Learn Mode API Software Reference

for Windows Operating Systems

05-1925-001

Publication Date: January, 2003

Intel Converged Communications, Inc.
1515 Route 10
Parsippany NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website:
http://developer.intel.com/design/telecom/support/

For **Products and Services Information**, visit the Intel Communications Systems Products website:
http://www.intel.com/network/csp/

For **Sales Offices**, visit the Intel Telecom Building Blocks Sales Offices page:
http://www.intel.com/network/csp/sales/

Table of Contents

List of Tables

About this Publication

Purpose

This document describes the learn mode API functions and data structures supported on Springware boards in the Windows operating system environment. It also provides programming guidelines for using the learn mode software.

Intended Audience

This document is intended for the following audience:

- Distributors
- System Integrators
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)
- End Users

How This Guide is Organized

This document assumes that you are familiar with and have prior experience with Windows operating systems and the C programming language.

The information this guide is organized as follows:

Chapter 1 provides an overview of the learn mode software, the API library functions and data structures.

Chapter 2 provides programming guidelines and techniques for developing an application to learn tones.

Chapter 3 discusses compiling and linking requirements such as include files and library files.

Chapter 4 provides an alphabetical reference to the learn mode functions.

Chapter 5 describes error handling and error codes that may be returned by the learn mode functions.

Chapter 6 provides an alphabetical reference to the data structures used by the library functions.

A **Glossary** and **Index** are provided at the end of this guide for reference.

Related Publications

You can download Intel® Dialogic® documentation (known as the online bookshelf) as part of the system release software installation or from the Telecom Support Resources website at *http://resource.intel.com/telecom/support/documentation/releases/index.htm* .

See the following Intel® Dialogic® documentation for more information:

- *Voice Software Reference - Features Guide*
- *Voice Software Reference - Programmer's Guide*
- *Voice Software Reference - Standard Runtime Library*
- Release Guide for the current system release, for information on release features, system requirements, hardware and software supported on this release, technical documentation, and more.
- Release Update for the current system release, for details on compatibility issues, restrictions and limitations, known problems, and late-breaking updates or corrections to the release documentation.

For definitions of domestic and international line status tones, see the following publications, respectively:

- Bellcore LATA Switching Systems Generic Requirements (LSSGR): Signaling, Section 6, Issue 2, Revision 1, December 1988 (TR-TSY-000506)
- International Telegraph and Telephone Consultative Committee (CCITT), Blue Book Volume II, Fascicle II.2: Telephone Network and ISDN – Operation, Numbering, Routing and Mobile Service, Recommendations E.100–E.333, Study Group II, November 1988 (ISBN 92-61-03261-3). Additional country-specific line status tones are defined by the NET 4 contributions.

1. Product Description

1.1. Introduction to Learn Mode

Computer telephony applications use Intel telecom boards that are often installed "behind" proprietary private branch exchange (PBX) or key systems. This type of configuration can pose challenges for the developer in terms of integrating the application with each vendor's individual PBX or key system. The challenge occurs because each switch does not always produce standard telephone company call progress tones such as busy, dial tone, or ringback.

To remedy the situation, the standard call progress tone detection algorithms within the telecom board must know these custom frequencies and cadences.

Although the PBX Expert/32 utility is available for learning tones, the learn mode software gives you the flexibility to integrate tone learning capability directly in your computer telephony applications. The PBX Expert/32 utility (previously called PBXpert/32) calls on the learn mode API under the hood to accomplish PBX tone learning. For more information on the PBX Expert/32 utility, see Intel Communications Systems Products website: *http://www.intel.com/network/csp/*.

The learn mode software is supported on specific boards and runs on Windows operating systems. It consists of learn mode library and header files, device drivers, firmware, and documentation to help you develop applications with tone learning capability. The learn mode library supplies functions that interface with the voice driver. For information on hardware that supports the learn mode API, see the release guide accompanying the system release.

The learn mode software works together with the voice library global tone detection feature to learn tones and capture tone descriptions. For more information on global tone detection, see the *Voice Software Reference – Features Guide*.

1.2. Learn Mode Functions

Use learn mode functions to initiate and operate learn mode to obtain a tone description.

The following learn mode functions are available:

Function Name	Definition
lm_clramp( )	• clears the tone amplitude structure (TN_AMP) for the **lm_LearnTone( )** function
lm_clrdesc( )	• clears the tone description structure (TN_DESC) for the **lm_LearnTone( )** function. The associated tone frequency structure (TN_FREQ) and tone duration structure (TN_DUR) are also cleared.
lm_clrparm( )	• clear learn mode parameters structure (LM_PARM) for the **lm_LearnTone( )** function
lm_LearnTone( )	• initiates learn mode to characterize a tone that occurs on the specified channel and to obtain the complete tone description for use with global tone detection or call progress analysis

1.3. Learn Mode Data Structures

The following data structures are used by learn mode functions:

Data Structure	Definition
TN_DESC (tone description)	• specifies the characteristics of a tone
TN_FREQ (tone frequency)	• specifies frequency information for a tone and is a member of the TN_DESC structure
TN_DUR (tone duration)	• specifies cadence information for a tone and is a member of the TN_DESC structure
LM_PARM (learn mode parameters)	• specifies the parameters and options for learning a tone
TN_AMP (tone amplitude)	• specifies amplitude information used to learn a tone

Data Structure	Definition
TN_INFO (tone information)	• reports the data on which learn mode bases the final tone description
TN_INFOLIST (tone information list)	• reports the data for each frame (sample) and includes the TN_INFO data structure

2. Application Development Guidelines

2.1. Mode of Operation

The learn mode library functions operate in synchronous (blocking) mode. Synchronous functions perform an action on a device and do not return control to the calling process until the action is completed.

2.2. Learn Mode Requirements and Limitations

The following requirements apply to learn mode:

- The tone to be learned must be either a continuous tone or a repeating tone.
- To learn a continuous tone, the tone must occur continuously for a minimum duration of one second without any other tone being present. (If another tone is present, you may be able to exclude it from being learned by restricting the **lm_LearnTone( )** learning parameters specified in **tn_rangep**.) The duration of the tone is configurable in the LM_PARM structure.
- To learn a tone having a cadence, several repetitions of the tone must occur without any other tone being present. The cadence must contain at least five tone-on/tone-off transitions, although 10 transitions will characterize the tone with more accuracy. (The default is 10.)

The following limitations apply to learn mode:

- Learn mode requires that the channel be completely dedicated to its operations while learn mode is active. Simultaneous use of any other signal or tone detection on the channel is not compatible, including DTMF, MF, R2 MF, Socotel, user-defined tone detection, PerfectCall call progress analysis, and basic call progress analysis. However, you can perform these operations on another channel and route the learn mode channel's receive time slot to the other channel's time slot.
- Since learn mode can characterize either a tone that has several cadence repetitions or a tone that is continuous, it is invalid to use learn mode to characterize the following:

  - A sequence of different tones, such as a SIT tri-tone sequence or a PBX warble tone (see call progress analysis to detect SIT tones).
  - Short duration, single instance tones that do not repeat, such as a “bong” tone (unless the duration is long enough to qualify as a continuous tone).

  In these cases, to characterize a tone that occurs in a single instance or is one in a sequence, you must simulate the repetitions by recording the tone and then playing the recorded tone repeatedly. (Of course, the simulation will be more robust if you concatenate multiple recordings of the tone).

- Learn mode can accurately characterize almost all tones having a complex cadence. In the laboratory, it is possible to create a situation where the initial definition of a complex cadence overlaps with another similar tone description. In this unusual case, adjusting the tone description can correct the problem. Note that such a conflict is extremely rare.

2.3. Hints for Using Learn Mode

The following hints are provided to help you use learn mode:

- If your final tone description is being falsely triggered by noise or voice, you can:
  - Change the qualification ID (by setting lm_qualid field in LM_PARM structure) for more immunity to noise and less sensitivity to the tone. See *6.2. LM_PARM: Learn Mode Parameters* for more information on how to set this value.
  - Increase the repetition count if the tone has a simple cadence. (If you increase the repetition count and then fail to detect the tone, your tone description may be for a complex-cadence tone; in this case, you must set the repetition count to 1.)
- To learn a particular tone, you may need to change the learning parameters. For example, when learning a cadence tone that contains a limited number of repetitions, you can reduce the number of learning samples or cadence repetitions specified in lm_frames field in LM_PARM structure. Another example is when learning a short continuous tone, you can reduce the continuous tone minimum on-time specified in lm_cnt_min in LM_PARM structure.
- If you get an EDX_TNINFO error, which means there is not enough tone information to do the learning, try increasing the number of frames in

lm_frames field in LM_PARM structure. If you get an EDX_TNINVALID error, which means an invalid tone was detected, try lowering the lm_qualid field in LM_PARM structure. Learn mode requires that at least 5 frames/samples be used.

- The Learn Mode library uses 4 seconds as the default duration for learning continuous tones (rather than the minimum of 1 second). There are two reasons why the default duration for learning a continuous tone is 4 seconds: one is for learning the tone and the other is for using the final tone description for detection.

  A 4-second minimum duration is needed for learn mode to determine whether the tone is cadence or continuous (rarely does a cadence tone have a 4-second tone-on duration). If you want to learn a continuous tone in less than 4 seconds, you can set the lm_cadflag field in LM_PARM structure to 2 to specify a continuous tone, and then set the lm_cnt_min field to a value under 400 (10 ms units).

  A 4-second minimum duration may also be needed when using the final tone description for the continuous tone for detection. To prevent the continuous tone final tone description from detecting a cadence tone that has the same frequency range, the tone-on duration for the continuous tone must be greater than the tone-on duration for the cadence tone. A 4-second minimum duration helps prevent a detection overlap with a cadence tone. In this case, you may want to set a short minimum on-time to learn the frequency of the continuous tone, and then increase the on-time when you build the tone for detection. *The continuous tone on-time must be greater than any cadence tone on-time of the same frequency.*

3. Building Applications

3.1. Include Files

Function prototypes and equates are defined in include files, also known as header files. Applications that use Intel Dialogic library functions must contain statements for include files in this form, where filename represents the include file name:

```
#include <filename.h>
```

The following header files must be included in application code **in the order shown** prior to calling Intel Dialogic library functions:

- *srllib.h* – Standard Runtime Library (SRL) header file. This header file is used for all application development.
- *dxxxlib.h* – voice library header file. Include this header file in applications that use voice library functions (typically, those that begin with dx_).
- *lmodelib.h* – learn mode library header file. Include this header file in applications that use learn mode library functions (typically, those that begin with lm_).

NOTE: *srllib.h* must be included in code before all other Intel Dialogic header files.

By default, the header files are located in the following directory: \<*install directory*>*dialogic**inc*.

3.2. Required Libraries

Simple C language interfaces in source-code format are provided to each individual technology DLL (such as standard runtime, voice, learn mode). These C language interfaces allow an application to perform run-time linking instead of compile-time linking.

NOTE: Compile-time linking requires that all functions called in an application be contained in the DLL that resides on the system.

You must link the following library files **in the order shown** when compiling your application:

- *liblmode.lib* learn mode library file
- *libdxxmt.lib* – main voice library file
- *libsrlmt.lib* – Standard Runtime Library file

By default, the library files are located are located in the following directory: \<*install directory*>*dialogic**lib*.

4. Function Reference

This chapter provides an alphabetical reference to the functions contained in the learn mode library.

The following information is included to describe the function:

- Reference header information
- Description
- Cautions
- Example
- See Also (list of related functions, when applicable)

Additional sections may be included as needed.

Name:	void lm_clramp(tn_amplp)	
Inputs:	TN_AMP *tn_amplp	• pointer to the TN_AMP
Outputs:	none	
Returns:	none	
Includes:	srllib.h dxxxlib.h lmodelib.h	
Mode:	synchronous	
Category:	learn mode	
Platform:	Springware	

■ Description

The **lm_clramp( )** function clears the tone amplitude structure (TN_AMP). The function initializes all fields in the structure to their default value.

Call this function before setting values in the **lm_LearnTone( )** function **tn_amplp** parameter.

Parameter	Description
tn_amplp	Points to the TN_AMP structure that is to be cleared.

■ Cautions

None.

■ Errors

For a list of error codes, see *Section 5.2. Learn Mode Error Codes*.

■ Example

```
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include "srllib.h"
#include "dxxxlib.h"
#include "lmodelib.h"

#define DEVICE "dxxxB1C1"
#define ERR -1
```

```
int lrn_Learn(int lrnDev);

int main()
{
int i,rc;
int Caller;


/* learn the dial tone here */

/* Prepare the scenario for dial tone */

if ((Caller  = dx_open(DEVICE,(int)NULL)) == ERR)
{
//Error occurred in opening DEVICE
//Handle the error here and return
//error
return(ERR);
}
if ( dx_sethook(Caller, DX_OFFHOOK, EV_SYNC) == ERR)
{
//Error to set off hook
return(ERR);
}

printf("Start learning..............\n");
if ((rc = lrn_Learn(Caller)) != ERR)
{
printf("Finish learning..............\n");
}
else
{
printf("Error in Learning...\n");
}
//reset the channel
return 0;
}

int lrn_Learn(int Dev)
{
   LM_PARM parm;
   TN_AMP amp;
   TN_DESC desc;
   short flagDual;
int ret = -1;

lm_clrparm(&parm);
lm_clramp(&amp);
lm_clrdesc(&desc);

flagDual = 0; //Unknown
//set necessary fields of lm_parm
parm.lm_frames = 10;
parm.lm_qualid = QT_LMMID;
parm.lm_cadflag = 0;
/* min duration for tone to qualify as continuous */
parm.lm_cnt_min = 400;
parm.lm_method = 2;

ret = lm_LearnTone(Dev, &parm, &amp, &flagDual, NULL, &desc, NULL, EV_SYNC, NULL);

if( ret == ERR )
{
//Learning failed so process and return error
```

```
return(ERR):
}
/*process tone ( frequency and cadence) information returned in desc structure. */

return(ret);
}
```

See Also

- **lm_LearnTone( )**
- **lm_clrdesc( )**
- **lm_clrparm( )**

Name:	void lm_clrdesc(tn_descp)	
Inputs:	TN_DESC *tn_descp	• pointer to the TN_DESC
Outputs:	none	
Returns:	none	
Includes:	srllib.h dxxxlib.h lmodelib.h	
Mode:	synchronous	
Category:	learn mode	
Platform:	Springware	

Description

The **lm_clrdesc( )** function clears the tone description structure (TN_DESC), including the associated tone frequency structure (TN_FREQ) and tone duration structure (TN_DUR). The function initializes all fields in the structure to their default value.

Call this function to clear **lm_LearnTone( )** function **tn_rangep** and **tn_tonep** parameters.

Parameter	Description
tn_descp	Points to the TN_DESC structure that is to be cleared.

Cautions

None.

Errors

For a list of error codes, see *Section 5.2. Learn Mode Error Codes*.

Example

```
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include ″srllib.h″
#include "dxxxlib.h″
#include "lmodelib.h"
```

```
#define DEVICE "dxxxB1C1"
#define ERR  -1

int lrn_Learn(int lrnDev);

int main()
{
int i,rc;
int Caller;

/* learn the dial tone here */

/* Prepare the scenario for dial tone */

if ((Caller  = dx_open(DEVICE,(int)NULL)) == ERR)
{
//Error occurred in opening DEVICE
//Handle the error here and return
//error
return(ERR);
}
if ( dx_sethook(Caller, DX_OFFHOOK, EV_SYNC) == ERR)
{
//Error to set off hook
return(ERR);
}

printf("Start learning.............\n");
if ((rc = lrn_Learn(Caller)) != ERR)
{
printf("Finish learning.............\n");
}
else
{
printf("Error in Learning...\n");
}
//reset the channel
return 0;
}

int lrn_Learn(int Dev)
{
   LM_PARM parm;
   TN_AMP amp;
   TN_DESC desc;
   short flagDual;
int ret = -1;

lm_clrparm(&parm);
lm_clramp(&amp);
lm_clrdesc(&desc);

flagDual = 0;  //Unknown
//set necessary fields of lm_parm
parm.lm_frames = 10;
parm.lm_qualid = QT_LMMID;
parm.lm_cadflag = 0;
/* min duration for tone to qualify as continuous */
parm.lm_cnt_min = 400;
parm.lm_method = 2;

ret = lm_LearnTone(Dev, &parm, &amp, &flagDual, NULL, &desc, NULL, EV_SYNC, NULL);

if( ret == ERR )
```

```
{
//Learning failed so process and return error
return(ERR):
}
/*process tone ( frequency and cadence) information returned in desc structure. */

return(ret);
}
```

See Also

- **lm_LearnTone( )**
- **lm_clramp( )**
- **lm_clrparm( )**

Name:	void lm_clrparm(lm_parmp)
Inputs:	LM_PARM *lm_parmp • pointer to the LM_PARM
Outputs:	none
Returns:	none
Includes:	srllib.h dxxxlib.h lmodelib.h
Mode:	synchronous
Category:	learn mode
Platform:	Springware

■ Description

The **lm_clrparm( )** function clears the learn mode parameters structure (LM_PARM). The function initializes all fields in the structure to their default value.

Call this function before setting values in the **lm_LearnTone( )** function **lm_parmp** parameter.

Parameter	Description
lm_parmp	Points to the LM_PARM structure that is to be cleared.

■ Cautions

None.

■ Errors

For a list of error codes, see *Section 5.2. Learn Mode Error Codes*.

■ Example

```
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include "srllib.h"
#include "dxxxlib.h"
#include "lmodelib.h"

#define DEVICE "dxxxB1C1"
#define ERR  -1
```

```
int lrn_Learn(int lrnDev);

int main()
{
int i,rc;
int Caller;

/* learn the dial tone here */

/* Prepare the scenario for dial tone */

if ((Caller  = dx_open(DEVICE,(int)NULL)) == ERR)
{
//Error occurred in opening DEVICE
//Handle the error here and return
//error
return(ERR);
}
if ( dx_sethook(Caller, DX_OFFHOOK, EV_SYNC) == ERR)
{
//Error to set off hook
return(ERR);
}

printf("Start learning..............\n");
if ((rc = lrn_Learn(Caller)) != ERR)
{
printf("Finish learning..............\n");
}
else
{
printf("Error in Learning...\n");
}
//reset the channel
return 0;
}

int lrn_Learn(int Dev)
{
   LM_PARM parm;
   TN_AMP amp;
   TN_DESC desc;
   short flagDual;
int ret = -1;

lm_clrparm(&parm);
lm_clramp(&amp);
lm_clrdesc(&desc);

flagDual = 0;  //Unknown
//set necessary fields of lm_parm
parm.lm_frames = 10;
parm.lm_qualid = QT_LMMID;
parm.lm_cadflag = 0;
/* min duration for tone to qualify as continuous */
parm.lm_cnt_min = 400;
parm.lm_method = 2;

ret = lm_LearnTone(Dev, &parm, &amp, &flagDual, NULL, &desc, NULL, EV_SYNC, NULL);

if( ret == ERR )
{
//Learning failed so process and return error
return(ERR):
```

```
}
/*process tone ( frequency and cadence) information returned in desc structure. */

return(ret);
}
```

See Also

- **lm_LearnTone( )**
- **lm_clrdesc( )**
- **lm_clramp( )**

Name:	long **lm_LearnTone**(cd, lm_parmp, tn_amplp, dflagp, tn_rangep, tn_tonep, tn_infop, mode)	
Inputs:	int cd	• channel device handle
	LM_PARM *lm_parmp	• pointer to the LM_PARM structure
	TN_AMP *tn_amplp	• pointer to the TN_AMP structure
	short *dflagp	• pointer to single/dual tone flag
	TN_DESC *tn_rangep	• pointer to the TN_DESC structure specifying an optional learning range
	TN_DESC *tn_tonep	• pointer to the TN_DESC structure specifying the tone description
	TN_INFOLIST *tn_infolistp	• pointer to the TN_INFOLIST structure
	int mode	• synchronous
Outputs:	none	
Returns:	0 if success -1 if failure	
Includes:	srllib.h dxxxlib.h lmodelib.h	
Mode:	synchronous	
Category:	learn mode	
Platform:	Springware	

■ Description

The **lm_LearnTone( )** function initiates learn mode to characterize a tone that occurs on the specified channel and to obtain the complete tone description for use with global tone detection or call progress analysis. For more information on global tone detection and call progress analysis, see the *Voice Software Reference – Features Guide*.

Parameter	Description
cd	Specifies the channel device handle.
lm_parmp	Points to the LM_PARM data structure which specifies parameters used to characterize the tone. For more

Parameter	Description
	information, see the LM_PARM structure in *Chapter 6. Data Structure Reference*.
tn_amplp	Points to the TN_AMP data structure. This specifies tone amplitude boundaries that restrict the learning to a specified amplitude range. For example, you can set the amplitude range lower to learn a poor quality tone, although noise may interfere with the results, or you can set the amplitude range higher to detect a high-quality tone. For more information, see TN_AMP structure in *Chapter 6. Data Structure Reference*.
dflagp	Specifies the single/dual tone flag, which indicates whether the tone is a single tone or dual tone. This flag must be set manually. When you set this flag manually, make sure you take the voice board dual-tone resolution into account. 0 If unknown or new tone 1 single tone (or a dual tone that falls below the dual-tone resolution) 2 dual tone
tn_rangep	Points to the TN_DESC data structure specifying optional tone learning boundaries to restrict learning to the specified range. For more information, see TN_DESC structure in *Chapter 6. Data Structure Reference*. If you set **tn_rangep** to NULL, this feature is disabled.
tn_tonep	Points to the TN_DESC data structure specifying the tone description. This parameter serves two different purposes: (1) it specifies an existing tone description as optional input to the **lm_LearnTone( )** function, and learn mode incorporates this input into the final tone description, and (2) it provides the final tone description that is returned by **lm_LearnTone( )**. See Requirements section for more information on using this parameter.
tn_infolistp	Points to the TN_INFOLIST structure which contains an array of tone information (TN_INFO data structure). This structure provides the actual frequency and on/off times for each frame (sample) of a detected tone. This information is used primarily

Parameter	Description
	for debugging. If you set **tn_infolistp** to NULL, this feature is disabled, and the data is not available for your analysis. If you want to analyze the data on which learn mode bases the final tone description, you must allocate an array of memory for saving the tone information of each sample. The required memory is equal to the number of lm_frames (stored in LM_PARM structure) + OFFSET (defined in *lmodelib.h*) multiplied by the size of the TN_INFO structure. OFFSET allows the library to store tone information for extra frames. The **tn_infolistp** parameter is also used to learn disconnect tones with more than one on time and more than one off time.
mode	You must set the mode to EV_SYNC for synchronous operation.

■ Requirements

- Before executing the **lm_LearnTone( )** function to learn a **new** tone, you must set **tn_tonep** to zeros and also set the **dflagp** location to indicate a flag value for a single or dual tone. You can use **lm_clrdesc( )** to set **tn_tonep** to zeros (this function clears the LM_DESC structure).
- Before executing the **lm_LearnTone( )** function to learn an **existing** tone, you must specify in **tn_tonep** the complete tone description to be adjusted and also set the **dflagp** location to specify a flag value for a single or dual tone.
- If you set the **tn_rangep** fields to restrict learning to a specified frequency or cadence range, you must set the lm_cadflag field in LM_PARM structure to indicate a cadence or continuous tone.
- The tone to be learned must either be a continuous tone with a minimum duration of one second or a cadence tone with a minimum of five tone-on/off transitions (although 10 transitions will characterize the tone with more accuracy).

  NOTE: If you execute this function and 10 seconds elapse without a tone being present for the minimum duration as previously described, the function returns -1 to indicate failure and last error is set

EDX_TIMEOUT. Last error can be retrieved using **ATDV_LASTERR( )**.

See the *Voice Software Reference – Features Guide* for details on global tone detection.

The tone description is returned in the TN_DESC structure specified by **tn_tonep** upon completion of **lm_LearnTone( )**.

- To learn a disconnect tone with more than one on time and more than one off time, use the TN_INFOLIST structure and find the on time of the longest duration for the tone in the first pass. In the second pass, use **lm_parmp** parameter, set lm_cnt_min field of LM_PARM structure to 5% less than the on time found in the previous pass, and then call **lm_LearnTone( )** again to learn the tone.
- After using the **lm_LearnTone( )** function to obtain a complete tone description for the tone, you can use this tone information to change the default tone detection definitions or you can create a new tone and add it to the tone template. For more information, see the global tone detection topic in the *Voice Software Reference – Features Guide.*

■ Cautions

- Before using the **lm_LearnTone( )** function, you should clear existing data structure field values using the **lm_clramp( )**, **lm_clrdesc( )**, and **lm_clrparm( )** functions.
- Learn mode requires that the channel be completely dedicated to its operations while learn mode is active. Simultaneous use of any other signal or tone detection on the channel is not compatible, including DTMF, MF, R2 MF, Socotel, user-defined tone detection, and call progress analysis. However, you can perform these operations on another channel and route the other channel's receive time slot to the time slot of the channel being used for learning.

■ Errors

For a list of error codes, see *Section 5.2. Learn Mode Error Codes*.

■ Example

```
/* This example code shows how to "learn" a dial tone. */

/*
      The line is set off-hook to generate a dial tone,
      then lm_LearnTone( ) is called to learn the tone.
/*


#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include ”srllib.h”
#include “dxxxlib.h”
#include "lmodelib.h"

#define DEVICE "dxxxB1C1"
#define ERR  -1

int lrn_Learn(int lrnDev);

int main()
{
int i,rc;
int Caller;

/* learn the dial tone here */

/* Prepare the scenario for dial tone */

if ((Caller  = dx_open(DEVICE,(int)NULL)) == ERR)
{
//Error occurred in opening DEVICE
//Handle the error here and return
//error
return(ERR);
}
if ( dx_sethook(Caller, DX_OFFHOOK, EV_SYNC) == ERR)
{
//Error to set off hook
return(ERR);
}

printf("Start learning..............\n");
if ((rc = lrn_Learn(Caller)) != ERR)
{
printf("Finish learning..............\n");
}
else
{
printf(“Error in Learning...\n”);
}
//reset the channel
return 0;
}

int lrn_Learn(int Dev)
{
   LM_PARM parm;
   TN_AMP amp;
   TN_DESC desc;
   short flagDual;
int ret = -1;
```

```
lm_clrparm(&parm);
lm_clramp(&amp);
lm_clrdesc(&desc);

flagDual = 0;   //Unknown
//set necessary fields of lm_parm
parm.lm_frames = 10;
parm.lm_qualid = QT_LMMID;
parm.lm_cadflag = 0;
/* min duration for tone to qualify as continuous */
parm.lm_cnt_min = 400;
parm.lm_method = 2;

ret = lm_LearnTone(Dev, &parm, &amp, &flagDual, NULL, &desc, NULL, EV_SYNC, NULL);

if( ret == ERR )
{
//Learning failed so process and return error
return(ERR):
}
/*process tone ( frequency and cadence) information returned in desc structure. */

return(ret);
}
```

See Also

- **lm_clramp( )**
- **lm_clrdesc( )**
- **lm_clrparm( )**

5. Error Handling and Error Codes

5.1. Error Handling

All learn mode library functions return a value to indicate success or failure of the function. A return value of zero or a non-negative number indicates success. A return value of -1 indicates failure.

If a library function fails, call the standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to determine the reason for failure. For more information on these functions, see the *Voice Software Reference - Standard Runtime Library*.

NOTE: If **ATDV_LASTERR( )** returns the EDX_SYSTEM error code, an operating system error has occurred. Check the global variable **errno** contained in *errno.h*. Use **dx_fileerrno( )** to obtain the system error value.

For a list of errors that can be returned by a learn mode library function, see 5.2. Learn Mode Error Codes.

5.2. Learn Mode Error Codes

Learn mode error codes can be found in *lmodelib.h* and include the following:

Name	Description
EDX_TNINFO	Not enough tone information to do the learning (requires 5 frames/samples)
EDX_TNINVALID	Invalid tone detected
EDX_FRAMES	Invalid number of frames (**lm_frames**) in LM_PARM
EDX_CONTIM	Invalid continuous tone minimum on-time (**lm_cnt_min**) in LM_PARM
EDX_CADFLAG	Invalid continuous/cadence flag (**lm_cadflag**) in LM_PARM

Name	Description
EDX_LMPARM	Invalid parameters in LM_PARM
EDX_TNAMP	Invalid tone amplitude in TN_AMP
EDX_LMMETHOD	Invalid frequency/tolerance learning method (**lm_method**) in LM_PARM
EDX_FREQTOL	Invalid learn mode frequency tolerance (**lm_frq_tol**) in LM_PARM
EDX_LMDURTOL	Invalid learn mode duration tolerance (**lm_dur_tol**) in LM_PARM
EDX_TNDFLAG	Invalid dual/single tone flag (**dflagp**)
EDX_TNPARM	Invalid tone template parameter
EDX_LMQUALID	Invalid learn mode qualification template ID (**lm_qualid**) in LM_PARM

6. Data Structure Reference

6.1. Data Structures Overview

This chapter provides an alphabetical reference to the data structures used by the learn mode library functions:

- LM_PARM – learn mode parameters structure
- TN_AMP – tone amplitude structure
- TN_DESC – tone description structure
- TN_DUR – tone duration structure
- TN_FREQ – tone frequency structure
- TN_INFO – tone information structure
- TN_INFOLIST – tone information list structure

6.2. LM_PARM: Learn Mode Parameters

The LM_PARM structure is declared as follows:

```
typedef struct {
   unsigned short int lm_cadflag;    /* cadence/continuous tone flag    */
   unsigned short int lm_cnt_min;    /* min on-time for continuous tone */
   unsigned short int lm_frames;     /* number of tone on/off samples   */
   unsigned short int lm_qualid;     /* qualification ID used           */
   unsigned short int lm_method;     /* method for freq/dur tolerances  */
   float              lm_frq_tol;    /* frequency parm for tolerance    */
   float              lm_dur_tol;    /* duration parm for tolerance     */
} LM_PARM;
```

The learn mode parameters structure (LM_PARM) specifies the parameters and options for learning a tone and is used for input to the **lm_LearnTone( )** function. These options include the continuous tone minimum on-time, the number of learning samples, the qualification ID (which controls the immunity to noise), and the methods and values for calculating the frequency and cadence range for the final tone description.

NOTE: Set any field to 0 to use the default value.

Use the **lm_clrparm( )** function to clear the fields of an LM_PARM structure.

Table 1. LM_PARM Structure

Field	Description
lm_cadflag	Length: 2 (unsigned short int) Default: 0
	Continuous or cadence mode. Specifies whether the tone to be learned is continuous or has a cadence. If you set the **tn_rangep** parameter of **lm_LearnTone( )** function to restrict learning to a specified range, you must set the lm_cadflag field to indicate a cadence or continuous tone.
	0 To learn a tone when you don't know whether the tone is continuous or cadence.
	1 To learn a tone having a cadence.
	2 To learn a continuous tone.
lm_cnt_min	Length: 2 (unsigned short int) Units: 10 ms. Range: 1 – 1000 Default: 400 (4 seconds)
	Continuous tone minimum on-time. Specifies in 10 ms units the tone-on duration for the tone to qualify as a continuous tone. After the tone qualifies, sampling is performed for the number of frames (samples) specified in lm_frames.
	Set this field to 0 to use the default value (400).
lm_frames	Length: 2 (unsigned short int) Default: 10 Range: 5 – 127
	Number of learning samples. For a cadence tone, this specifies the number of tone-on/tone-off transitions for learning the cadence. For a continuous tone, this specifies the number of samples taken after the tone qualifies as a continuous tone (see lm_cnt_min) and on which the final frequency range is based. It is recommended that you set lm_frames to a minimum of 30 if you are using lm_method LM_STATISTICAL.
	Set this field to 0 to use the default value (10).

Field	Description	
lm_qualid	Length: 2 (unsigned short int) Default: QT_LMDEF (sets to QT_LMMID)	
	Qualification ID. Specifies the qualification template ID used for learning. The qualification ID specified here is also output in the tn_qualid field of the final tone description in the TN_DESC structure. Increase the qualification ID for more immunity to noise and less sensitivity to the tone.	
	QT_LMDEF	Sets the qualification ID to the default (middle).
	QT_LMLOW	Low qualification (poor line quality).
	QT_LMMID	Middle qualification (medium line quality).
	QT_LMHIGH	High qualification (high line quality).
lm_method	Length: 2 (unsigned short int) Default: DEF_METHOD (sets to LM_RATIO)	
	User-selectable tolerance method. Indicates the method used to calculate the frequency range and cadence range needed to detect the tone.	
	DEF_METHOD	Sets to the default method, LM_RATIO.
	LM_ABSOLUTE	Absolute Tolerance (constant value) takes the samples with the highest and lowest values and then adds and subtracts the constant value specified in lm_frq_tol or lm_dur_tol.
	LM_RATIO	Ratio Tolerance (percentage) takes the samples with the highest and lowest values and then adds and subtracts the percentage specified in lm_frq_tol or lm_dur_tol.

Field	Description	
	LM_STATISTICAL	Statistical Tolerance method based on calculating the mean and variance of the samples; lm_frq_tol and lm_dur_tol specify the percentage of extreme samples to exclude; places less weight on a "bad" sample (aberration or extreme deviation); preferable method when the tone data is poor. For good results with this method, we recommend that you set lm_frames to a minimum of 30.
lm_frq_tol	Length: 4 (float)	
	Frequency Tolerance. Specifies a value used in calculating the frequency tolerance. The default, range, and units depend upon the tolerance lm_method selected.	
	LM_ABSOLUTE	Units: Hz Range: 0 – 300 Default: 30 Hz Specifies in Hz a constant value to add to and subtract from the samples with the highest and lowest values to obtain the frequency detection range.
	LM_RATIO	Units: Percentage Range: 0 – 1 Default: 0.06 Specifies a percentage to add to and subtract from the samples with the highest and lowest values to obtain the frequency detection range.
	LM_STATISTICAL	Units: Percentage Range: 0 – 0.2 Default: 0.03

Field	Description	
		Specifies the percentage of extreme samples to exclude from the normal distribution.
lm_dur_tol	Length: 4 (float)	
	Duration Tolerance. Specifies a value used in calculating the tolerance for the tone cadence (tone-on/tone-off durations). The default, range, and units depend upon the tolerance lm_method selected.	
	LM_ABSOLUTE	Units: 10 ms Range: 0 – 100 Default: 3.0
		Specifies in units of 10 ms a constant value to add to and subtract from the samples with the highest and lowest values to obtain the cadence detection range.
	LM_RATIO	Units: Percentage Range: 0 – 1 Default: 0.08
		Specifies a percentage to add to and subtract from the samples with the highest and lowest values to obtain the cadence detection range.
	LM_STATISTICAL	Units: Percentage Range: 0 – 0.2 Default: 0.03
		Specifies the percentage of extreme samples to exclude from the normal distribution.

6.3. TN_AMP: Tone Amplitude

The TN_AMP structure is declared as follows:

```
typedef  struct  {
   short int  tn_fq1_minamp;         /* 1st tone min. amplitude (dB) */
   short int  tn_fq1_maxamp;         /* 1st tone max. amplitude (dB) */
   short int  tn_fq2_minamp;         /* 2nd tone min. amplitude (dB) */
   short int  tn_fq2_maxamp;         /* 2nd tone max. amplitude (dB) */
} TN_AMP;
```

The TN_AMP structure specifies tone amplitude boundaries. It is used by learn mode for input to the **lm_LearnTone( )** function to restrict the learning to a specified amplitude range. For example, you can set the amplitude range lower to learn a poor-quality tone, although noise may interfere with the results, or you can set the amplitude range higher to detect a high-quality tone.

When setting the amplitude range for a dual tone, it is typical to use the same amplitude range for both frequencies.

NOTE: Set any field to 0 to use the default value.

Use the **lm_clramp( )** function to clear the fields of a TN_AMP structure.

Table 2. TN_AMP Structure

Field	Description
tn_fq1_minamp	Length: 2 (short int) Default: -42 Range: -42 dBm to 0 dBm Units: dBm First frequency minimum amplitude. Specifies in dBm the minimum amplitude of the tone with the lower frequency. The value 0 sets the amplitude to the default (-42).
tn_fq1_maxamp	Length: 2 (short int) Default: 0 Range: -42 dBm to 0 dBm Units: dBm First frequency maximum amplitude. Specifies

Field	Description
	in dBm the maximum amplitude of the tone with the lower frequency. The value 0 sets the amplitude to the default (-0).
tn_fq2_minamp	Length: 2 (short int) Default: -42 Range: -42 dBm to 0 dBm Units: dBm Second frequency minimum amplitude. Specifies in dBm the minimum amplitude of the tone with the higher frequency. The value 0 sets the amplitude to the default (-42).
tn_fq2_maxamp	Length: 2 (short int) Default: 0 Range: -42 dBm to 0 dBm Units: dBm Second frequency maximum amplitude. Specifies in dBm the maximum amplitude of the tone with the higher frequency. The value 0 sets the amplitude to the default (-0).

6.4. TN_DESC: Tone Description

The TN_DESC structure is declared as follows:

```
typedef struct {
   TN_FREQ            tn_freq;        /* input/output frequency structure */
   TN_DUR             tn_dur;         /* input/output duration structure  */
   unsigned short int tn_qualid;      /* output qualification ID          */
}  TN_DESC;
```

The TN_DESC structure specifies the characteristics of a tone.

The TN_DESC structure is used in two different parameters of the **lm_LearnTone( )** function, and serves the following purposes:

- **tn_rangep** (input): Specifies optional tone learning boundaries as input to the **lm_LearnTone( )** function, restricting learning to the specified range.
- **tn_tonep** (input): Specifies an existing tone description as optional input to the **lm_LearnTone( )** function, and which learn mode incorporates in the final tone description.
- **tn_tonep** (output): Provides the final tone description that is returned by **lm_LearnTone( )**.

NOTE: Set any field to 0 to use the default value.

Use the **lm_clrdesc( )** function to clear the fields of a TN_DESC structure, including the fields in the TN_FREQ and TN_DUR structures.

Table 3. TN_DESC Structure

Field	Description
tn_freq	Frequency Parameters. Specifies the frequency information in the TN_FREQ data structure. See TN_FREQ data structure.
tn_dur	Cadence Parameters. Specifies the cadence information in the TN_DUR data structure. See TN_DUR data structure.
tn_qualid	Length: 2 (unsigned short int) Default: QT_LMMID
	Output Qualification ID. Returns in **tn_tonep** the qualification ID associated with the tone description. This ID matches the input qualification ID specified in LM_PARM lm_qualid. Before creating the tone for global tone detection or call progress analysis, you must use the **dx_selqual( )** function to select the qualification ID that is returned here. This field is not used for TN_DESC **tn_tonep** input or TN_DESC **tn_rangep** input.
	QT_LMLOW: Low qualification (poor line quality).

Field	Description
	QT_LMMID: Middle qualification (medium line quality). QT_LMHIGH: High qualification (high line quality).

6.5. TN_DUR: Tone Duration

The TN_DUR structure is declared as follows:

```
typedef  struct {
   short int tn_on;            /* Cadence on-time (10msec)    */
   short int tn_ondev;         /* on-time deviation (10msec)  */
   short int tn_off;           /* Cadence off-time (10msec)   */
   short int tn_offdev;        /* off-time deviation (10msec) */
} TN_DUR;
```

The TN_DUR structure specifies cadence information for a tone and is a member of the TN_DESC structure.

The TN_DESC structure is used in two different parameters of the **lm_LearnTone( )** function, and serves the following purposes:

- **tn_rangep** (input): Specifies optional tone learning boundaries as input to the **lm_LearnTone( )** function, restricting learning to the specified range.
- **tn_tonep** (input): Specifies an existing tone description as optional input to the **lm_LearnTone( )** function, and which learn mode incorporates in the final tone description.
- **tn_tonep** (output): Provides the final tone description that is returned by **lm_LearnTone( )**.

Use the **lm_clrdesc( )** function to clear the fields of a TN_DESC structure, including the fields in the TN_FREQ and TN_DUR structures.

Table 4. TN_DUR Structure

Field	Description
tn_on	Length: 2 (short int) Units: 10 ms Range: 0 – 1000

Field	Description
	Tone-On Time. Specifies the tone-on duration in 10 ms units for the tone cadence.
tn_ondev	Length: 2 (short int) Units: 10 ms Range: -1000 – 1000
	Tone-On Time Deviation. Specifies the deviation of the tone-on duration in 10 ms units for the tone cadence. This deviation is subtracted from and added to tn_on to define the minimum and maximum on-time for the cadence.
tn_off	Length: 2 (short int) Units: 10 ms Range: 0 – 1000
	Tone-Off Time. Specifies the tone-off duration in 10 ms units for the tone cadence.
tn_offdev	Length: 2 (short int) Units: 10 ms Range: -1000 – 1000
	Tone-Off Time Deviation. Specifies the deviation of the tone-off duration in 10 ms units for the tone cadence. This deviation is subtracted from and added to tn_off to define the minimum and maximum off-time for the cadence.

6.6. TN_FREQ: Tone Frequency

The TN_FREQ structure is declared as follows:

```
typedef  struct {
   unsigned short int tn_freq1;        /* 1st tone frequency */
   unsigned short int tn_fq1dev;       /* 1st tone deviation */
   unsigned short int tn_freq2;        /* 2nd tone frequency */
   unsigned short int tn_fq2dev;       /* 2nd tone deviation */
} TN_FREQ;
```

The TN_FREQ structure specifies frequency information for a tone and is a member of the TN_DESC structure.

The TN_DESC structure is used in two different parameters of the **lm_LearnTone( )** function, and serves the following purposes:

- **tn_rangep** (input): Specifies optional tone learning boundaries as input to the **lm_LearnTone( )** function, restricting learning to the specified range.
- **tn_tonep** (input): Specifies an existing tone description as optional input to the **lm_LearnTone( )** function, and which learn mode incorporates in the final tone description.
- **tn_tonep** (output): Provides the final tone description that is returned by **lm_LearnTone( )**.

Use the **lm_clrdesc( )** function to clear the fields of a TN_DESC structure, including the fields in the TN_FREQ and TN_DUR structures.

Table 5. TN_FREQ Structure

Field	Description
tn_freq1	Length: 2 (unsigned short int). Units: Hz. Range: 300 Hz – 3000 Hz. Tone 1 Frequency: Specifies the frequency of the first tone in Hz. This information serves different purposes depending upon where the TN_DESC structure is specified: For **tn_tonep** Existing Tone Input: Set this field to a non-zero value to adjust an existing tone description with the **lm_LearnTone( )** function; you must specify in **tn_tonep** the complete tone description to be adjusted (also, make sure to set the **dflagp** location to specify a flag value for a single or dual tone). For **tn_tonep** New Tone Input: Set this field to zero to learn a new tone with the **lm_LearnTone( )** function; you must set the **dflagp** location to specify a flag value for a single or dual tone. For **tn_tonep** Output: When **lm_LearnTone( )** is complete, this field returns frequency information for the final tone description. For **tn_rangep** Input: Set this field to a non-zero value

Field	Description
	to restrict the learning range; you must set lm_cadflag field in LM_PARM structure to indicate cadence or continuous if you do this.
tn_fq1dev	Length: 2 (unsigned short int). Units: Hz.
	Tone 1 Frequency Deviation: Specifies the frequency deviation of the first tone in Hz. This deviation is subtracted from and added to tn_freq1 to define the tone detection range.
tn_freq2	Length: 2 (unsigned short int). Units: Hz. Range: 300 Hz – 3000 Hz.
	Tone 2 Frequency: Specifies the frequency of the second tone in Hz.
	0: Indicates there is no second tone (signal is a single tone).
tn_fq2dev	Length: 2 (unsigned short int). Units: Hz.
	Tone 2 Frequency Deviation: Specifies the frequency deviation of the second tone in Hz. This deviation is subtracted from and added to tn_freq2 to define the tone detection range.

6.7. TN_INFO: Tone Information

The TN_INFO structure is declared as follows:

```
typedef struct {
   unsigned short int  tn_freq1;     /* 1st tone frequency in Hz */
   unsigned short int  tn_freq2;     /* 2nd tone frequency in Hz */
   unsigned short int  tn_on;        /* cadence on-time (10 ms units) */
   unsigned short int  tn_off;       /* cadence off-time (10 ms units) */
   unsigned short int  tn_rep_cnt;   /* actual rep count */
} TN_INFO;
```

The TN_INFO structure description is provided here for reference purposes. This structure is included in the TN_INFOLIST structure.

The tone information structure (TN_INFO) reports the data on which learn mode bases the final tone description. It provides the actual frequency and actual on/off

times for each frame (sample) of a detected tone. This information is used primarily for debugging. If the final tone description does not detect a tone as desired, you can analyze this data to obtain an accurate picture of the tone.

Table 6. TN_INFO Structure

Parameter	Description
tn_freq1	Length: 2 (unsigned short int) Units: Hz
	Frequency 1: Returns the frequency in Hz of the first tone in the tone sample.
tn_freq2	Length: 2 (unsigned short int) Units: Hz
	Frequency 2: Returns the frequency in Hz of the second tone in the tone sample.
tn_on	Length: 2 (unsigned short int) Units: 10 ms
	On-Time: Returns the tone-on time in 10 ms units for the tone sample.
tn_off	Length: 2 (unsigned short int) Units: 10 ms
	Off-Time: Returns the tone-off time in 10 ms units for the tone sample.
tn_rep_cnt	Length: 2 (unsigned short int) Default: 0
	Tone-On/Off Repetition Count: *Not Used for learn mode tone information (always 0).*

6.8. TN_INFOLIST: Tone Information List

The TN_INFOLIST structure is declared as follows:

```
typedef  struct {
   TN_INFO *tn_infop;
   unsigned short size;
}TN_INFOLIST
```

The TN_INFOLIST structure reports the data for each frame (sample) of a detected tone. This data or tone information is used to determine the final tone definition. This structure is useful for debugging purposes.

Table 7. TN_INFOLIST Structure

Field	Description
tn_infop	This field is a pointer to an array of type TN_INFO.
size	This field is the size of the array of TN_INFO passed as tn_infop. You must allocate sufficient memory for the array. Suggested size of array is the number of lm_frames (stored in LM_PARM structure) + OFFSET (defined in *lmodelib.h*) or higher. See TN_INFO data structure description for more information. See LM_PARM data structure description for more information.

Glossary

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.

answer supervision: A telephone system feature that returns a momentary drop in loop current when a connection has been established. When call progress analysis detects a transient loop current drop, it returns a connect event.

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.

automatic gain control: See *AGC*.

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 voice board, or emulated devices.

bps (bits per second): Serial digital stream data rate.

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 that 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 or I/O function active on it.

cadence: A pattern of tones and silence intervals generated by a given audio signal. 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.

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

call progress analysis: The process used to automatically determine what happens after an outgoing call is dialed.

CCITT: International Telephone and Telegraph Consultative Committee, a part of the ICU (International Telecommunications Union) responsible for formulating telecommunications standards.

channel: **1.** When used in reference to an Intel 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 an Intel 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*.

CO: Central Office. The telephone company facility where subscriber lines are linked, through switches, to other subscriber lines (including local and long distance lines).

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

data structure: C programming term for a data element consisting of fields, where each field may have a different type definition and length. The elements of a data structure usually share a common purpose or functionality, rather than being similar in size, type, etc.

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

device channel: An Intel Dialogic voice data path that processes one incoming or outgoing call at a time (equivalent to the terminal equipment terminating a phone line).

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 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 tone set: The Intel Dialogic tone sets supplied with the Learn Mode API. They contain tone definitions for most PBXs and networks. There is no consolidation data.

digital: Information represented as binary code.

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

downloaded code: Program instructions and routines that 1. run at the board level and were previously resident on the board in EPROM and 2. are now loaded during board initialization to a reserved section of shared RAM.

driver: A software module that provides a defined interface between a program and the hardware.

DSP: **1.** Digital signal processor. A microprocessor with an architecture that is optimized particularly to perform mathematical algorithms that manipulate digital signals. **2.** Digital signal processing.

DTMF: Dual Tone Multi Frequency. Refers to a method of encoding digits over analog telephone lines.

dynamic link library (DLL): A file in the Intel Dialogic system release that contains the Intel Dialogic library functions. Compare *Library*.

emulated device: A virtual device whose software interface mimics the interface of a particular physical device, such as a D/4x voice board that is emulated by a D/240SC voice board. On a functional level, a D/240SC voice board is perceived by an application as six emulated D/4x voice boards. See *physical device*.

EPROM: Electrically Programmable Read Only Memory.

event: An unsolicited or asynchronous communication 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.

firmware: A set of program instructions that are resident (usually in EPROM) on an expansion board.

frequency detection: A voice driver feature that detects the tri-tone Special Information Tone (SIT) sequences and other single-frequency tones from 300Hz to 2100Hz.

global tone detection: A feature that allows the creation and detection of user-defined tone descriptions on a channel by channel basis.

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 that controls the on-hook and off-hook state of the Voice telephone interface.

I/O: Input/Output

idle channel: A channel that has no functions active on it.

idle device: A device that has no functions active on it. See *busy device*.

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: See *interrupt request level*.

library: A file in the Intel Dialogic system release that contains links to the *DLL*.

loop: The physical circuit between the telephone switch and the voice processing 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 a 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. .

MF: Multifrequency. An in-band signaling transmission scheme similar to *DTMF* but used mainly within the Central Office (see *CO*).

micro channel bus: The communication channel or architecture in a computer used to interface the host processor to add-on adapter boards.

off-hook: The state of a telephone station when the conductor loop between the station and the switch is closed and current is flowing.

physical device: A device that is an actual piece of hardware, such as a D/xx voice board; not an emulated device. See *emulated device*.

pointer: A memory address to either a function or data.

polling: The process of repeatedly checking the status of a resource to determine when state changes occur.

resolution: Granularity of measurement.

resource: Functionality (e.g., voice-store-and-forward) that can be assigned to call. Resources are *shared* when functionality is selectively assigned to a call (usually via an SCbus or CT Bus time slot) and may be shared among multiple calls. Resources are *dedicated* when functionality is fixed to the one call.

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 device.

route: Assign a resource to a time slot.

sampling rate: Frequency with which a digitizer takes measurements of the analog voice signal.

shared resource: see *Resource*.

signaling: The transmission of electrical signals on the telephone network. The voice software supports the following signaling methods: DTMF, MF, R2 MF, Socotel, global tone detection and generation, and dial pulse detection and generation.

silence threshold: The level that sets whether incoming data to a voice board with call-progress analysis is recognized as silence or non-silence.

SIT: Special Information Tone. Detection of a SIT sequence indicates an operator intercept or other problem in completing the call. See also *Frequency Detection*.

string: A data element consisting of a collection of contiguous 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).

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.

Index

S

T