Operations in ECR Mode

Unavailable Voice Operations in ECR Mode	Comment
Play	-
Record	8 KHz PCM record is the only supported record when a channel is in the ECR mode. Any such 8 KHz PCM record is a recording of the echo-cancelled signal.
Dial	-
Tone generation	-
R2MF	-
Transaction record	-

If a channel is actively performing any of the above operations, a **dx_listenecr( )** or **dx_listenecrex( )** function is not performed, and the function returns an error to the application. Conversely, if a channel is in ECR mode, a request for any of the these operations is not honored, except for the record noted. A channel may be returned to SVP mode dynamically via the **dx_unlistenecr( )** function.

8.4. Application Models

Two application models are provided in this section to illustrate building an echo-cancelled connection via the SCbus.

8.4.1. How to Set Up the ECR Bridge

This application model uses two Modular Station Interface (MSI/SC) station devices connected via the SCbus to two voice channel devices. The voice channel devices are operating in ECR mode. Two telephones are connected to the MSI/SC stations for providing input and for listening to the echo-cancelled output of each voice device.

1. Get SCbus transmit time slots of both MSI/SC devices and the ECR transmit time slots of the two voice channel devices.

```
ms_getxmitslot (MS1, &MS1_TX);
ms_getxmitslot (MS2, &MS2_TX);
dx_getxmitslotecr (CH1, &CH1_ECR_TX);
dx_getxmitslotecr (CH2, &CH2_ECR_TX);
```

2. Have both MSI/SC stations listen to the ECR transmit of the opposite voice channel.

```
ms_listen (MS1, &CH2_ECR_TX);
ms_listen (MS2, &CH1_ECR_TX);
```

3. Have both voice channel devices listen to their corresponding MSI/SC station device.

```
dx_listen (CH1, &MS1_TX);
dx_listen (CH2, &MS2_TX);
```

4. Have each voice channel connect its echo canceller's receive time slot to the opposite echo canceller's ECR transmit. These signals are used as echo-reference signals.

```
dx_listenecr (CH1, & CH2_ECR_TX);
dx_listenecr (CH2, & CH1_ECR_TX);
```

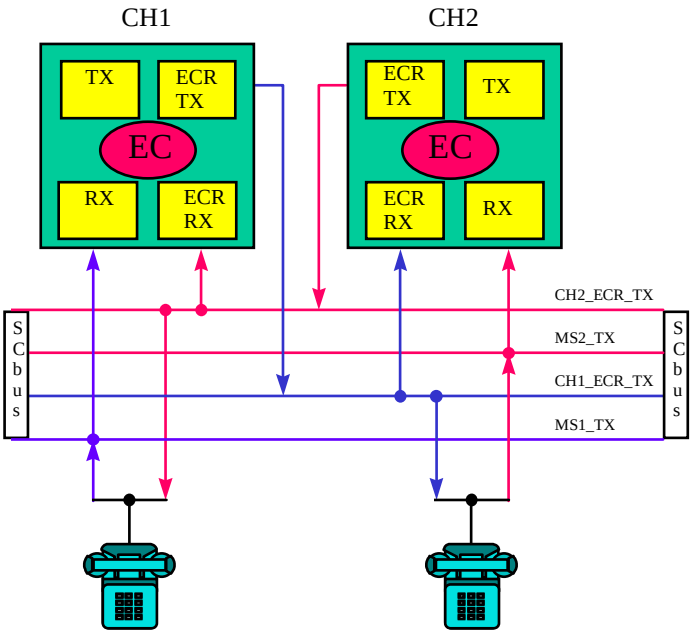


Figure 21. ECR Bridge Example Diagram

■ Example

```
#include <stdio.h>
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <msilib.h>
#include <errno.h>

main()
{
   int chdev1, chdev2;          /* Voice channel device handles */
   int msdev1, msdev2;          /* MSI/SC station device handles */
   SC_TSINFO  sc_tsinfo;        /* SCbus time slot information structure */
   long scts;                   /* Pointer to SCbus time slot */
   long ms1txts, ms2txts,       /* Transmit time slots of stations 1 & 2 */
        ch1ecrtxts, ch2ecrtxts; /* Transmit time slots of echo cancellers on voice
                                   channels 1 & 2 */

   /* Open voice board 1 channel 1 device */
   if ((chdev1 = dx_open("dxxxB1C1", 0)) == -1) {
       printf("Cannot open channel dxxxB1C1.  errno = %d", errno);
       exit(1);
```

```
}
/* Open voice board 1 channel 2 device */
if ((chdev2 = dx_open("dxxxB1C2", 0)) == -1) {
    printf("Cannot open channel dxxxB1C2.  errno = %d", errno);
    exit(1);
}
/* Open MSI/SC board 1 station 1 device */
if ((msdev1 = ms_open("msiB1C1", 0)) == -1) {
    printf("Cannot open station msiB1C1.  errno = %d", errno);
    exit(1);
}
/* Open MSI/SC board 1 station 2 device */
if ((msdev2 = ms_open("msiB1C2", 0)) == -1) {
    printf("Cannot open station msiB1C2.  errno = %d", errno);
    exit(1);
}

/* Initialize an SCbus time slot information */
sc_tsinfo.sc_numts = 1;
sc_tsinfo.sc_tsarrayp = &scts;

/* Get SCbus time slot connected to transmit of MSI/SC station 1 on board 1 */
if (ms_getxmitslot(msdev1, &sc_tsinfo) == -1) {
    printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev1), ATDV_NAMEP(msdev1));
    exit(1);
}
ms1txts = scts;

/* Get SCbus time slot connected to transmit of MSI/SC station 2 on board 1 */
if (ms_getxmitslot(msdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev2), ATDV_NAMEP(msdev2));
   exit(1);
}
ms2txts = scts;

/* Get SCbus time slot connected to transmit of voice channel 1 on board 1 */
if (dx_getxmitslotecr(chdev1, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
   exit(1);
}
   ch1ecrtxts = scts;

/* Get SCbus time slot connected to transmit of voice channel 2 on board 1 */
if (dx_getxmitslotecr(chdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}
ch2ecrtxts = scts;

/* Have MSI/SC station 1 listen to channel 2's echo-cancelled transmit */
if (ms_listen(msdev1, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev1), ATDV_NAMEP(msdev1));
   exit(1);
}

scts = ch1ecrtxts;

/* Have MSI/SC station 2 listen to channel 1's echo-cancelled transmit */
if (ms_listen(msdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev2), ATDV_NAMEP(msdev2));
   exit(1);
}
```

```
scts = ms1txts;

/* Have channel 1 listen to station 1's transmit */
if (dx_listen(chdev1, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
   exit(1);
}

scts = ms2txts;

/* Have channel 2 listen to station 2's transmit */
if (dx_listen(chdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}

scts = ch2ecrtxts;

/* Have channel 1's echo canceller listen to channel 2's echo-cancelled transmit */
if (dx_listenecr(chdev1, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
   exit(1);
}

scts = ch1ecrtxts;

/* Have channel 2's echo canceller listen to channel 1's echo-cancelled transmit */
if (dx_listenecr(chdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}
/* Bridge connected, both stations receive echo-cancelled signal */

/*
 *
 * Continue
 *
 */

/* Then perform xx_unlisten() and dx_unlistenecr(), plus all necessary xx_close()s */

if (ms_unlisten(msdev2) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev2), ATDV_NAMEP(msdev2));
   exit(1);
}
if (ms_unlisten(msdev1) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev1), ATDV_NAMEP(msdev1));
   exit(1);
}
if (dx_unlistenecr(chdev2) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}
if (dx_unlistenecr(chdev1) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
   exit(1);
}
if (dx_unlisten(chdev2) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}
```

```
   if (dx_unlisten(chdev1) == -1) {
      printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
      exit(1);
   }
   if (dx_close(chdev1) == -1) {
      printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
      exit(1);
   }
   if (dx_close(chdev2) == -1) {
      printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
      exit(1);
   }
   if (ms_close(msdev1) == -1) {
      printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
      exit(1);
   }
   if (ms_close(msdev2) == -1) {
      printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev2), ATDV_NAMEP(msdev2));
      exit(1);
   }
   return(0);
}
```

8.4.2. How to Set Up an ECR Play Over the SCbus

In this model, two MSI/SC station devices are connected via the SCbus to two voice channel devices. The second voice channel device is operating in ECR mode. Two telephones are connected to the MSI/SC stations for providing input and listening to the echo-cancelled output of the second voice device, and to the non-echo-cancelled output of the first voice device.

1. Get SCbus transmit time slots of both MSI/SC devices and the ECR transmit time slots of the two voice channel devices.

   ```
   ms_getxmitslot (MS1, &MS1_TX);
   dx_getxmitslot (CH1, &CH1_TX);
   dx_getxmitslotecr (CH2, &CH2_ECR_TX);
   ```

2. Have the MSI/SC station 1 listen to the transmit (TX) of channel 1.

   ```
   ms_listen (MS1, & CH1_TX);
   ```

3. Have MSI/SC station 2 listen to the ECR transmit of channel 2.

   ```
   ms_listen (MS2, & CH2_ECR_TX);
   ```

4. Have voice channel 2 listen to MSI/SC station 1’s transmit.

   ```
   dx_listen (CH2, & MS1_TX);
   ```

5. Have voice channel 2 connect its echo canceller's receive time slot to transmit of channel 1. This signal is used as the echo-reference signal.

```
dx_listenecr (CH2, & CH1_TX);
```

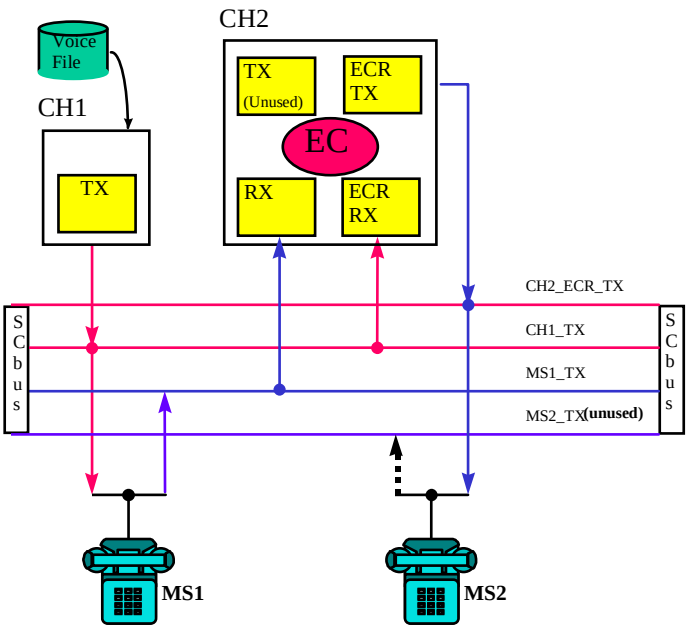


Figure 22. An ECR Play Over the SCbus

■ Example

```
#include <stdio.h>
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <msilib.h>
#include <errno.h>

main()
{
   int chdev1, chdev2;          /* Voice channel device handles */
   int msdev1, msdev2;          /* MSI/SC station device handles */
   SC_TSINFO  sc_tsinfo;        /* SCbus time slot information structure */
   long scts;                   /* Pointer to SCbus time slot */
   long ms1txts,                /* Transmit time slots of stations 1 & 2 */
```

```
     ch1txts, ch2ecrtxts;  /* Transmit time slots of echo cancellers on
                              voice channels 1 & 2 */

/* Open voice board 1 channel 1 device */
if ((chdev1 = dx_open("dxxxB1C1", 0)) == -1) {
    printf("Cannot open channel dxxxB1C1.  errno = %d", errno);
    exit(1);
}
/* Open voice board 1 channel 2 device */
if ((chdev2 = dx_open("dxxxB1C2", 0)) == -1) {
    printf("Cannot open channel dxxxB1C2.  errno = %d", errno);
    exit(1);
}
/* Open MSI/SC board 1 station 1 device */
if ((msdev1 = ms_open("msiB1C1", 0)) == -1) {
    printf("Cannot open station msiB1C1.  errno = %d", errno);
    exit(1);
}
/* Open MSI/SC board 1 station 2 device */
if ((msdev2 = ms_open("msiB1C2", 0)) == -1) {
    printf("Cannot open station msiB1C2.  errno = %d", errno);
    exit(1);
}

/* Initialize an SCbus time slot information */
sc_tsinfo.sc_numts = 1;
sc_tsinfo.sc_tsarrayp = &scts;

/* Get SCbus time slot connected to transmit of voice channel 1 on board 1 */
if (ms_getxmitslot(msdev1, &sc_tsinfo) == -1) {
printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev1), ATDV_NAMEP(msdev1));
    exit(1);
}
ms1txts = scts;

/* Get SCbus time slot connected to transmit of voice channel 1 on board 1*/
if (dx_getxmitslot(chdev1, &sc_tsinfo) == -1) {
printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
   exit(1);
}
ch1txts = scts;

/* Get SCbus time slot connected to transmit of voice channel 1 on board 1 */
if (dx_getxmitslotecr(chdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}
ch2ecrtxts = scts;

/* Have station 1 listen to file played by voice channel 1 */
scts = ch1txts;
if (ms_listen(msdev1, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev1), ATDV_NAMEP(msdev1));
   exit(1);
}

/* Have station 2 listen to echo-cancelled output of voice channel 2 */
scts = ch2ecrtxts;
if (ms_listen(msdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev2), ATDV_NAMEP(msdev2));
   exit(1);
}
```

```
/* Have voice channel 2 listen to echo-carrying signal from station 1 */
scts = ms1txts;
if (dx_listen(chdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
   exit(1);
}

/* And activate the ECR feature on voice channel 2, with the echo-reference signal
   coming from voice channel 1 */
scts = ch1txts;
if (dx_listenecr(chdev2, &sc_tsinfo) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}

/* Setup completed, any signal transmitted from channel device 1,
 *  will a) be received by station 1,
 *       b) contribute echo to the transmit of station 1,
 *       c) will be heard AFTER echo cancellation (on channel 2) by
 *          station 2.*/

/*
.
. Continue
.
*/

/* Then perform xx_unlisten() and dx_unlistenecr(), plus all necessary xx_close()s */

if (ms_unlisten(msdev2) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev2), ATDV_NAMEP(msdev2));
   exit(1);
}
if (ms_unlisten(msdev1) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev1), ATDV_NAMEP(msdev1));
   exit(1);
}
if (dx_unlistenecr(chdev2) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), TDV_NAMEP(chdev2));
   exit(1);
}
if (dx_unlisten(chdev2) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}
if (dx_close(chdev1) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev1), ATDV_NAMEP(chdev1));
   exit(1);
}
if (dx_close(chdev2) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(chdev2), ATDV_NAMEP(chdev2));
   exit(1);
}
if (ms_close(msdev1) == -1) {
   printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev1), ATDV_NAMEP(msdev1));
   exit(1);
}
```

```
   if (ms_close(msdev2) == -1) {
      printf("Error message = %s, on %s", ATDV_ERRMSGP(msdev2), ATDV_NAMEP(msdev2));
      exit(1);
   }
   return(0);
}
```

9. Analog Display Services Interface (ADSI)

9.1. Overview

The Analog Display Services Interface (ADSI) is a Bellcore standard that defines a protocol used to transmit data to a display-based, ADSI-compliant telephone. ADSI enables data to be sent across an analog telephone line, providing asynchronous data communications with 8 data bits, 1 start and 1 stop bit, and no parity.

For many years, Dialogic provided one-way ADSI support through the **dx_play( )** and **dx_playf( )** functions. This ADSI support enabled developers to use Dialogic boards to make ADSI servers that work with ADSI phones and to support ADSI features such as visual Voice Mail. This is referred to as the "older" implementation of one-way ADSI (for details on this older method, see the **dx_play( )** function in the *Voice Software Reference: Programmer's Guide*).

The newer implementation of ADSI offers several enhancements and is supported through the **dx_RxIottData( )**, **dx_TxIottData( )**, and **dx_TxRxIottData( )** functions. This implementation is referred to simply as "**ADSI Support**" or "**Two-Way ADSI**." This newer Dialogic ADSI support provides for both two-way and one-way ADSI transmission and is the recommended method for implementing either one-way or two-way ADSI in an application program. The older one-way ADSI support can be used but is not the recommended method. See *Section 9.5.3. Modifying Older One-Way ADSI Applications* for information on converting from the older to the newer method for using ADSI.

Dialogic ADSI support includes the following features:

9.2. One-Way ADSI

One-way ADSI support enables Dialogic boards to be used as ADSI servers and to support ADSI features such as visual Voice Mail. One-way ADSI allows for the one-way transmission of data from a server to a Customer Premises Equipment

(CPE) device, such as a display-based telephone. The phone (CPE) sends dual tone multi-frequency (DTMF) messages to the server, indicating whether the data was received successfully.

NOTE: For a more detailed description of the one-way ADSI data transfer process, see *Section 9.5. Developing ADSI Applications.*

ADSI data can be transferred only to display-based telephones that are ADSI compliant. (Check with your telephone manufacturer to find out if your telephone is a true ADSI-compliant device.) An ADSI alert tone, referred to as a CAS (CPE Alerting Signal), is sent by the Dialogic server to query a CPE device, such as an ADSI display phone. The device responds appropriately and, if the device is ADSI-compliant, the ADSI data transfer is initiated.

NOTE: ADSI-compliant phones are also referred to as "Type 3 CPE Devices" by Bellcore and by the Electronic Industry Association/Telecommunications Industry Association (EIA/TIA).

9.3. Two-Way ADSI

Two-way ADSI includes several enhancements to one-way ADSI, including the two-way Frequency Shift Keying (FSK). Two-way ADSI adds the following enhancements to the one-way ADSI capabilities:

- Transmit to On-Hook CPE (ADSI phones): Allows messages to be sent to an ADSI phone when the phone is either On-Hook or Off-Hook.
- Two-Way Frequency Shift Keying (FSK): Allows users to send and receive character or binary data at 1200 bits/second between the server and compatible devices, such as certain ADSI phones with keyboards. The two-way FSK feature supports applications such as off-line Email editing and sending FSK Caller ID data to a customer premise equipment (CPE) device through an MSI/SC board.

9.3.1. Transmit to On-Hook CPE

The Transmit to On-Hook CPE feature supports the transmission of FSK data burst messages to CPE devices that are kept in the On-Hook state by either the CO or the PBX/KTS. This allows an ADSI/Caller ID phone to receive and potentially display messages while it is in the On-Hook state. For example, ADSI phones can

be configured, accessed, and downloaded with features, outside of regular business hours while the phone is On-Hook, without ringing and without subscriber intervention.

NOTE: The Transmit to On-Hook CPE feature works only if the CO supports this feature, or if an ADSI device is connected directly to a Dialogic MSI/SC board.

9.3.2. Two-Way FSK

FSK (Frequency Shift Keying) is the modulation technique used to transfer data over voice lines. The basic ADSI capability supports only FSK Transmit (one-way FSK), in which an FSK message is sent from the Dialogic server to an ADSI display phone, with the phone in the Off-Hook state. The phone (CPE) sends dual tone multi-frequency (DTMF) messages to the server. As DTMF messages are sent to the server, the effective data rate is very slow, approximately 6 characters per second maximum. This speed is satisfactory for ACK/NAK signaling but it is not usable for any bulk data transport in the inbound direction from the CPE.

NOTE: FSK data reception uses a DSP-based Bell 202/V.23 low speed (1200 baud) modem receiver. A 1200 baud modem does not need to train for data transmission, and therefore is faster than a high-speed modem for short data bursts.

Two-way FSK for ADSI supports the transmission and the reception of FSK data between the server and the CPE. The server initiates the reception of data from the CPE by sending a CAS to tell the CPE to switch to data mode, followed by a message that tells the CPE to switch to peripheral mode. Once it is in peripheral mode, the CPE can send FSK messages to the server using the ADSI Data Message Format (ADMF), instead of the slower DTMF-based scheme.

NOTES:

1. See *Section 9.5. Developing ADSI Applications* for a more detailed description of how use the Dialogic library functions to develop two-way ADSI data transfer applications.
2. For more information about two-way FSK transmission, see the Bellcore Special Report SR-3462, *A Two-Way Frequency Shift Keying Communication for the ADSI*. Bellcore documents can be obtained from Bellcore by calling 1-800-521-CORE.

In addition to features provided by basic ADSI Two-way FSK for ADSI can be used in the following applications:

- sending and receiving Email between display-based ADSI phones and the server
- sending FSK Caller ID data through the MSI to a CPE device

NOTE: To send Caller ID information through an MSI/SC board, the ADSI phones connected to the board must be able to detect and accept the transmission of Caller ID information prior to the first ring.

9.3.3. Two-Way FSK for ADSI Functions

There are three Data Transfer functions included in *the Voice Software Reference: Programmer's Guide* that are used to support the two-way FSK feature . These functions are as follows:

- **dx_TxIottData( )** - sends data
- **dx_RxIottData( )** - receives data
- **dx_TxRxIottData( )** - sends and then receives data

Section 9.5. Developing ADSI Applications provides instructions for using these functions to develop one-way and two-way ADSI data transfer applications.

NOTE: Users with older ADSI applications: The **dx_play( )** function can still be used for the transmission of ADSI data. However, future enhancements to the ADSI feature and functionality will not be made to the **dx_play( )** function. Therefore, we recommend that you convert your older ADSI applications and replace **dx_play( )** with the new **dx_TxIottData( )** function. See *Section 9.5.3. Modifying Older One-Way ADSI Applications* for more information.

In addition, the following functions are used to support the two-way ADSI:

- **dx_setparm( )** - sets board or channel parameters
- **dx_getparm( )** - retrieves current parameter settings

9.4. The ADSI Protocol

ADSI is a superset of the Caller ID and Call Waiting functions. ADSI is built on the same protocol as Caller ID and shares the same ADSI Data Message Format (ADMF). The ADSI protocol requires a Bell 202/V.23 1200 bps FSK-based modem for data transmission.

The ADSI protocol consists of three defined layers, as follows:

- Message Assembly Layer - assembles the body of the ADMF message
- Data Link Layer - generates the checksum, which is used to for error detection, and sends it to the driver
- Physical Layer - transports the composite message via the modem to the CPE on a transparent (bit-for-bit) basis

NOTES:

1. Dialogic provides only the Physical Layer and a portion of the Data Link Layer of the ADSI protocol. The user is responsible for creating the ADSI messages and the corresponding checksums.
2. The ADSI data must conform to interface requirements described in Bellcore Technical Reference GR-30-CORE, *Voiceband Data Transmission Interface Generic Requirements*. For information about message requirements (how the data should be displayed on the CPE), see Bellcore Technical Reference TR-NWT-001273, *Generic Requirements for and SPCS to Customer Premises Equipment Data Interface for Analog Display Services*. These technical references can be obtained from Bellcore by calling 1-800-521-CORE.

9.5. Developing ADSI Applications

This section provides the following information:

- A technical overview of how the one-way and two-way ADSI data transfer processes work
- Guidelines for using the Dialogic library functions to develop applications for both one-way and two-way ADSI data transfer
- Instructions for modifying older one-way ADSI applications to replace the **dx_play( )** function with the **dx_TxIottData( )** function

NOTE: For more information about ADSI and two-way FSK, refer to the following Bellcore documents:

- For information about interface requirements, see Bellcore Technical Reference GR-30-CORE, *Voiceband Data Transmission Interface Generic Requirements.*
- For information about message requirements (how the data should be displayed on the CPE), see Bellcore Technical Reference TR-NWT-001273, *Generic Requirements for and SPCS to Customer Premises Equipment Data Interface for Analog Display Services.*
- For information about two-way FSK transmission, see the Bellcore Special Report SR-3462, *A Two-Way Frequency Shift Keying Communication for the ADSI.*

Bellcore documents can be obtained from Bellcore by calling 1-800-521-CORE.

9.5.1. One-Way ADSI Data Transfer

In one-way ADSI data transfer, the ADSI server sends ADSI messages to a CPE device, such as an ADSI-compliant telephone. The transactions that occur between the server and the CPE during one-way ADSI data transfer are as follows:

1. The server initiates the data transfer by sending a CPE Alerting Signal (CAS) to the CPE.
2. When the CPE receives the CAS, the device generates an ACK (DTMF 'A' signal) to the server. At this point the CPE device has switched from voice mode to data mode. (If the CPE device remains in data mode, subsequent transmissions do not require the CAS.)

   NOTE: Only ADSI-compliant CPE devices will respond to the CAS sent by the server. Check with your manufacturer to verify that your CPE device is a true ADSI-compliant device. ADSI-compliant devices are also referred to as "Type 3 CPE Devices" by Bellcore and the EIA/TIA.

3. Upon receipt of the ACK signal, the server initiates the FSK transmission sequence. Each FSK transmission sequence can contain anywhere from 1 to 5 messages.

4. The CPE receives the FSK data and uses the checksum included within the sequence to determine the number of messages successfully received.
5. The CPE device then responds to the server with an acknowledgment digit (DTMF 'D') followed by a DTMF of '0' through '5,' which indicates the number of messages successfully received.
6. The server interprets the DTMF as follows:
   - ACK = 'D' followed by a DTMF in the range of 1 – 5
   - NAK = 'D' followed by a DTMF '0'

Using dx_TxIottData( ) to Implement One-Way ADSI

The **dx_TxIottData( )** function is used to send the CAS to the CPE and implement one-way ADSI data transfer. To transfer ADSI FSK data, the function parameters and structures must be configured as follows:

- set the **wType** parameter DT_ADSI
- configure the DX_IOTT structure with the appropriate ADSI FSK data file(s). The application is responsible for constructing the messages and checksums for each transmission
- set the termination conditions with the DV_TPT structure
- set **dwTxDataMode** within the ADSI_XFERSTRUC referenced by **lpParams** to ADSI_ALERT to generate the CAS

The following scenario illustrates the function calls that are required to generate an initial CAS to the CPE and begin one-way ADSI data transfer. The chart describes what is happening on the voice channel between the server and the CPE.

Step	Server	Voice Channel	CPE
1	dx_clrdigbuf( )	Digit buffer cleared	
2	dx_TxIottData( ) with dwTxDataMode = ADSI_ALERT		
3		CAS---------->	<---ACK (DTMF 'A')
4			NAK (DTMF '0') or
5		Data file(s)-------->	<---ACK (DTMF 'Dx' where x = 1 to 5)
6	dx_getdig( ) - retrieve ACK digits	Digits received	

1. Prior to executing **dx_TxIottData( )**, the digit buffer for the desired voice channel is cleared using the **dx_clrdigbuf( )** function.
2. The **dx_TxIottData( )** function is issued. To generate an initial CAS to the CPE device, **dwTxDataMode** within ADSI_XFERSTRUC must be set to ADSI_ALERT.
3. The CAS is received by the CPE and the CPE sends an acknowledgment digit (DTMF 'A') to the voice device.

   NOTE: If the DTMF acknowledgment digit is not received from the CPE device within 500 ms following the end of the CAS, the function will return a 0 but the termination mask returned by **ATDX_TERMMSK( )** will be TM_MAXTIME to indicate an ADSI protocol error. (The function will return a -1 if a failure is due to a general transmission error.)

4. Upon receipt of the DTMF 'A' ACK, the voice device automatically transmits the data file referenced in the DX_IOTT structure.
5. After receiving the data file(s), the CPE responds with a DTMF ACK or NAK, indicating the number of messages successfully received. (The application is responsible for determining whether the message count acknowledgment matches the number of messages that were transmitted and for re-transmitting any messages.)

   NOTE: Upon successful completion, the function terminates with a TM_EOD (end of data) termination mask returned by **ATDX_TERMMSK( )**.

6. After completion of **dx_TxIottData( )**, the **dx_getdig( )** function retrieves the DTMF ACK sequence from the CPE device. Set the DV_TPT **tp_termno** parameter to DX_DIGTYPE to receive the DTMF string "adx," where "x" is the message count acknowledgment digit (1-5).

After the CAS is sent to the CPE, as described in the preceding scenario, the CPE is in data mode. Provided that the ADSI messages sent to the CPE instruct the CPE to remain in data mode, subsequent ADSI transmissions to the CPE do not require the CAS. To send ADSI data without the CAS, set the **dwTxDataMode** within the ADSI_XFERSTRUC referenced by **lpParams** to ADSI_NOALERT. All other settings are the same as above.

The following scenario illustrates the function calls that are required to transfer ADSI data when the CPE is already in data mode (without sending a CAS). The chart describes what is happening on the voice channel between the server and the CPE.

Step	Server	Voice Channel	CPE
1	dx_clrdigbuf( )	Digit buffer cleared	
2	dx_TxIottData( ) with dwTxDataMode = ADSI_NOALERT	Data file(s)----->	
3	dx_getdig( ) - retrieve ACK digits	Digits received	NAK (DTMF '0') or <--ACK (DTMF 'Dx' where x = 1 to 5)
4			

1. Prior to executing **dx_TxIottData( )**, the **dx_clrdigbuf( )** function is issued to ensure the voice channel digit buffer is empty.
2. The **dx_TxIottData( )** function is issued with **dwTxDataMode** within the ADSI_XFERSTRUC set to ADSI_NOALERT. This initiates the immediate transfer of the data file(s) referenced in the DX_IOTT structure to the CPE device.
3. After receiving the data file(s), the CPE responds with a DTMF ACK or NAK, indicating the number of messages successfully received. (The application is responsible for determining whether the message count acknowledgment matches the number of messages that were transmitted and for re-transmitting any messages.)
4. After completion of **dx_TxIottData( )**, the **dx_getdig( )** function retrieves the DTMF ACK sequence from the CPE device. Set the DV_TPT **tp_termno** parameter to DX_DIGTYPE to receive the DTMF string "adx," where "x" is the message count acknowledgment digit (1-5).

9.5.2. Two-Way ADSI Data Transfer

In two-way ADSI data transfer, both the ADSI server and CPE device can transmit and receive ADSI data messages. The CAS is used to initiate the transfer of ADSI FSK data and to return the CPE to voice mode after the data exchange is completed.

The transactions that occur between the server and the CPE in two-way ADSI data transfer are as follows:

1. The server initiates the data transfer by sending a CPE Alerting Signal (CAS) to the CPE equipment.
2. Upon receipt of the CAS, the CPE device generates an ACK (DTMF 'A' signal) to the server. At this point the CPE device has switched from voice mode to data mode. (Once the CPE device is in data mode, subsequent FSK data transmissions do not require the CAS.)

   NOTE: Only ADSI-compliant CPE devices will respond to the CAS sent by the server. Check with your manufacturer to verify that your CPE device is a true ADSI-compliant device. ADSI-compliant devices are also referred to as "Type 3 CPE Devices" by Bellcore.
3. When the ACK signal is received, the server initiates the FSK transmission sequence. Each FSK transmission sequence can contain anywhere from 1 to 5 messages. A "Switch to Peripheral Mode" message (using 0x0A as a 'requested peripheral' code) must be included within the FSK transmission sequence.
4. The CPE receives the FSK data and uses the checksum included within the sequence to determine the number of messages successfully received.
5. The CPE device then responds to the server with a DTMF 'D' followed by a DTMF '0' through '5' to indicate the number of messages successfully received. In addition, the CPE device acknowledges the "Switch to Peripheral Mode" message by responding with either
   - DTMF 'B,' indicating that the requested peripheral is available and on line
   - DTMF 'A,' indicating that the requested peripheral is not available
6. The server interprets the DTMF signals as follows:
   - 'D' followed by a DTMF in the range of 1 – 5 = ACK
   - 'D' followed by a DTMF '0' = NAK
   - DTMF 'B' = requested peripheral available (ready to receive and transmit ADSI data)
   - DTMF 'A' = requested peripheral unavailable (unable to transmit or receive ADSI data)

Once the CPE device has acknowledged the "Switch to Peripheral Mode" message, the CPE may transmit data to the server at any time. The server must be prepared to receive data at any time until the CPE peripheral is switched back to voice mode. To return the CPE peripheral to voice mode, the server sends a CAS to the CPE. Upon receipt of the CAS, the CPE responds with a DTMF 'A' signal. Receipt of DTMF 'A' at the server completes the return to voice mode transition.

Using dx_TxIottData( ) to Implement Two-Way ADSI

The **dx_TxIottData( )** function is used to implement two-way ADSI data transfer. The **dx_TxIottData( )** function transmits the CAS and the subsequent "Switch to Peripheral Mode Message" to the CPE. To transfer ADSI FSK data, set the parameters and configure the structures as described below:

- set the **wType** parameter DT_ADSI
- configure the DX_IOTT structure with the appropriate ADSI FSK data file(s), including the "Switch to Peripheral Mode" message. The application is responsible for constructing the messages and checksums for each transmission
- set the termination conditions with the DV_TPT structure
- set **dwTxDataMode** within the ADSI_XFERSTRUC referenced by **lpParams** to ADSI_ALERT to generate the CAS

The following scenario illustrates the function calls that are required to generate an initial CAS to the CPE and begin two-way ADSI data transfer. The chart describes what is happening on the voice channel between the server and the CPE.

Step	Server	Voice Channel	CPE
1	dx_clrdigbuf( )	Digit buffer cleared	
2	dx_TxIottData( ) with dwTxDataMode = ADSI_ALERT	CAS---------->	
3			<---ACK (DTMF 'A')
4		Data file(s) including 'Switch to Peripheral Mode' -------->	
			NAK (DTMF '0') or
5		Digits received	<---ACK (DTMF 'Dx' where x = 1 to 5)
			Peripheral available (DTMF 'B') or
6		Digits received	<---Peripheral unavailable (DTMF 'A')
7	dx_getdig( ) - retrieve ACK digits		

1. Prior to executing **dx_TxIottData( )** , the digit buffer for the desired voice channel is cleared using the **dx_clrdigbuf( )** function.
2. The **dx_TxIottData( )** function is issued. To generate an initial CAS to the CPE device, **dwTxDataMode** within ADSI_XFERSTRUC must be set to ADSI_ALERT.
3. The CAS is received by the CPE and the CPE sends an acknowledgment digit (DTMF 'A') to the voice device.

   NOTE: If the DTMF acknowledgment digit is not received from the CPE device within 500 ms following the end of the CAS, the function will return a 0 but the termination mask returned by **ATDX_TERMMSK( )** will be TM_MAXTIME to indicate an ADSI protocol error. (The function will return a -1 if a failure is due to a general transmission error.)

4. Upon receipt of the DTMF 'A' ACK, the voice device automatically transmits the data file referenced in the DX_IOTT structure, which must include the "Switch to Peripheral Mode" message.
5. After receiving the data file(s), the CPE responds with a DTMF ACK or NAK, indicating the number of messages successfully received. (The application is responsible for determining whether the message count acknowledgment matches the number of messages that were transmitted and for re-transmitting any messages.)

NOTE: Upon successful completion, the function terminates with a TM_EOD (end of data) termination mask returned by **ATDX_TERMMSK( )**.

6. The CPE responds to the "Switch to Peripheral Mode" message with either DTMF 'B' if the peripheral is available or DTMF 'A' if the peripheral is unavailable.

7. After completion of **dx_TxIottData( )**, the **dx_getdig( )** function retrieves the DTMF ACK sequence from the CPE device. Set the DV_TPT **tp_termno** parameter to DX_DIGTYPE to receive the DTMF string "adxb," where "x" is the message count acknowledgment digit (1-5). When the DTMF string is received, additional messages can be sent and received between the server and the CPE peripheral.

Using dx_TxRxIottData( ) for Two-Way ADSI Transmissions

After the two-way ADSI transmission is implemented using the **dx_TxIottData( )** function, additional ADSI FSK messages are typically sent to the CPE peripheral to configure the display and soft keys. Since at this point the CPE peripheral has been configured to send data to the server, the **dx_TxRxIottData( )** function should be used to send the data to the CPE and then quickly turnaround and be ready to receive data from the CPE.

To transfer ADSI FSK data using **dx_TxRxIottData( )**, set the function parameters and configure the structures as described below:

- set **wType** to DT_ADSI
- configure DX_IOTT structures referenced by **lpTxIott** and **lpRxIott** with the appropriate ADSI FSK data files. The application is responsible for constructing the messages and checksums for each transmission
- set the termination conditions for the transmit and receive portions of the function with the DV_TPT structures referenced by **lpTxTerminations** and **lpRxTerminations**, respectively
- set **dwTxDataMode** and **dwRxDataMode** within the ADSI_XFERSTRUC referenced by **lpParams** to ADSI_NOALERT to transmit and receive FSK ADSI data without generation of a CAS

The following scenario illustrates the function calls that are required to send and receive FSK ADSI data between the server and the CPE. The chart describes what is happening on the voice channel between the server and the CPE.

Step	Server	Voice Channel	CPE
1	dx_clrdigbuf( )	Digit buffer cleared	
2	dx_TxRxIottData( ) with dwTxDataMode and dwRxDataMode = ADSI_NOALERT	Data file(s) ------>	
3		Digits received	NAK (DTMF '0') or <---ACK (DTMF 'Dx' where x = 1 to 5)
4		Wait for data	
5		Data received	<---Send FSK ADSI data to server
6	dx_getdig( ) - retrieve ACK digits		
7	Thread to process received data		
8	dx_Rx_IottData( ) with dwRxDataMode = ADSI_NOALERT for next data burst from CPE	Wait for data	

1. Prior to executing **dx_TxIottData( )**, the digit buffer for the desired voice channel is cleared using the **dx_clrdigbuf( )** function.
2. The **dx_TxRxIottData( )** function is issued with **dwTxDataMode** and **dwRxDataMode** within ADSI_XFERSTRUC set to ADSI_NOALERT. This initiates the transmission of the data file referenced in the DX_IOTT structure to the CPE. The server voice channel is placed automatically in FSK ADSI data receive mode to receive data from the CPE.
3. After receiving the data file(s), the CPE responds with a DTMF ACK or NAK, indicating the number of messages successfully received. (The application is responsible for determining whether the message count acknowledgment matches the number of messages that were transmitted and for re-transmitting any messages.)
4. The server voice channel is ready and waiting for data from the CPE.
5. The CPE sends FSK ADSI data to the server. When an ADSI FSK message is successfully received or when the termination conditions set in **lpRxTerminations** are met, the **dx_TxRxIottData( )** function completes.

6. After completion of **dx_TxRxIottData( )**, the **dx_getdig( )** function retrieves the DTMF ACK sequence for the transmit portion of the function. When the DTMF string is received, additional messages can be sent and received between the server and the CPE peripheral.
7. In another thread of execution at the server, the received message(s) are processed by the application to determine the number of messages received and the integrity of the information.
8. The **dx_RxIottData( )** function is issued to receive messages from the CPE. This function should be issued as soon as possible because the CPE peripheral can send data to the server after a minimum of 100 msec following the completion of its transmission.

   If data needs to be transmitted to the CPE when the server is waiting to receive data, issue **dx_stopch( )** to terminate the current **dx_RxIottData( )** function. Upon confirmation of termination of **dx_RxIottData( )**, issue **dx_clrdigbuf( )** to clear the voice device channel buffer, and then issue **dx_TxIottData( )** to send the data to the CPE.

9.5.3. Modifying Older One-Way ADSI Applications

Prior to the release of the two-way ADSI, including two-way FSK,, applications used the **dx_play( )** function to implement one-way ADSI applications. With two-wayADSI, transmit and receive data functions are introduced for data transfer. To take advantage of on-hook ADSI transfer in a one-way ADSI application, and/or to introduce two-way FSK concepts into applications, older applications need to be modified.

Applications developed prior to the release of the two-wayADSI use the following sequence of commands to generate a CAS tone followed by transmission of an ADSI file:

```
/* Setup DX_IOTT to play from disk */

/* Setup DV_TPT for termination conditions */

/* Initiate ADSI play when DTMF 'A' is received from CPE */
parmval = DM_A;
if (dx_setparm(Voice_Device, DXCH_DTINITSET, (void *)&parmval) == -1) {
   /* Process error */
}

/* Clear digit buffer for impending ADSI protocol DTMFs */
if (dx_clrdigbuf(Voice_Device) == -1) {
```

```
    /* Process error */
}

  /* Send CAS followed by ADSI data when DTMF 'A' is received */
if (dx_play(Voice_Device, &Iott_struct, &Tpt_struct, EV_SYNC | PM_ADSIALERT) == -1) {
    /* Process error */
}
```

In older applications, the use of **dx_play( )** for ADSI transmission can be replaced with the specialized **dx_TxIottData( )** data transfer function. The same DV_TPT and DX_IOTT are used by **dx_TxIottData( )** as for **dx_play( )**, however, the following additional parameters need to be configured:

- **wType** - specifies the data type transferred. To transfer ADSI FSK data, **wType** is set to DT_ADSI.
- **lpParams** - specifies the data type specific information. To transmit CAS followed by the ADSI FSK message, **dwTxDataMode** within the ADSI_XFERSTRUC data structure pointed to by **lpParams** is set to ADSI_ALERT.

Using these parameters, the CAS will be transmitted and, upon receipt of DTMF 'A,' the ADSI FSK data will be sent to the CPE device.

The following sample code illustrates the use of the **dx_TxIottData( )** function to generate a CAS tone and transmit an ADSI file:

```
/* Setup DX_IOTT to play from disk */

/* Setup DV_TPT for termination conditions */

/* Setup ADSI_XFERSTRUC to send CAS followed by ADSI FSK upon receipt of DTMF 'A' */
adsimode.cbSize = sizeof(adsimode);
adsimode.dwTxDataMode = ADSI_ALERT;

/* Clear digit buffer for impending ADSI protocol DTMFs */
if (dx_clrdigbuf(Voice_Device) == -1) {
    /* Process error */
}

/* Send CAS followed by ADSI data when DTMF 'A' is received */
if (dx_TxIottData(Voice_Device, &IOTT, &TPT, DT_ADSI, &adsimode, EV_SYNC) == -1) {
    /* Process error */
}
```

10. Voice Features Demonstration Programs

10.1. Overview

This chapter provides instructions for using the demonstration programs. There are two demonstration programs supplied with this System Release Software for Windows:

- Multithreaded Text Based Application Program
- Multithreaded GUI Based Application Program

10.2. Multithreaded Text Based Application Program

The Dialogic Multithreaded Text Based Application Program performs asynchronous plays and records on up to forty-eight voice channels, using one thread per channel.

To run the program, change to the directory where the demo is located (<*install drive*>:\<*install directory*>*dialogic**samples**voice*). Then type **testmt -b***x* **-t***y* to perform plays and records on *y* channels, where *x* is the first board the program tests (default is 1) and *y* is the number of threads to create in the system (default is 4).

NOTES: 1. **testmt -?** will display demo options.

2. To exit the demo program prior to completion of the demonstration, press CTRL C.

10.3. Multithreaded GUI Based Voice Features Application Program

The Multithreaded GUI Based Application Program is a multichannel Dialogic voice GUI based demonstration program for Windows. The files are located in the base directory <*install drive*>:\<*install directory*>*dialogic**samples**voice.*

The purpose of the Multithreaded GUI Based Application Program is to show how to use the API functions in an asynchronous voice application within the Windows programming environment. As is the case with most small Windows programs, the majority of the code is for graphical user interface support.

The Multithreaded GUI Based Application Program demonstrates the following functions:

- Set hook state
- Play VOX and WAV files
- Adjust playback volume
- Record to VOX and WAV files
- Play a user defined tone
- Make a call with or without PerfectCall
- Wait for digits

The Multithreaded GUI Based Application Program is a MDI application running with two threads. The main thread creates a child window for each voice device that is opened. The main thread will also initiate the selected voice function on the active voice device (foreground child window). This thread will print a brief message in the child window and will gray out menu items that should not be selected while a channel is busy.

The purpose of the second thread is to poll for device events using **sr_waitevt( )**. Upon receiving an event, a brief message is displayed in the child window and menu items are enabled for the next function selection.

The 'Options' Menu allows the user to select PerfectCall options, adjust playback volume, define tone for playback, and select WAVE recording options. These settings are on a per channel basis and are stored in the structure CHINFO. When a device is opened, its CHINFO structure is initialized with default settings.

NOTES: **1.** Linear PCM is not supported on the D/41D

2. Volume for playback can be adjusted at any time

10.4. Running the Multithreaded GUI Based Application Program for Voice Boards

Before running the Multithreaded GUI Based Application Program for Voice Boards, ensure that you perform the following:

1. Connect a Dialogic PromptMaster, a CO Simulator, or PBX to the Dialogic voice board.
2. Start the Dialogic boards using the Dialogic Configuration Manager.

To run the Multithreaded GUI Based Application Program for Voice Boards:

1. Select the *Multithreaded GUI Based Application* icon in the Dialogic Development Package folder, or change to the directory where the demo is located and type SAMPLE.EXE.
2. Select Open from the File Menu.
3. Click OK to open the channel (default: DxxxB1C1 - board 1, channel 1). You can open multiple channels concurrently.
4. Select Off-hook from the Function Menu.

You can now select any of the operations shown on the Function menu and modify their operation using the Options Menu.

Appendix A

Related Publications

- For information about the SCbus and SCbus routing, see the *SCbus Routing Guide* and the *SCbus Routing Software Reference*.
- For information about the Voice and Diagnostic Libraries and about library data structures, see the *Voice Software Reference: Programmer's Guide*.
- For information about the Standard Runtime Library, see the *Voice Software Reference: Standard Runtime Library*.
- For information about the digital interface device functions for the D/240SC-T1 and D/300SC-E1 boards, see the *Digital Network Interface Software Reference*.
- For information about the primary rate functions, see *the ISDN Software Reference.*

Glossary

ACK: A DTMF “A” (acknowledge) signal from the CPE or a DTMF “D” signal followed by a DTMF 1 through 5 sent as part of the FSK signal.

ADMF: ADSI Data Message Format

ADPCM: Adaptive Differential Pulse Code Modulation. A sophisticated compression algorithm for digitizing audio that stores the differences between successive samples rather than the absolute value of each sample. This method of digitization also reduces storage requirements from 64K bits/second to as low as 24K bits/second.

ADSI: Analog Display Services Interface. A Bellcore standard defining a protocol on the flow of information between a switch, a server, a voice mail system, a service bureau, or a similar device and a subscriber's telephone, PC, data terminal, or other communicating device with a screen. The idea of ADSI is to add words to a system that usually only uses touch tones. In a typical voice mail system, you call up and hear choices: “to listen to new messages, press 1, to hear saved messages, press 2,” etc. ADSI is designed to display the choices you're hearing on a screen attached to your phone. Your response is the same: a touch tone button. ADSI's signaling is DTMF and standard Bell 202 modem signals from the service to your 202-modem equipped phone. From the phone to the service it's only touch tone. ADSI works on every phone line in the world.

AGC: Automatic Gain Control. An electronic circuit used to maintain the audio signal volume at a constant level.

analog: 1. A method of telephony transmission in which the signals from the source (for example, speech in a human conversation) are converted into an electrical signal that varies continuously over a range of amplitude values analogous to the original signals. 2. Not digital signaling. 3. Used to refer to applications that use loop start signaling.

ANI: Automatic Number Identification.

Antares: A Dialogic open platform for easily incorporating speech recognition, Text-To-Speech, fax and many other DSP technologies. Dialogic PC-based expansion board with four TI floating point DSPs, SPOX DSP operating system, and the Antares board downloadable firmware and device driver.

ASCIIZ string: A null-terminated string of ASCII characters.

asynchronous function: A function that allows program execution to continue without waiting for a task to complete. To implement an asynchronous function, an application-defined event handler must be enabled to trap and process the completed event. Contrast with *synchronous function*.

AT bus: The common communication channel in a PC AT. The channel uses a 16-bit data path architecture, which allows up to 16 bits of data transfer. This bus architecture includes the standard PC bus plus a set of 36 lines for additional data transmission, addressing, and interrupt request handling.

AT: Used to describe an IBM or IBM-compatible Personal Computer (PC) containing an 80286 or higher microprocessor, a 16-bit bus architecture, and a compatible BIOS.

Automatic Gain Control: See *AGC*.

base memory address: A starting memory location (address) from which other addresses are referenced.

Bell 202: A 1200 bits per second (bps) FSK modem, developed by Bell Labs, used mainly for signaling between the CO and the CPE. It uses one carrier frequency and assigns a frequency for mark bits (1200 Hz) and space bits (2200 Hz) and is, by definition, phase continuous.

bit mask: A pattern which selects or ignores specific bits in a bit mapped control or status field.

bitmap: An entity of data (byte or word) in which individual bits contain independent control or status information.

board device: A board-level object that can be manipulated by a physical library. Board devices can be real physical devices, such as a D/4x board, or emulated devices, such as one of the D/4x boards that is emulated by a D/81A, D/12x or D/xxxSC board.

Board Locator Technology (BLT): Operates in conjunction with a rotary switch to determine and set non-conflicting slot and IRQ interrupt-level parameters, thus eliminating the need to set confusing jumpers or DIP switches.

bps: Bits Per Second

buffer: A block of memory or temporary storage device that holds data until it can be processed. It is used to compensate for the difference in the rate of the

flow of information (or time occurrence of events) when transmitting data from one device to another.

bus: An electronic path which allows communication between multiple points or devices in a system.

busy device: A device that is stopped, being configured, has a multitasking or non-multitasking function, or I/O function active on it.

cadence detection: A voice driver feature that analyzes the audio signal on the line to detect a repeating pattern of sound and silence.

cadence: A rhythmic sequence or pattern. Once established, it can be classified as a single ring, a double ring, or a busy signal by comparing the periods of sound and silence to establish parameters.

Cadenced Tone Generation: To generate a signal with single or dual tone elements; an enhancement to GTG.

Call Progress Analysis: The process used to automatically determine what happened after an outgoing call is dialed.

Call Status Transition Event Functions: Functions that set and monitor events on devices.

Caller ID: Calling Party Identification information.

CAS: CPE Alerting Signal

CCITT: Comite Consultatif Internationale de Telegraphique et Telephonique. One of the four permanent parts of the International Telecommunications Union, a United Nations agency based in Geneva. The CCITT is divided into three sections: 1. Study Groups set up standards for telecommunications equipment, systems, networks, and services. 2. Plan Committees develop general plans for the evolution of networks and services. 3. Specialized Autonomous Groups produce handbooks, strategies, and case studies to support developing countries.

channel: 1. When used in reference to a Dialogic expansion board that is analog, an audio path, or the activity happening on that audio path (for example, when you say the channel goes off-hook). 2. When used in reference to a Dialogic expansion board that is digital, a data path, or the activity happening on that data path. 3. When used in reference to a bus, an electrical circuit carrying control information and data.

channel device: A channel-level object that can be manipulated by a physical library, such as an individual telephone line connection. A channel is also a subdevice of a board. See *subdevice*.

checksum: A one byte entity for error detection, which is computed by transmitter and appended to the Message, and is computed by the receiver and compared to the sent checksum for basic error detection. Only one checksum is used per SDM or MDM message.

CLASS: Custom Local Area Signaling Services; a Caller ID standard published by Bellcore.

CO: Central Office. A local phone exchange. In general, "CO' refers to the phone network exchange that provides your phone lines. The term "Central Office" is used in North America. The rest of the world calls it PTT, for Post, Telephone and Telegraph. The telephone company facility where subscriber lines are linked, through switches, to other subscriber lines (including local and long distance lines).

computer telephony: The extension of computer-based intelligence and processing over the telephone network to a telephone. Lets you interact with computer databases or applications from a telephone and also enables computer-based applications to access the telephone network. Computer telephony makes computer-based information readily available over the world-wide telephone network from your telephone. Computer telephony technology incorporated into PCs supports applications such as: automatic call processing; automatic speech recognition; text-to-speech conversion for information-on-demand; call switching and conferencing; unified messaging that lets you access or transmit voice, fax, and E-mail messages from a single point; voice mail and voice messaging; fax systems including fax broadcasting, fax mailboxes, fax-on-demand, and fax gateways; transaction processing such as Audiotex and Pay-Per-Call information systems; call centers handling a large number of agents or telephone operators for processing requests for products, services or information; etc.

configuration file: An unformatted ASCII file that stores device initialization information for an application.

configuration functions: Functions that alter the configuration of devices.

convenience functions: Functions that simplify application writing.

CP: Control Processor

CPE: Customer Premise Equipment

CPE Alerting Signal: A special machine detectable DTMF signal.

CPU: Central Processing Unit

Data Link Layer: Layer 2 in ADSI, it is responsible for the first level of framing (or de-framing) of the data to be transmitted (or received). The Data Link Layer includes the appending (or checking) of Checksum/CRC data as well as preamble sequence generation (or removal).

data structure: Programming term for a data element consisting of fields, where each field may have a different type definition and length. A group of data structure elements usually share a common purpose or functionality.

DCM: Dialogic Configuration Manager. A utility with a graphical user interface (GUI) that enables you to add new boards to your system and work with board configuration data. You can: 1) view and modify configuration data for boards with Board Locator Technology; 2) add, modify and delete configuration data for hardware-configurable boards; 3) start and stop Dialogic System Service.

debouncing: Eliminating false signal detection by filtering out rapid signal changes. Any detected signal change must last for the minimum duration as specified by the debounce parameters before the signal is considered valid. Also known as deglitching.

deglitching: Eliminating false signal detection by filtering out rapid signal changes. Any signal change shorter than that specified by the deglitching parameters is ignored.

device: A computer peripheral or component controlled through a software device driver. A Dialogic voice and/or network interface expansion board is considered a physical board containing one or more logical board devices, and each channel or time slot on the board is a device.

device channel: A Dialogic voice data path that processes one incoming or outgoing call at a time (equivalent to the terminal equipment terminating a phone line). There are 4 device channels on a D/4x, 12 on a D/12x, 16 on a D/160SC-LS, 24 on a D/240SC or D/240SC-T1, 30 on a D/300SC-E1, and 32 on a D/320SC board. See **time slot** definition.

device driver: Software that acts as an interface between an application and hardware devices.

device handle: Numerical reference to a device, obtained when a device is opened using **xx_open( )**, where *xx* is the prefix defining the device to be opened. The device handle is used for all operations on that device.

device management functions: Functions that open and close devices.

device name: Literal reference to a device, used to gain access to the device via an **xx_open( )** function, where *xx* is the prefix defining the device to be opened.

Dialogic Configuration Manager: See DCM.

DIALOG/HD Series: Dialogic High Density products, including the D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1, and D/320SC, provide a powerful set of advanced computer telephony features that developers can use to create cost-efficient, high-density systems.

digital: Information represented as binary code.

digitize: The process of converting an analog waveform into a digital data set.

download: The process where board level program instructions and routines are loaded during board initialization to a reserved section of shared RAM.

downloadable SpringWare firmware: Software features loaded to Dialogic voice hardware. Features include voice recording and playback, enhanced voice coding, tone detection, tone generation, dialing, call progress analysis, voice detection, answering machine detection, speed control, volume control, ADSI support, automatic gain control, and silence detection.

driver: A software module which provides a defined interface between an application program and the firmware interface.

DSP-based boards: 1. Digital signal processor. A specialized microprocessor designed to perform speedy and complex operations with digital signals. 2. Digital signal processing.

DTMF digits: Dual Tone Multi Frequency. Push button or touch tone dialing based on transmitting a high and a low frequency tone identify each digit on a telephone keypad. The tones are (Hz):

1: 697,1209 2: 697,1336 3:697,1477
4: 770,1209 5: 770,1336 6: 770,1477
7: 852,1209 8: 852,1336 9: 852,1477
0: 941,1336 *: 941,1209 #: 941,1477

echo: The component of an analog device's receive signal reflected into the analog device's transmit signal.

echo cancellation: Removal of echo from an echo-carrying signal.

echo-cancelled signal: The output signal of an echo canceller after echo has been removed from the echo-carrying signal.

echo canceller: The software component responsible for performing echo cancellation.

echo-carrying signal: In regard to the **echo canceller**, a signal containing incoming speech data plus an echo component.

echo-producing circuit: Typically the interface between 4-wire (typically digital) and 2-wire (typically analog) circuits, which, due to impedance mismatches, reflects part of the receive signal into the transmit signal.

echo-reference signal: The signal that initially introduced echo into the echo-carrying signal. This signal is used by the **echo canceller** to estimate the echo component in the echo carrying signal.

ECR: Dialogic's Echo Cancellation Resource, consisting of three Voice library function APIs for implementing echo cancellation on a Dialogic voice channel device.

ECR mode: The operational mode for a Dialogic voice channel utilizing the Dialogic ECR feature at its highest level.

ECR_RX: The receive signal of the voice channel device's **echo canceller** containing the echo-reference signal.

ECR_TX: The transmit signal produced by the voice channel device's **echo canceller** containing the echo-cancelled signal.

EIA: Electronic Industry Association

emulated device: A virtual device whose software interface mimics the interface of a particular physical device, such as a D/4x boards that is emulated by a D/12x or a D/xxxSC board. On a functional level, a D/12x board is perceived by an application as three D/4x boards. See *physical device*.

event: An unsolicited or asynchronous message from a hardware device to an operating system, application, or driver. Events are generally attention-getting messages, allowing a process to know when a task is complete or when an external event occurs.

event handler: A portion of a Dialogic application program designed to trap and control processing of device-specific events. The rules for creating a DTI/1xx event handler are the same as those for creating a Windows signal handler.

event management functions: Class of device-independent functions (contained in the Standard Runtime Library) that connect events to application-specified event handlers, allowing users to retrieve and handle events that occur on the device. See *Standard Runtime Library (SRL Guide)*.

extended attribute functions: Class of functions that take one input parameter (a valid Dialogic device handle) and return device-specific information. For instance, a Voice device's Extended Attribute function returns information specific to the Voice devices. Extended Attribute function names are case-sensitive and must be in capital letters. See Standard Runtime Library.

firmware: A set of program instructions that reside on an expansion board.

flash: A signal which consists of a momentary on-hook condition used by the Voice hardware to alert a telephone switch. This signal usually initiates a call transfer.

frequency detection: A voice driver feature that detects the tri-tone Special Information Tone (SIT) sequences and other single-frequency tones for call progress analysis.

Frequency Shift Keying: A frequency modulation technique used to send digital data over voice band telephone lines.

FSK: Frequency Shift Keying

G.726: an international standard for encoding 8 kHz sampled audio signals for transmission over 16, 24, 32 and 40 kbps channels. The G.726 standard specifies an adaptive differential pulse code modulation (ADPCM) system for coding and decoding samples.

Global Dial Pulse Detection: Enables applications to detect dial pulses from rotary or pulse phones and use them as if they were DTMF digits.

Global Tone Detection: A feature that allows the creation and detection of user-defined tone descriptions on a channel by channel basis.

GSM: A speech compression algorithm developed for the Global System for Mobile telecommunication (GSM), Europe’s popular protocol suite for digital cellular communication.

hook state: A general term for the current line status of the channel: either on-hook or off-hook. A telephone station is said to be on-hook when the conductor loop between the station and the switch is open and no current is flowing. When the loop is closed and current is flowing the station is off-hook. These terms are derived from the position of the old fashioned telephone set receiver in relation to the mounting hook provided for it.

hook switch: The name given to the circuitry which controls on-hook and off-hook state of the Voice device telephone interface.

hybrid: See **echo-producing circuit**.

I/O: Input-Output

I/O Functions: Functions that transfer data to and from devices.

idle device: A device that has no functions active on it.

interrupt request level: A signal sent to the central processing unit (CPU) to temporarily suspend normal processing and transfer control to an interrupt handling routine. Interrupts may be generated by conditions such as completion of an I/O process, detection of hardware failure, power failures, etc.

IRQ: Interrupt ReQuest. A signal sent to the CPU to temporarily suspend normal processing and transfer control to an interrupt handling routine. A means of toggling between applications so that your system does not crash.

kernel: A set of programs in an operating system that implement the system's functions.

KTS: Key Telephone System

loop: The physical circuit between the telephone switch and the D/xxx board.

loop current: The current that flows through the circuit from the telephone switch when the Voice device is off-hook.

loop current detection: A voice driver feature that returns a connect after detecting a loop current drop.

loop start: In an analog environment, an electrical circuit consisting of two wires (or leads) called tip and ring, which are the two conductors of a telephone cable pair. The CO provides voltage (called "talk battery" or just "battery") to power the line. When the circuit is complete, this voltage produces a current called loop current. The circuit provides a method of starting (seizing) a

telephone line or trunk by sending a supervisory signal (going off-hook) to the CO.

loop-start interfaces: Devices, such as an analog telephones, that receive an analog electric current. For example, taking the receiver off-hook closes the current loop and initiates the calling process.

LSI/120: A Dialogic 12-line loop start interface expansion board.

Mbps: Million or Mega bits per second

Message Body: The portion of the SDM or MDM that does not include the Message Type byte nor the Checksum byte.

Message Assembly Layer: Layer 3 in ADSI, it is used to construct SDM, MDM, ADMF, or other valid messages, and transport them via the Data Link and Physical Layers to and from the CPE device.

MSI/SC: Modular Station Interface. An SCbus-based Dialogic expansion board that interfaces SCbus time slots to analog station devices.

NAK: Negative Acknowledgment. NAK is a control character in ASCII that means a packet arrived with the check digits in error. It is sent from the computer receiving the packets to the sender, implying that the packet should be retransmitted so that all bits will arrive intact next time.

NLP: Non Linear Processor. Operates on the output of the **echo canceller** to provide improved echo suppression as long as the echo-reference signal contains a speech signals and the echo-carrying signal does **not**.

off-hook: The state of a telephone station when the conductor loop between the station and the switch is closed and current is flowing. When a telephone handset is lifted from its cradle (or equivalent condition), the telephone line state is said to be off-hook.

PBX: A local premises or campus switch.

PC: Personal Computer. In this manual, the term refers to an IBM Personal Computer or compatible machine.

PCM Expansion Bus: See PEB.

PEB: PCM Expansion Bus. A Dialogic open platform, digital voice bus for electrically and digitally connecting different voice processing components. Information on the PEB is encoded using the Pulse Code Modulation (PCM)

method. Non-Dialogic products using PCM encoding may interface with Dialogic products by using this bus.

PerfectDigit: Dialogic SpringWare DTMF or MF signaling.

PerfectLevel: Dialogic SpringWare Volume control.

PerfectPitch: Dialogic SpringWare Speed control.

PerfectVoice: Dialogic SpringWare Enhanced voice coding.

physical device: A device that is an actual piece of hardware, such as a D/4x board; not an emulated device. See emulated device.

physical layer: Layer 1 of ADSI, it describes the electrical specifications of the interface, including FSK modem-based data transmission (reception) and in-band signaling.

polling: The process of repeatedly checking the status of a resource to determine when state changes occur.

PSTN/STN: Public or Private Switched Telephony Network

resource: Functionality (for example, voice-store-and-forward) that can be assigned to call. Resources are **shared** when functionality is selectively assigned to a call (usually via a PEB time slot) and may be shared among multiple calls. Resources are **dedicated** when functionality is fixed to the one call.

RFU: Reserved for future use.

ring detect: The act of sensing that an incoming call is present by determining that the telephone switch is providing a ringing signal to the Voice board.

route: Assign a resource to a time slot.

routing functions: For SCbus, functions that assign analog and digital channels to specific SCbus time slots; these SCbus time slots can then be connected to transmit or listen to other SCbus time slots. For PEB, functions that change the routing of channels to the time slots on the PCM Expansion Bus (PEB).

sampling rate: Frequency with which a digitizer takes measurements of the analog voice signal.

SCbus: Signal Computing Bus. A hardwired connection between Switch Handlers (SC2000 chips) on SCbus-based products. SCbus is a third generation TDM (Time Division Multiplexed) resource sharing bus that allows

information to be transmitted and received among resources over 1024 time slots.

SCbus routing functions: Functions that enable an application to connect or disconnect (make or break) the receive (listen) channel of a device to or from an SCbus time slot.

SCSA: See Signal Computing System Architecture.

Signal Computer System Architecture: SCSA. A Dialogic standard open development platform. An open hardware and software standard that incorporates virtually every other standard in PC-based switching. All signaling is out of band. In addition, SCSA offers time slot bundling and allows for scalability.

silence threshold: The level that sets whether incoming data to the Voice board is recognized as silence or non-silence.

Special Information Tones: SIT. (1) Standard Information Tones. Tones sent out by a central office to indicate that the dialed call has been answered by the distant phone. (2) Special Information Tone. Detection of a SIT sequence indicates an operator intercept or other problem in completing the call.

speed and volume control: Voice software that contains functions and data structures to control the speed and volume of play on a channel. The end user controls the speed or volume of a message by entering a DTMF tone.

speed and volume modification table: Each channel on a voice board has a table with twenty entries that allow for a maximum of ten increases and decreases in speed or volume, and one "origin" entry that represents regular speed or volume.

SpringBoard functions: Functions used on SpringBoard devices only.

SpringBoard: A Dialogic expansion board using digital signal processing to emulate the functions of other products. SpringBoard is a development platform for Dialogic products such as the D/120 and D/121.

SpringWare: Software algorithms build into the downloadable firmware that provides the voice processing features available on all Dialogic voice boards.

SRL: See *Standard Runtime Library.*

standard attribute functions: Class of functions that take one input parameter (a valid Dialogic device handle) and return generic information about the device.

For instance, Standard Attribute functions return IRQ and error information for all device types. Standard Attribute function names are case-sensitive and must be in capital letters. Standard Attribute functions for all Dialogic devices are contained in the Dialogic SRL. See *Standard Runtime Library*.

Standard Runtime Library: A Dialogic software resource containing Event-Management and Standard Attribute functions and data structures used by all Dialogic devices, but which return data unique to the device. See the *Voice Software Reference: Standard Runtime Library*.

station device: Any analog telephone or telephony device (such as a telephone or headset) that uses a loop-start interface and connects to an MSI/SC board.

string: An array of ASCII characters.

subdevice: Any device that is a direct child of another device. Since "subdevice" describes a relationship between devices, a subdevice can be a device that is a direct child of another subdevice, as a channel is a child of a board.

SVP mode: The standard voice processing mode for a Dialogic voice channel, where all standard Dialogic voice channel features are enabled and the default echo cancellation is provided.

synchronous function: Blocks program execution until a value is returned by the device. Also called a blocking function. Contrast with asynchronous function.

tap: The echo tail length that the echo canceller handles in the ECR mode is 16 ms, also referred to as 128 tap. SVP mode echo cancellation utilizes a 48 tap (6 ms) echo canceller.

termination condition: An event or condition which, when present, causes a process to stop.

termination event: An event that is generated when an asynchronous function terminates. See *asynchronous function*.

TIA: Telecommunications Industry Association

time slot assignment: The ability to route the digital information contained in a time slot to a specific analog or digital channel on an expansion board. See *device channe*l.

time slot: In a digital telephony environment, a normally continuous and individual communication (for example, someone speaking on a telephone) is (1) digitized, (2) broken up into pieces consisting of a fixed number of bits, (3) combined with pieces of other individual communications in a regularly repeating, timed sequence (multiplexed), and (4) transmitted serially over a single telephone line. The process happens at such a fast rate that, once the pieces are sorted out and put back together again at the receiving end, the speech is normal and continuous. Each individual pieced-together communication is called a time slot.

Transaction Record: Records two SCbus time slots from a single channel.

Universal Dialogic Diagnostic program: Software diagnostic routines for testing board-level functions of Dialogic hardware.

voice processing: Science of converting human voice into data that can be reconstructed and played back at a later time. Dialogic equipment can place 2-30 ports in one PC slot. They also use common APIs for scalability and the SCbus to connect to a broad range of technologies.

voice system: A combination of expansion boards and software that let you develop and run high-density voice processing applications.

Index

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

X

NOTES