Learn Mode and Tone Set File API Software Reference

for Linux and Windows Operating Systems

05-2069-001

Publication Date: November, 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/design/network/products/telecom/

For **Sales Offices** and other contact information, visit the Where to Buy Intel Telecom Products page:
http://www.intel.com/buy/wtb/wtb1028.htm

Table of Contents

List of Tables

Revision History

This revision history summarizes the changes made in each published version of this document.

Document No.	Publication Date	Description of Revisions
05-2069-001	November 2003	Initial version of document.

About this Publication

Purpose

This document provides a complete reference to the learn mode and tone set file (TSF) API library functions and data structures. It also discusses programming guidelines for these APIs. The learn mode and TSF APIs are supported on Springware and DM3 boards for use in the Windows and Linux operating systems.

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

The information in this guide is organized as follows:

Chapter 1 provides an overview of the learn mode and tone set file software.

Chapter 2 summarizes the learn mode and tone set file API library functions.

Chapter 3 provides programming guidelines for developing an application to learn tones and create tone set files.

Chapter 4 discusses compiling and linking requirements such as include files and library files.

Chapter 5 provides an alphabetical reference to the learn mode and tone set file API library functions.

Chapter 6 describes error codes that may be returned by the API library functions.

Chapter 7 provides an alphabetical reference to the data structures used by the API 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 API Library Reference* – provides information on all voice API library functions, parameters, and data structures.
- *Voice API Programming Guide* – describes voice API library features and programming guidelines.
- *Standard Runtime Library API Library Reference* – provides information on all Standard Runtime Library functions, parameters, and data structures.
- *Standard Runtime Library API Programming Guide* – describes Standard Runtime Library features and programming guidelines. Also provides information on programming models.
- *Release Guide* for your system release – provides information about the system release, system requirements, software and hardware features, supported hardware, and release documentation.
- *Release Update* for your system release – (available on the Technical Support Web site only) describes compatibility issues, restrictions and limitations, known problems, and late-breaking updates or corrections to the release

documentation. The *Release Update* is updated with new information as needed during the lifecycle of the release.

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 and Tone Set File API Libraries

Integrating an interactive voice response (IVR), voicemail, or auto attendant system with one or more PBX or key system switches presents a unique challenge. The challenge occurs because each switch does not always produce standard telephone company call progress tones such as busy, dial tone, or ringback. In such cases, your computer telephony system may be unable to recognize the state of a call and unable to perform vital functions such as transferring or disconnecting a call.

To remedy this situation, the standard call progress tone detection algorithms within the telecom board must know these custom frequencies and cadences. The learn mode API and tone set file API are designed to simplify the PBX integration process. These APIs make it quicker and easier to integrate Intel telecom boards with virtually any telephone switch or Public Switched Telephone Network (PSTN). The learning process eliminates the need to manually define and test the frequency and cadence characteristics of call progress tones.

The PBX Expert utility is available for learning tones. This utility (previously called PBXpert/32) uses the learn mode API and the tone set file API to accomplish PBX tone learning and tone set file management. The learn mode and tone set file API libraries give you the flexibility to integrate tone learning capability directly in your computer telephony applications. The PBX Expert utility is provided with the system release.

By saving a set of tone characteristics (dial tone, busy, ringback, reorder, and disconnect) for each unique switch, the tone set file API enables the Intel board to detect their call progress tones, allowing the board to recognize the state of a call and perform appropriate functions. Additionally, you can easily edit and implement tone characteristics without having to spend time recompiling your application-level software.

The TSF API also lets you combine up to 10 sets of tone characteristics into a single, consolidated tone set. That means the Intel board can recognize call progress tones from up to 10 unique sources using this consolidated tone set file.

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 API Programming Guide*.

The learn mode and tone set file API libraries supply functions that interface with the voice driver. These libraries are supported on specific boards and run on Linux and Windows operating systems. See the release guide accompanying the system release for information on hardware support.

1.2. Key Features of Learn Mode and Tone Set File API Libraries

Key features of the learn mode and tone set file API libraries include the following:

- ability to characterize a call progress tone from a PBX, key system or PSTN and store that information
- ability to change default tone definitions that are provided by the voice library
- ability to create a new tone definition and add it to the tone template for use with call progress analysis
- ability to store an unlimited number of tone sets on your system (subject to storage constraints)
- ability to combine up to 10 sets of tone characteristics into a single, consolidated tone set
- ability to support up to 10 sets of tone characteristics (that is, for up to 10 different PBX or key systems).

2. Function Summary by Category

2.1. Overview

This chapter describes the categories in which the learn mode and tone set file API functions can be grouped.

2.2. 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

2.3. Tone Set File Functions

Use tone set file functions to manage tone set data and tone set files.

The tone set file functions can be categorized as follows:

- I/O functions
- tone set handling functions
- tone set consolidation functions

2.3.1. I/O Functions

The following functions are used in file handling:

Function Name	Definition
tsf_ActivateFile( )	• activates the tone set file
tsf_CloseFile( )	• closes an open tone set file
tsf_GetFileInformation( )	• retrieves updated information about a tone set file after **tsf_OpenFile( )** is called
tsf_OpenFile( )	• opens the tone set file specified
tsf_SaveFile( )	• saves tone set data to a tone set file

2.3.2. Tone Set Handling Functions

The following functions are used in handling tone sets:

Function Name	Definition
tsf_AddToneSet( )	• adds a tone set entry to the tone set file
tsf_DeleteToneSet( )	• deletes a tone set entry from the tone set file
tsf_DuplicateToneSet( )	• duplicates a tone set in the tone set file
tsf_GetNumberOfToneSets( )	• convenience function that returns the number of tone sets in the open tone set file
tsf_GetToneSet( )	• returns tone set information
tsf_GetToneSetKeys( )	• convenience function that returns the keys (identifiers) of the tone sets

Function Name	Definition
	in the tone set file
tsf_GetToneSetName( )	• returns the tone set ID or tone set name (typically the PBX or key system manufacturer name and model number)
tsf_ModifyToneSet( )	• changes the parameters of a tone set

2.3.3. Tone Set Consolidation Functions

The following functions are used in consolidating tone sets:

Function Name	Definition
tsf_ClearConsolidationOptions( )	• clears the consolidation options structure
tsf_ClearDefaultToneSetToConsolidate( )	• clears the default tone information for consolidation and sets all values to 0
tsf_ConsolidateToneSets( )	• consolidates the tone sets marked for consolidation
tsf_GetConsolidatedToneSet( )	• returns tone set information of the consolidated tone set
tsf_GetConsolidatedToneSetKeys( )	• returns keys of tone sets consolidated in an array
tsf_GetDefaultToneSetInConsolidation( )	• returns default tone set values included in consolidation
tsf_GetNumberOfToneSetsConsolidated( )	• returns number of tone sets included in consolidation
tsf_SetDefaultToneSetToConsolidate( )	• sets values of the default tone set to be included in the consolidation process

3. Application Development Guidelines

3.1. Mode of Operation

The learn mode and TSF API 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.

3.2. 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 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.
- 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.

3.3. 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 *section 7.2. 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.*

3.4. Understanding Tone Sets and Tone Set Files

A tone set file (TSF) is a binary file consisting of the following parts:

- tone sets. Each tone set contains tone information for up to 10 different call progress tones. These tones include three dial tones (local dial tone, international dial tone, extra dial tone), two ringback tones, two busy tones, a disconnect tone, and two fax tones. Each tone set is typically associated with a specific PBX or key system.
- a consolidated tone set within the tone set file. Up to 10 tone sets can be selected for consolidation. Note that your tone set file can contain more than 10 tone sets; however, only 10 tone sets can be selected for consolidation at a time. Consolidation lets you combine several PBX tone set definitions and

compute a single tone set, which can detect the tones of all PBXs associated with the consolidation.

- a binary flag that indicates whether the tone set file contains a valid consolidated tone set.

You can have only one active tone set file per system and one consolidated tone set per tone set file.

3.5. Creating Tone Sets and Tone Set Files

The learn mode API and tone set file API are designed to make it easier to manage the unique call progress tones produced by PBXs, key systems, and PSTNs. The tone sets and tone set file contain information on these unique call progress tones. These new call progress tone definitions can in turn be used by the call progress analysis feature in your computer telephony application.

Call progress analysis is a feature of the voice API library that uses a combination of frequency and cadence detection to identify call progress tones to determine the status of a call. For detailed information on call progress analysis, see the *Voice API Programming Guide*.

Applications can learn tone characteristics using the functions from the learn mode API library. Information on several different tones forms one tone set. Tone sets can be written to a TSF using the tone set file API. It is an application's responsibility to consolidate different tone sets and save the consolidated tone sets using the TSF API functions. To activate the consolidated tone sets on a board, use voice API library functions. The process for this last step differs depending on the board type (DM3 or Springware) and the operating system (Linux or Windows.)

For example, let's say you already have a TSF containing five PBX tone sets, and you want to add another tone set for a new switch, for which there is no tone set definition. Use the following guidelines to manage unique call progress tones produced by PBXs, key systems, and PSTNs:

NOTE: Not all function calls and data structures required for your application are mentioned here. These guidelines help provide the general flow of the application.

1. Open the tone set file (TSF) containing existing tone sets using **tsf_OpenFile( )**.

2. Learn the new tone characteristics using the learn mode API.
3. Add the learned tones to your TSF using **tsf_AddToneSet( )** (creating a new tone set).
4. Add additional tone sets as required using **tsf_AddToneSet( )**.
5. Perform file operations on your TSF as necessary, such as modifying, deleting, and duplicating a tone set.
6. Decide which tone sets to include in the consolidated tone set.
7. As a precaution, before consolidating a tone set, clear the old default tone set using **tsf_ClearDefaultToneSetToConsolidate( )**.
8. (Optional) Set the new default tone set using **tsf_SetDefaultToneSetToConsolidate( )**.
9. As a precaution, clear the consolidation options structure using **tsf_ClearConsolidationOptions( )**.
10. Consolidate specified tone sets in the TSF using **tsf_ConsolidateToneSets( )**. Even if only one tone set will be activated, you must call this function. The consolidated tone set can be used to replace the voice API library's default tone set definitions.
11. Save the tone set data using **tsf_SaveFile( )**.
12. To activate the new tone set file for use with call progress analysis, call **tsf_ActivateFile( )**.
13. Close the tone set file using **tsf_CloseFile( )**.
14. On DM3 boards, choose one of the following ways to enable and load the new tones on the board:
    - Stop the board and restart it from configuration manager (DCM).

      OR

    - Retrieve the new tone information from the tone set file by calling **tsf_GetConsolidatedToneSet( )**. Then use the information for each tone to change the default tone definition on the board by calling **dx_querytone( )**, **dx_deletetone( )**, and **dx_createtone( ).**
15. On Springware boards, close the channels and re-open them using **dx_open( )**. Opening a channel device with **dx_open( )** after activating a tone set file (via **tsf_ActivateFile( )**) will enable the use of the consolidated tone

set (and its tone definitions) from the tone set file. Then use **dx_initcallp( )** to initialize enhanced call progress analysis and send the new tone definitions to the board.

On Springware boards running Linux, this function reads the consolidated tone set from the tone set file according to settings in the *pbxsetting.txt* file. For more information, see *3.6. TSFs and pbxsetting.txt* on page 12.

On Springware boards running Windows, this function reads the consolidated tone set from the tone set file according to settings in the configuration manager (DCM). The **tsf_ActivateFile( )** function updates TSFFileSupport and TSFFileName fields in DCM.

16. To perform call progress analysis on Springware boards, use the Voice API library. For more information on the dx_ voice API functions, see the *Voice API Library Reference*. On DM3 boards, use the Global Call API library. For more information, see the *Global Call API Library Reference* and the Technology User's Guide for the protocol you are using.

3.6. TSFs and pbxsetting.txt

The *pbxsetting.txt* is used on the Linux operating system only. The *pbxsetting.txt* is located in the directory */usr/dialogic/cfg*.

The purpose of the *pbxsetting.txt* file is as follows:

1. To indicate whether or not a TSF is supported (DLGCTSFSUPPORT).
2. If TSF is supported, to provide the location and name of the TSF (DLGCTSFFILEPATH).
3. To indicate whether or not disconnect supervision is enabled (DLGCDISCONNECTTONE).

There should only be three lines in the *pbxsetting.txt* file. The values DLGCTSFSUPPORT, DLGCTSFFILEPATH, and DLGCDISCONNECTTONE comprise the file. These three values should be the first three lines of the file, as all other lines after the first three lines are ignored. No commenting is supported in the first three lines.

3.6.1. Rules for Using pbxsetting.txt

If DLGCTSFSUPPORT = YES, then a TSF is used to configure call progress analysis (CPA) on the system. If TSF is enabled, then DLGCTSFFILEPATH must contain the path and name of the TSF.

If DLGCTSFSUPPORT = NO, then TSF is not used to configure call progress analysis, and the value of DLGCTSFFILEPATH is ignored.

If DLGCDISCONNECTTONE = YES, then disconnect tone supervision is enabled.

If DLGCDISCONNECTTONE = NO, then disconnect tone supervision is disabled.

3.6.2. Examples of pbxsetting.txt

This section provides examples of settings in pbxsetting.txt.

Example 1

```
DLGCTSFSUPPORT = YES
DLGCTSFFILEPATH = /usr/dialogic/cfg/sample.tsf
DLGCDISCONNECTTONE  = YES
```

The above *pbxsetting.txt* file example configures a system to use the TSF *sample.tsf* and enables disconnect tone supervision.

Example 2

```
DLGCTSFSUPPORT = NO
DLGCTSFFILEPATH = /usr/dialogic/cfg/sample.tsf
DLGCDISCONNECTTONE  = YES
```

The above *pbxsetting.txt* file example configures a system to not enable any TSF and enables disconnect tone supervision with the default disconnect tone definitions. The value of DLGCTSFFILEPATH is ignored. All the default tone information is used for CPA.

Example 3

```
DLGCTSFSUPPORT = YES
```

```
DLGCTSFFILEPATH = /usr/dialogic/cfg/sample.tsf
DLGCDISCONNECTTONE  = NO
```

The above *pbxsetting.txt* file example configures a system to use the TSF *sample.tsf* and disables disconnect tone supervision.

Example 4

```
DLGCTSFSUPPORT = NO
DLGCTSFFILEPATH = /usr/dialogic/cfg/sample.tsf
DLGCDISCONNECTTONE  = No
```

The above *pbxsetting.txt* file example configures a system to not use any TSF and disables disconnect tone supervision. Therefore, all default tone information goes into effect, except the disconnect tone.

4. Building Applications

4.1. 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_).
- *tsfiolib.h* – tone set file library header file. Include this header file in applications that use tone set file library functions (typically, those that begin with tsf_).

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

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

4.2. 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.

In Linux, you must link the following library files when compiling your application:

- *libtsfio.so* – tone set file library file. Specify **-ltsfio** in makefile.
- *liblmode.so* – learn mode library file. Specify **-llmode** in makefile.
- *libdxxx.so* – main voice library file . Specify **-ldxxx** in makefile.
- *libsrl.so* – Standard Runtime Library file. Specify **-lsrl** in makefile.

In Windows, you must link the following library files when compiling your application:

- *libtsfio.lib* – tone set file library file
- *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 in the directory given by the INTEL_DIALOGIC_LIB environment variable.

5. 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 (when applicable)
- Errors (when applicable)
- 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:	DM3, 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 6.1. Learn Mode Error Codes*.

■ Example

For an example, see the **lm_learnTone( )** example code.

■ See Also

- **lm_LearnTone( )**

clears the tone amplitude structure

- **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:	DM3, 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 6.1. Learn Mode Error Codes*.

■ Example

For an example, see the **lm_learnTone( )** example code.

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:	DM3, 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 6.1. Learn Mode Error Codes*.

■ Example

For an example, see the **lm_learnTone( )** example code.

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:	DM3, 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 API Programming Guide* and *Voice API Library Reference*.

Parameter	Description
cd	Specifies the handle for the voice channel device. **NOTE:** If using the Global Call API for call control, you must retrieve the voice handle of the line device and

Parameter	Description
	use that handle here.
lm_parmp	Points to the LM_PARM data structure which specifies parameters used to characterize the tone. For more information, see the LM_PARM structure in *Chapter 7. 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 7. 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 7. 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

Parameter	Description
	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 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 API Programming 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 API Programming 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

This function returns 0 or a non-negative number to indicate success. This function returns -1 to indicate failure. For a list of error codes, see *Section 6.1. Learn Mode Error Codes*.

If a library function fails, call the standard attribute function **ATDV_LASTERR( )** to obtain the error code and **ATDV_ERRMSGP( )** to obtain a descriptive error message. For more information on these functions, see the *Standard Runtime Library API Library Reference* .

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.

■ 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 rc;
   int Caller;

   /* Prepare the scenario for learning dial tone */

   Caller = dx_open(DEVICE,(int)NULL);
   if (Caller == ERR)
   {
      /* Error occurred in opening DEVICE */
      /* Handle the error here and return */
      /* error                            */
      return(ERR);
   }

   /* Set the device offhook to generate a dial tone */
   rc = dx_sethook(Caller, DX_OFFHOOK, EV_SYNC);
   if (rc == ERR)
   {
      /* Error setting off hook */
      return(ERR);
   }
```

```
   printf("Beginning Learning.............\n");

   /* Learn the dial tone */
   rc = lrn_Learn(Caller);
   if (rc != ERR)
   {
      printf("Finished Learning.............\n");
   }
   else
   {
      printf("Error Learning...\n");
   }

   /* reset the channel here */

   return 0;
}

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

   /* Empty the data structures */
   lm_clrparm(&parm);
   lm_clramp(&amp);
   lm_clrdesc(&desc);

   flagDual = 0; /* Unknown whether Dual/Single tone */

   /* 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);
   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( )**

Name:	TSF_RET tsf_ActivateFile(char *filename)
Inputs:	filename • tone set file name
Outputs:	none
Returns:	error code
Includes:	tsfiolib.h
Category:	TSF
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **tsf_ActivateFile( )** function activates the tone set file.

In Linux, this function updates the *pbxsetting.txt* which contains information about the tone set file. For more information on this file, see *section 3.6. TSFs and pbxsetting.txt*.

In Windows, this function updates the TSFFileSupport field (Misc tab) and the TSFFileName field (File tab) in the configuration manager (DCM) at runtime. You can also set these fields manually using DCM before downloading the board.

Parameter	Description
filename	specifies the tone set file name **NOTE:** If NULL is passed, the currently opened tone set file is activated.

■ Cautions

None.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"


int main() {
int ret;
  TSF_FILE_INFO file_info;
  TSF_CONSOLIDATIONOPTIONS conscfg;
  TONESETINFO default_info;

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */

  /* Consolidate the tone set file */
  /* Clear the TSF_CONSOLIDATIONOPTIONS structure first before tsf_ConsolidateToneSets()
*/
  tsf_ClearConsolidationOptions (&conscfg);

  conscfg.dial_tone_id = TID_DIAL_LCL;
  conscfg.exclude_default_defs = FALSE;                // so include default tone
information in consolidation process

/*clear out default tone information from consolidation*/
tsf_ClearDefaultToneSetToConsolidate();

/* Set default tone information in default_info structure variable.
*
*
*
*/
/*set defaults */
tsf_SetDefaultToneSetToConsolidate ( &default_info);

  /* Consolidate tone set file */
  ret = tsf_ConsolidateToneSets(&conscfg);
  if (ret == TSF_SUCCESS) {
    printf("OK: tsf_ConsolidateToneSets ()\n");
  }
  else {
    printf("Fail: tsf_ConsolidateToneSets () ret=%d\n", ret);
  }

  /* Right after the tsf_ConsolidateToneSets (), Save the file  */
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

/* Activate tone set file */
```

```
  ret = tsf_ActivateFile(NULL);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_ActivateFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_ConsolidateToneSet( )**

Name:	TSF_RET tsf_AddToneSet (TSF_KEY *pkey, TONESET_ID *ptoneset_name, TONESETINFO *ptonesetinfo)	
Inputs:	ptonesetinfo	• pointer to TONESETINFO
	ptoneset_name	• pointer to tone set ID information
Outputs:	pkey	• key associated with the newly added tone set
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_AddToneSet( )** function adds a tone set entry to the tone set file. The new tone set entry contains information about the tone set in **ptoneset_name** and **ptonesetinfo**.

Parameter	Description
pkey	specifies a pointer to a local variable of type TSF_KEY. The function returns the key associated with the newly added tone set in the local variable. This key is generated by the TSF library and is a tone set identifier. This key is used for subsequent activities on a tone set such as deleting, modifying and so on.
ptoneset_name	specifies a pointer to the tone set name ID information. The variable holds the tone set file entry that contains the tone set name associated with the key **tsf_key**. The name usually consists of the manufacturer of the PBX or key system and its model number.

Parameter	Description
ptonesetinfo	specifies a pointer to the structure variable of type TONESETINFO. This structure contains tone information for all MAX_TONES tones in the tone set. This tone set is added to the tone set file. MAX_TONES is a define that specifies the maximum number of tones in a tone set. For more information on the structure, see *section 7.11. TONESETINFO*.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before adding a tone set entry.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TONESETINFO tonesetinfo;
  TONESET_ID  toneset_id;
  TSF_KEY   return_key;

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }
```

```
  /* fill in data structure toneset_id and tone_info with desired value before
     tsf_AddToneset()
  */
  ret = tsf_AddToneSet(&return_key, &toneset_id, &tonesetinfo);

  if (ret == TSF_SUCCESS) {
     printf("OK: tsf_AddToneSet() key=%d\n", return_key);
  }
  else {
    printf("Fail: tsf_AddToneSet() ret=%d\n", ret);
  }

  /* Continue processing such as adding more Toneset and consolidation
   *.
   *.
   *.
   */

  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_DeleteToneSet( )**
- **tsf_ModifyToneSet( )**
- **tsf_DuplicateToneSet( )**

Name:	void tsf_ClearConsolidationOptions (TSF_CONSOLIDATIONOPTIONS *consolidateops)	
Inputs:	consolidateops	• pointer to a TSF_CONSOLIDATIONOPTIONS structure to be cleared.
Outputs:	Clearing of consolidateops	
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	

■ Description

The **tsf_ClearConsolidationOptions( )** function clears consolidation structure. This function clears the TSF_CONSOLIDATIONOPTIONS structure and resets all consolidation options to the default. This is a convenience function.

It is good practice to call this function before consolidating tone sets using **tsf_ConsolidateToneSets( )**.

Parameter	Description
consolidateops	specifies a pointer to a variable of type TSF_CONSOLIDATIONOPTIONS which configures options for the consolidation process. For more information on this structure, see *section 7.12. TSF_CONSOLIDATIONOPTIONS*.

■ Cautions

None.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_CONSOLIDATIONOPTIONS  config;

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding, deleting entry and mark tone sets for
consolidation
   *.
   *.
   *.
   */
  /* Consolidate the Tone set file */
  /* Clear the TSF_CONSOLIDATIONOPTIONS structure first before tsf_ConsolidateToneSets()
*/
  tsf_ClearConsolidationOptions (&config);

  config.dial_tone_id = TID_DIAL_LCL;
  config.exclude_default_defs = TRUE;

  /* Consolidate Tone set file */
  ret = tsf_ConsolidateToneSets(&config);
  if (ret == TSF_SUCCESS) {
    printf("OK: tsf_ConsolidateToneSets ()\n");
  }
  else {
    printf("Fail: tsf_ConsolidateToneSets () ret=%d\n", ret);
  }

  /* Right after the tsf_ConsolidateToneSets (), Save the file  */
  ret = tsf_SaveFile(NULL, 1);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }
  /* return from main() */
  return 0
}
```

See Also

- **tsf_ConsolidateToneSets( )**

Name: void tsf_ClearDefaultToneSetToConsolidate (void)
Inputs: None
Outputs: None
Returns: error code
Includes: tsfiolib.h
Category: TSF
Mode: synchronous
Platform: DM3, Springware

■ Description

The **tsf_ClearDefaultToneSetToConsolidate( )** function clears the default tone information for consolidation.

It is good practice to call this function before consolidating tone sets using **tsf_ConsolidateToneSets( )**.

■ Cautions

This function will not have any effect after the consolidation process is performed; that is, after calling **tsf_ConsolidateToneSets( )**.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_CONSOLIDATIONOPTIONS conscfg;
  TONESETINFO default_info;

  /*  tsf_OpenFile()  */
```

```
  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */

  /* Consolidate the Tone set file */
  /* Clear the TSF_CONSOLIDATIONOPTIONS structure first before tsf_ConsolidateToneSets()
*/
  tsf_ClearConsolidationOptions (&conscfg);

  conscfg.dial_tone_id = TID_DIAL_LCL;
  conscfg.exclude_default_defs = FALSE;                // so include default tone
information in consolidation process

/*clear out default tone information from consolidation*/
tsf_ClearDefaultToneSetToConsolidate();

/* Set default tone information in default_info structure variable.
*
*
*
*/
/*set defaults */
tsf_SetDefaultToneSetToConsolidate ( &default_info);

  /* Consolidate Tone sets */
  ret = tsf_ConsolidateToneSets(&conscfg);
  if (ret == TSF_SUCCESS) {
    printf("OK: tsf_ConsolidateToneSets ()\n");
  }
  else {
    printf("Fail: tsf_ConsolidateToneSets () ret=%d\n", ret);
  }

  /* Right after the tsf_ConsolidateToneSets (), Save the file  */
  ret = tsf_SaveFile(NULL, 1);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

■ See Also

- **tsf_GetDefaultToneSetInConsolidation( )**
- **tsf_SetDefaultToneSetToConsolidate( )**

- **tsf_ConsolidateToneSets( )**

Name:	TSF_RET tsf_CloseFile (void)
Inputs:	none
Outputs:	none
Returns:	error code
Includes:	tsfiolib.h
Category:	TSF
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **tsf_CloseFile( )** function closes an open tone set file.

■ Cautions

The **tsf_CloseFile( )** function does not save changes to the tone set file. To save updates, call **tsf_SaveFile( )** after opening a file and before closing a file.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }
```

```
  /* Continue processing such as adding, deleting Tone Set entry
   *.OR process consolidation
   *.
   *.
   */

  /* Don't forget to save the file to the name as specified to tsf_OpenFile() */
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_OpenFile( )**
- **tsf_SaveFile( )**

Name:	TSF_RET tsf_ConsolidateToneSets(TSF_CONSOLIDATIONOPTIONS *consolidateops, TSF_KEY *keylist, unsigned int listsize)	
Inputs:	consolidateops	• pointer to a TSF_CONSOLIDATIONOPTIONS structure specifying some parameters which govern the consolidation procedure
	keylist	• array of keys of tone sets to consolidate
	listsize	• number of elements in array keylist
Outputs:	none	
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_ConsolidateToneSets( )** function consolidates tone sets after reading them from the tone set file and writes the results to the tone set file.

Parameter	Description
consolidateops	specifies a pointer to a TSF_CONSOLIDATIONOPTIONS structure specifying parameters which govern the consolidation process. For more information on this structure, see *section 7.12. TSF_CONSOLIDATIONOPTIONS*.
keylist	specifies an array of keys of tone sets to consolidate. A key is generated for every new tone set added to the tone set file using **tsf_AddToneSet( )**. This key is a tone set identifier.
listsize	specifies the number of elementsin the array keylist

■ Cautions

- You must create or open a tone set file using **tsf_OpenFile( )** before consolidating tone sets.
- You must call **tsf_SetDefaultToneSetToConsolidate( )** before calling **tsf_ConsolidateToneSets( )**. The **tsf_SetDefaultToneSetToConsolidate( )** function will not have any effect after the consolidation process is performed.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_CONSOLIDATIONOPTIONS conscfg;
  TSF_KEY keylist[2];
  unsigned short listsize;

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */

  /* Consolidate the tone set file */
  /* Clear the TSF_CONSOLIDATIONOPTIONS structure first before tsf_ConsolidateToneSets()
*/
  tsf_ClearConsolidationOptions (&conscfg);

  conscfg.dial_tone_id = TID_DIAL_LCL;
  conscfg.exclude_default_defs = TRUE;

  /* include tone sets with key 4 and 7 in consolidation */
  keylist[0] = 4;
  keylist[1] = 7;
```

```
  listsize = 2;

  /* Consolidate tone set file */
  ret = tsf_ ConsolidateToneSets(&conscfg,keylist,listsize);
  if (ret == TSF_SUCCESS) {
    printf("OK: tsf_ConsolidateToneSets ()\n");
  }
  else {
    printf("Fail: tsf_ConsolidateToneSets () ret=%d\n", ret);
  }

  /* Right after the tsf_ConsolidateToneSets (), Save the file  */
  ret = tsf_SaveFile(NULL, 1);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_AddToneSet( )**
- **tsf_OpenFile( )**

Name: TSF_RET tsf_DeleteToneSet (TSF_KEY tsf_key)
Inputs: tsf_key
- key associated with the tone set to be deleted

Outputs: none
Returns: error code
Includes: tsfiolib.h
Category: TSF
Mode: synchronous
Platform: DM3, Springware

■ Description

The **tsf_DeleteToneSet( )** function deletes a tone set entry from the tone set file. The tone set to be deleted is referenced by **tsf_key.**

Parameter	Description
tsf_key	specifies the key associated with the tone set to be deleted

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before deleting a tone set entry.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_KEY key_delete;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
```

```
    exit(1);
  }


  /* Continue processing such as adding entry
   *.
   *.
   *.
   */
  ret = tsf_DeleteToneSet(key_delete);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_DeleteToneSet ()\n");
  }
  else
  {
    printf("tsf_DeleteToneSet successfully deleted toneset with key value %d\n",
key_delete );
  }

  /* Don't forget to save the file to the name as specified in tsf_OpenFile() */
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }


  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
return 0;
}
```

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ See Also

- **tsf_AddToneSet( )**
- **tsf_ModifyToneSet( )**
- **tsf_DuplicateToneSet( )**

Name:	TSF_RET tsf_DuplicateToneSet (TSF_KEY tsf_key, TONESET_ID *ptoneset_name, TSF_KEY *pReturn_key)	
Inputs:	tsf_key	• Key indicating which tone set to duplicate
	ptoneset_name	• Tone set ID information for the newly created duplicate tone set entry
Outputs:	pReturn_key	• Key associated with the newly created duplicate tone set
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_DuplicateToneSet( )** function duplicates a tone set entry with the information specified in **tsf_key**. This duplicated tone set is written to the tone set file and can be referenced by the key indicated in **pReturn_key**.

Parameter	Description
tsf_key	specifies the key indicating which tone set to duplicate
ptoneset_name	specifies a pointer to a structure variable of type TONESET_ID. This structure is filled with new ID information.
pReturn_key	specifies the key associated with the newly created duplicate tone set. This is a pointer to a variable of type TSF_KEY. This key identifies the new tone set.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before duplicating a tone set entry.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_KEY tsf_key, new_tsf_key;
  TONESET_ID new_toneset_id;

/*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }


  /* Continue processing such as adding, deleting Tone Set entry
   *.
   *.
   */
/* Fill new_toneset_id structure for new toneset. tsf_key is the key of the tone set to
be copied*/
  ret = tsf_DuplicateToneSet (tsf_key, &new_toneset_id, &new_tsf_key );
  if (ret == TSF_SUCCESS) {
     printf("OK: tsf_ DuplicateToneSet() new key = %d\n", new_tsf_key);
  }
  else  {
     printf("Fail: tsf_ DuplicateToneSet()\n");
  }

  /* Don't forget to save the file to the name as specified to tsf_OpenFile() */
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_AddToneSet( )**

Name:	TSF_RET tsf_GetConsolidatedToneSet(TONESETINFO *ptonesetinfo)	
Inputs:		
Outputs:	ptonesetinfo	pointer to structure variable of type TONESETINFO
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetConsolidatedToneSet( )** function returns consolidated tone information for all MAX_TONES tones from the opened tone set file. This function returns the result of the consolidation process; that is, the consolidated tone set.

Parameter	Description
ptonesetinfo	specifies a pointer to the structure variable of type TONESETINFO. This structure contains tone information for all MAX_TONES tones in the tone set. This tone set is added to the tone set file. MAX_TONES is a define that specifies the maximum number of tones in a tone set. For more information on the structure, see *section 7.11. TONESETINFO*.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before performing other actions on this file.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_CONSOLIDATIONOPTIONS conscfg;
  TSF_KEY keylist[2];
  unsigned short listsize;
  TONESETINFO tonesetinfo;
  TN_TMPLT  tonetmplt[MAX_TONES];

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */

  /* Consolidate the tone set file */
  /* Clear the TSF_CONSOLIDATIONOPTIONS structure first before tsf_ConsolidateToneSets()
*/
  tsf_ClearConsolidationOptions (&conscfg);

  conscfg.dial_tone_id = TID_DIAL_LCL;
  conscfg.exclude_default_defs = TRUE;

  /* include tone sets with key 4 and 7 in consolidation */
  keylist[0] = 4;
  keylist[1] = 7;
  listsize = 2;

  /* Consolidate tone set file */
  ret = tsf_ConsolidateToneSets(&conscfg, keylist,listsize);
  if (ret == TSF_SUCCESS) {
    printf("OK: tsf_ConsolidateToneSets ()\n");
  }
  else {
    printf("Fail: tsf_ConsolidateToneSets () ret=%d\n", ret);
  }

  /* Right after the tsf_ConsolidateToneSets (), Save the file  */
```

```
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

tonetmplt
ret = tsf_GetConsolidatedToneSet(&tonesetinfo);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_GetConsolidatedToneSet()\n");
  }
else
{
/*process the tone template information for to be downloaded.
*
*
*/
}


  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_OpenFile( )**
- **tsf_ClearConsolidationOptions( )**
- **tsf_ConsolidateToneSets( )**

Name:	TSF_RET tsf_GetConsolidatedToneSetKeys (TSF_KEY *keylist, unsigned short *sizeoflist)	
Inputs:	sizeoflist	• pointer to unsigned short that contains number of elementsin array keylist
Outputs:	keylist	• array of TSF_KEY containing keys of consolidated tone sets
	sizeoflist	• pointer to unsigned short that contains actual number of consolidated tone sets
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetConsolidatedToneSetKeys( )** function returns keys of tone sets consolidated in an array of **parray**. A maximum of 10 tone sets can be consolidated.

Parameter	Description
keylist	Specifies an array of TSF_KEY that contains keys of consolidated tone sets in the opened tone set file upon successful return of the function.
sizeoflist	This is an input/output parameter. As input parameter, location pointed by this pointer contains the number of elementsin the array keylist. As output parameter, the location pointed by this pointer provides the actual number of consolidated tone sets in the tone set file.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before using this function.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  unsigned short NumToneSets = 5;
  TSF_KEY *keylist;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   * then apply consolidation process.
   *.
   *.
   */
   ret = tsf_GetNumberOfConsolidatedToneSets(&NumConsToneSets);
   if( ret == TSF_SUCCESS )
   {
     printf("Number of tone sets in the tsf file is %d\n",NumConsToneSets);
   }
   else
   {
     printf("tsf_GetNumberOfConsolidatedToneSets() is fail\n");
   }

keylist = new TSF_KEY[NumConsToneSets];
ret = tsf_GetConsolidatedToneSetKeys(keylist,&NumConsToneSets);
   if( ret == TSF_SUCCESS )
   {
     printf("Number of consolidated tone sets in the tsf file is %d\n",NumConsToneSets);
     /*process keylist here */
   }
   else
   {
     printf("tsf_tsf_GetConsolidatedToneSetKeys() is fail\n");
   }

    /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
```

```
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_GetNumberOfToneSetsConsolidated( )**

Name:	TSF_RET tsf_GetDefaultToneSetInConsolidation (TONESETINFO *ptonesetinfo)	
Inputs:	None	
Outputs:	ptonesetinfo	• Pointer to a structure which contains default tone information of MAX_TONES tones to add in consolidation process.
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetDefaultToneSetInConsolidation( )** function returns info on default tone set included in the consolidation process.

Use this function to return the results of what you set in **tsf_SetDefaultToneSetToConsolidate( )**.

Parameter	Description
ptonesetinfo	Specifies a pointer to the structure variable of type TONESETINFO. This structure contains tone information for all MAX_TONES tones in the tone set. MAX_TONES is a define that specifies the maximum number of tones in a tone set. For more information on the structure, see *section 7.11. TONESETINFO*.

■ Cautions

None.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_CONSOLIDATIONOPTIONS conscfg;
  TONESETINFO default_info;
  TONESETINFO get_default_info;
  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */

  /* Consolidate the tone set file */
  /* Clear the TSF_CONSOLIDATIONOPTIONS structure first before tsf_ConsolidateToneSets()
*/
  tsf_ClearConsolidationOptions (&conscfg);

  conscfg.dial_tone_id = TID_DIAL_LCL;
  conscfg.exclude_default_defs = FALSE;    // so include default tone information in
consolidation process

/*clear out default tone information from consolidation*/
tsf_ClearDefaultToneSetToConsolidate();
/* here tsf_GetDefaultToneSetInConsolidation(&get_default_info) API returns all 0 in
get_default_info */

/* Set default tone information in default_info structure variable.
*
*
*
*/
/*set defaults */
tsf_SetDefaultToneSetToConsolidate ( &default_info);

tsf_GetDefaultToneSetInConsolidation(&get_default_info);
/* here get_default_info structure variable have same values in default_info
*/
```

```
  /* Consolidate tone set file */
  ret = tsf_ConsolidateToneSets(&conscfg);
  if (ret == TSF_SUCCESS) {
    printf("OK: tsf_ConsolidateToneSets ()\n");
  }
  else {
    printf("Fail: tsf_ConsolidateToneSets () ret=%d\n", ret);
  }

  /* Right after the tsf_ConsolidateToneSets (), Save the file  */
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }
  /* return from main() */
  return 0
}
```

See Also

- **tsf_SetDefaultToneSetToConsolidate()**
- **tsf_ClearDefaultToneSetToConsolidate()**
- **tsf_ConsolidateToneSets()**

Name:	TSF_RET tsf_GetFileInformation (TSF_FILE_INFO *pfile_info)	
Inputs:	none	
Outputs:	pfile_info	• Pointer to a local TSF_FILE_INFO structure
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetFileInformation( )** function gets updated information about a TSF after **tsf_OpenFile( )** has been called. The information about the tone set file is found in the TSF_FILE_INFO structure, which is pointed to by **pfile_info.**

Information about the tone set file is constantly updated by different operations on the file, such as adding a tone set, deleting a tone set and so on. At any point, you can use this function to get a snapshot of the state of the tone set file. Depending on previous activity on the file or functions invoked, the results will vary.

Parameter	Description
pfile_info	specifies a pointer to a local structure variable of type TSF_FILE_INFO, which holds the tone set file information.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before calling **tsf_GetFileInformation( )**.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info1, file_info0;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info0, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
   exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */

  ret = tsf_GetFileInformation(&file_info1);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_GetFileInformation()  ret= %d\n", ret);
  }

  /* Continue processing
   *.
   *.
   *.
   */

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }


  /* return from main() */
  return 0
}
```

■ See Also

- **tsf_OpenFile( )**

Name:	TSF_RET tsf_GetNumberOfToneSets(unsigned short *numoftonesets)	
Inputs:	None	
Outputs:	numoftonesets	• pointer to a variable of type unsigned short that holds number of tone sets in the opened tsf file
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetNumberOfToneSets( )** function returns the number of tone sets in an opened tone set file. This is a convenience function.

Parameter	Description
numoftonesets	Pointer to a variable of type unsigned short that holds number of tone sets in the opened tone set file. Successful execution of the function provides the number of tone sets in current opened tone set file.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before using this function.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  unsigned short NumToneSets;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */
   ret = tsf_GetNumberOfToneSets(&NumToneSets);
   if( ret == TSF_SUCCESS )
   {
     printf("Number of tone sets in the tsf file is %d\n",NumToneSets);
   }
   else
   {
     printf("tsf_GetNumberOfToneSets() failed\n");
   }

    /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

■ See Also

- **tsf_GetToneSetKeys()**

Name:	TSF_RET tsf_GetNumberOfToneSetsConsolidated(unsigned short *numoftonesets)	
Inputs:	None	
Outputs:	numoftonesets	• pointer to a variable of type unsigned short that holds number of consolidated tone sets in the opened tsf file
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetNumberOfToneSetsConsolidated( )** function returns number of consolidated tone sets in an opened tone set file.

Parameter	Description
numoftonesets	Successful execution of the function provides the number of consolidated tone sets in the current opened tone set file.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before using this function.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
```

```
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  unsigned short NumToneSets;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   * Consolidate tone sets
   *
   *
   */
   ret = tsf_GetNumberOfToneSetsConsolidated(&NumToneSets);
   if( ret == TSF_SUCCESS )
   {
     printf("Number of consolidated tone sets in the tsf file is %d\n",NumToneSets);
   }
   else
   {
     printf("tsf_GetNumberOfToneSetsConsolidated() failed\n");
   }

    /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

■ See Also

- **tsf_GetToneSetKeys()**

Name:	TSF_RET tsf_GetToneSet (TSF_KEY tsf_key, TONESETINFO *ptonesetinfo)	
Inputs:	tsf_key	• key of tone set
Outputs:	ptonesetinfo	• Tone information for all MAX_TONES
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetToneSet( )** function returns tone information for a specific tone set, specified in **tsf_key.**

Parameter	Description
tsf_key	specifies the key or identifier of the tone set
ptonesetinfo	specifies a pointer to the structure variable of type TONESETINFO. This structure contains tone information for all MAX_TONES tones in the tone set. This tone set is added to the tone set file. MAX_TONES is a define that specifies the maximum number of tones in a tone set. For more information on the structure, see *section 7.11. TONESETINFO*.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before calling **tsf_GetToneSet( )**.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_KEY tsf_key;
  TONESETINFO toninfoset;

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding, deleting tone set entry
   *.
   *.
   *.
   */
  /* find tone set ID of the tone set with key tsf_key */

  ret = tsf_GetToneSet (tsf_key, &toninfoset);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_GetToneSet ()\n");
  }
  else
  {
      printf("OK: tsf_GetToneSet\n");
/* process here information received in tonesetinfo.toneinfo[MAX_TONES]
*
*
*/
}
/* Continue processing
*
*
*/

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }
```

```
  /* return from main() */
return 0;
}
```

See Also

- **tsf_OpenFile( )**

Name:	TSF_RET tsf_GetToneSetKeys (TSF_KEY *keylist, unsigned short *sizeoflist)	
Inputs:	sizeoflist	pointer to unsigned short that contains number of elements in array keylist
Outputs:	keylist	array of TSF_KEY
	sizeoflist	pointer to unsigned short that contains actual number of tone sets
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetToneSetKeys( )** function returns the keys of the tone sets in an opened tone set file. This is a convenience function.

Parameter	Description
keylist	Specifies an array of TSF_KEY that contains the keys of the tone sets in the opened tone set file upon successful return of the function.
sizeoflist	This is an input/output parameter. As input parameter, the location pointed to by this parameter contains the number of elements in the array keylist. As output parameter, the location pointed by this parameter provides the actual number of tone sets in the tone set file.

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before using this function.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  unsigned short NumToneSets = 5;
  TSF_KEY *keylist;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */
   ret = tsf_GetNumberOfToneSets(&NumToneSets);
   if( ret == TSF_SUCCESS )
   {
     printf("Number of tone sets in the tsf file is %d\n",NumToneSets);
   }
   else
   {
     printf("tsf_GetNumberOfToneSets() failed\n");
   }

keylist = new TSF_KEY[NumToneSets];
ret = tsf_GetToneSetKeys(keylist,&NumToneSets);
   if( ret == TSF_SUCCESS )
   {
     printf("Number of tone sets in the tsf file is %d\n",NumToneSets);
     /*process keylist here */
   }
   else
   {
     printf("tsf_GetToneSetKeys() failed\n");
   }

    /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
```

```
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_GetNumberOfToneSets()**

Name:	TSF_RET tsf_GetToneSetName (TSF_KEY tsf_key , TONESET_ID *ptoneset_name)	
Inputs:	tsf_key	• key of tone set
Outputs:	ptoneset_name	• Tone set name of the TSF entry at toneset_offset in tone set file
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_GetToneSetName( )** function returns the tone set name of the tone set associated with **tsf_key.**

Parameter	Description
tsf_key	specifies the key or identifier of the tone set
ptoneset_name	specifies a pointer to the tone set ID information

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before calling **tsf_GetToneSetName( )**.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"
```

```
#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_KEY tsf_key;
  TONESET_ID tonset_name;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }


  /* Continue processing such as adding, deleting tone set entry
   *.
   *.
   *.
   */
  /* find tone set name of the tone set with key tsf_key */

  ret = tsf_GetToneSetName(tsf_key, &tonset_name);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_GetToneSetName ()\n");
  }
  else
  {
      printf("OK: tsf_GetToneSetKeys\n");
      printf("Name of tone set is %s\n", tonset_name.toneset_name);
      printf("Model of tone set is %s\n", tonset_name.toneset_model);
}
/* Continue processing
*
*
*/


  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
return 0;
}
```

See Also

- **tsf_DeleteToneSet( )**

Name: TSF_RET tsf_ModifyToneSet (TSF_KEY tsf_key, TONESET_ID *ptoneset_name, TONESETINFO *ptonesetinfo)

Inputs:

tsf_key	• A key indicating which tone set to modify
ptoneset_name	• Tone set ID information to be written for this tone set
ptonesetinfo	• Tone information to be written for this tone set

Outputs: none
Returns: error code
Includes: tsfiolib.h
Category: TSF
Mode: synchronous
Platform: DM3, Springware

■ Description

The **tsf_ModifyToneSet( )** function modifies a tone set with the information specified in **ptonesetinfo**. If either the ID information or tone information is to remain unchanged, NULL may be specified for the appropriate argument; however, at least one of the pointers must be non-NULL.

Parameter	Description
tsf_key	specifies the key indicating which tone set to modify
ptoneset_name	specifies a pointer to the tone set ID information to be written for a particular tone set
ptonesetinfo	specifies the information to be written for a particular tone set

■ Cautions

You must create or open a tone set file using **tsf_OpenFile( )** before modifying a tone set entry.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TONESETINFO new_tonesetinfo;
  TONESET_ID  new_toneset_id;
  TSF_KEY   my_key;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */

  /* Before calling tsf_ModifyToneSet(), make a new_toneset_id and new_tone_info you
    desire. Modify tone set with key 1 here.
  */
  my_key = 1;
  ret = tsf_ModifyToneSet(my_key, &new_toneset_id, &new_tonesetinfo);
  if (ret == TSF_SUCCESS) {
    printf("OK: tsf_ModifyToneSet()\n");
  }
  else {
    printf("Fail: tsf_ModifyToneSet() ret=%d\n",ret);
  }

  /* Continue processing such as consolidation
   *.
   *.
   *.
   */

  /* Save the file */
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
```

```
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_DuplicateToneSet( )**
- **tsf_AddToneSet( )**

Name:	TSF_RET tsf_OpenFile (char * filename, TSF_FILE_INFO *pfile_info, unsigned short rfu)	
Inputs:	filename	• name of the tone set file to be opened
	rfu	• always 0
Outputs:	pfile_info	• pointer to a local FILE_INFO structure
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_OpenFile( )** function opens the tone set file specified in the **filename** parameter. If the file does not exist, this function creates a new one using the name specified in the **filename** parameter. Initial information about the file is copied into the TSF_FILE_INFO structure pointed to by **pfile_info**.

Parameter	Description
filename	specifies the name of the tone set file to be opened or created
rfu	reserved for future use. Always = 0
pfile_info	specifies a pointer to a local TSF_FILE_INFO structure. For more information about this structure, see *section 7.13. TSF_FILE_INFO*.

■ Cautions

- Only one tone set file can remain open at any point in time.
- Calling **tsf_OpenFile( )** before closing an existing open tone set file results in failure. There must be a **tsf_CloseFile( )** function call for each **tsf_OpenFile( )** function call.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;


  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as ToneSet entry and consolidation
   *.
   *.
   *.Save changes
   */
  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

■ See Also

- **tsf_GetFileInformation( )**
- **tsf_CloseFile( )**

Name:	TSF_RET tsf_SaveFile (char *filename, unsigned short flag)	
Inputs:	filename	• Alternate filename with which to save a TSF
	flag	• A Boolean indicating if an existing TSF may be overwritten
Outputs:	none	
Returns:	error code	
Includes:	tsfiolib.h	
Category:	TSF	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **tsf_SaveFile( )** function saves tone set data to a tone set file. If NULL is specified in **filename**, then the file is saved with the filename specified when the file was originally opened with **tsf_OpenFile( )**. If a file name is specified in **filename**, then that file name is used.

You should always call this function after consolidating tone sets using **tsf_ConsolidateToneSets( )** in order to save the results of the consolidation.

Parameter	Description
filename	Specifies an alternate filename with which to save a tone set file. Specify NULL to save the file using the same name with which it was opened.
flag	Specifies a Boolean expression indicating if an existing tone set file may be overwritten. Values are TRUE or FALSE.

■ Cautions

The tone set file must be open before it can be saved.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

#define TSF_FILE "intltsf.tsf"

int main() {
  int ret;
  TSF_FILE_INFO file_info;

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
  exit(1);
  }


  /* Continue processing such as adding, deleting Tone Set entry
   *.OR process consolidation
   *.
   *.
   */

  /* Don't forget to save the file to the name as specified to tsf_OpenFile() after
adding or modifying the file */
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }


  /* return from main() */
  return 0
}
```

■ See Also

- **tsf_OpenFile( )**
- **tsf_CloseFile( )**

Name: TSF_RET tsf_SetDefaultToneSetToConsolidate(TONESETINFO *ptonesetinfo)

Inputs: ptonesetinfo
- Pointer to a structure which contains default tone information of MAX_TONES tones to add in consolidation process

Outputs: None
Returns: error code
Includes: tsfiolib.h
Category: TSF
Mode: synchronous
Platform: DM3, Springware

■ Description

The **tsf_SetDefaultToneSetToConsolidate( )** function adds default tone set information to the consolidation process.

Parameter	Description
ptonesetinfo	specifies pointer to a structure variable of type TONESETINFO which contains default tone information for MAX_TONES tones.

■ Cautions

- If you want to add default tone set information to the consolidation process, you must call **tsf_SetDefaultToneSetToConsolidate( )** before calling **tsf_ConsolidateToneSets( )**. The **tsf_SetDefaultToneSetToConsolidate( )** function will not have any effect after the consolidation process is performed.
- This function must be called according to the exclude_default_defs field setting in the TSF_CONSOLIDATIONOPTIONS structure.
- If exclude_default_defs is set to TRUE, then you must call this function before **tsf_ConsolidateToneSets( )** to set default tone information for consolidation. Otherwise the consolidation fails with error code TSF_CONSOLIDATION_DEFAULTS_NOT_SET.

- If exclude_default_defs is set to FALSE, then you can take one of two actions: (1) you must not call this function to set default tone information for consolidation or (2) you must clear default information using **tsf_ClearDefaultToneSetToConsolidate()** after calling the function. Otherwise the consolidation fails with error code TSF_CONSOLIDATION_DEFAULTS_NOT_REQUESTED.

■ Errors

This function returns TSF_SUCCESS to indicate success. This function returns an error code in case of error. For a description of error codes, see *section 6.2. Tone Set File Error Codes*.

■ Example

```
#include <stdio.h>
#include "srllib.h"
#include "lmodelib.h"
#include "tsfiolib.h"

int main() {
  int ret;
  TSF_FILE_INFO file_info;
  TSF_CONSOLIDATIONOPTIONS conscfg;
  TONESETINFO default_info;

  /*  tsf_OpenFile()  */

  ret = tsf_OpenFile(TSF_FILE, &file_info, 0);
  if (ret != TSF_SUCCESS) {
     printf("Fail: tsf_OpenFile() : retvalue = %d\n", ret);
     exit(1);
  }

  /* Continue processing such as adding Tone Set entry
   *.
   *.
   *.
   */

  /* Consolidate the tone set file */
  /* Clear the TSF_CONSOLIDATIONOPTIONS structure first before tsf_ConsolidateToneSets()
*/
  tsf_ClearConsolidationOptions (&conscfg);

  conscfg.dial_tone_id = TID_DIAL_LCL;
  conscfg.exclude_default_defs = FALSE;                // so include default tone
information in consolidation process

/*clear out default tone information from consolidation*/
tsf_ClearDefaultToneSetToConsolidate();

/* Set default tone information in default_info structure variable.
*
```

```
*
*
*/
/*set defaults */
tsf_SetDefaultToneSetToConsolidate ( &default_info);

  /* Consolidate tone set file */
  ret = tsf_ConsolidateToneSets(&conscfg);
  if (ret == TSF_SUCCESS) {
    printf("OK: tsf_ConsolidateToneSets ()\n");
  }
  else {
    printf("Fail: tsf_ConsolidateToneSets () ret=%d\n", ret);
  }

  /* Right after the tsf_ConsolidateToneSets (), Save the file  */
  ret = tsf_SaveFile(NULL, TRUE);
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_SaveFile()\n");
  }

  /* Close file */
  ret = tsf_CloseFile();
  if (ret != TSF_SUCCESS) {
    printf("Fail: tsf_CloseFile()\n");
  }

  /* return from main() */
  return 0
}
```

See Also

- **tsf_GetDefaultToneSetInConsolidation()**
- **tsf_ClearDefaultToneSetToConsolidate()**
- **tsf_ConsolidateToneSets()**

6. Error Codes

6.1. Learn Mode Error Codes

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

Name	Description
EDX_CADFLAG	Invalid continuous/cadence flag (lm_cadflag) in LM_PARM structure
EDX_CONTIM	Invalid continuous tone minimum on-time (lm_cnt_min) in LM_PARM structure
EDX_FRAMES	Invalid number of frames (lm_frames) in LM_PARM structure
EDX_FREQTOL	Invalid learn mode frequency tolerance (lm_frq_tol) in LM_PARM structure
EDX_LMDURTOL	Invalid learn mode duration tolerance (lm_dur_tol) in LM_PARM structure
EDX_LMMETHOD	Invalid frequency/tolerance learning method (lm_method) in LM_PARM structure
EDX_LMPARM	Invalid parameters in LM_PARM structure
EDX_LMQUALID	Invalid learn mode qualification template ID (lm_qualid) in LM_PARM structure
EDX_TNAMP	Invalid tone amplitude in TN_AMP structure
EDX_TNDFLAG	Invalid dual/single tone flag (dflagp)
EDX_TNINFO	Not enough tone information to do the learning (requires 5 frames/samples)
EDX_TNINVALID	Invalid tone detected
EDX_TNPARM	Invalid tone template parameter

6.2. Tone Set File Error Codes

Tone set file error codes can be found in *tsfiolib.h* and include the following:

Name	Description
TSF_ACTIVATION_FAIL	The **tsf_ActivateFile( )** function failed. Can only activate file after file has been consolidated.
TSF_CONSOLIDATION _DEFAULTS_NOT_SET	Default tone information request to be included in consolidation process but not set default tone information.
TSF_CONSOLIDATION_ DEFAULTS_NOT_REQUESTED	Default tone information is set but not requested to include in compilation.
TSF_CONSOLIDATION_BLANK _ENTRY	A tone set to be consolidated contains no tone set data. A tone set entry has zeros defined for freq1 and freq2 tone definitions.
TSF_CONSOLIDATION_CAD _CONT_MIX	Tones of the same category (e.g., ringback or busy) in different tone sets have a mix of continuous and cadenced tone definitions. For instance, TonesetA has continuous tone defined for ringback and TonesetB has a cadenced tone defined for ringback. If this is true, TonesetA and TonesetB cannot both be marked for consolidation.
TSF_CONSOLIDATION_FREQ _TOO_WIDE	Deviation of the frequency component of a tone set is greater than the value defined in COMP_MAX_FREQ_DEV. The maximum range allowed is 420 (i.e., a deviation of 420/2 = 210).
TSF_CONSOLIDATION_GENERAL	When the error doesn't fit any of the other descriptions, this error is generated.
TSF_CONSOLIDATION_INVALID_ DTONE	A dial tone definition in the list of PBXs is not valid. It contains a cadence

Name	Description
	definition. This excludes debounce (0 offtime and negative ontime deviation).
TSF_CONSOLIDATION_NO_DATA	A consolidation was requested and there are no entries in the build list.
TSF_CONSOLIDATION_OVERLAP	The consolidation of the tone sets resulted in a conflict between the ringback and busy templates.
TSF_CONSOLIDATION_QUAL _TEMP	Across a given template definition for the set of PBXs, there are both single and dual tone definitions for the same tone.
TSF_CONSOLIDATION_SINGLE _DUAL	Tones of the same category (e.g., ringback or busy) in different tone sets have a mix of single and dual tone definitions. For instance, TonesetA has a single frequency tone defined for ringback and TonesetB has a dual frequency tone defined for ringback. If this is true, TonesetA and TonesetB cannot both be consolidated.
TSF_CONSOLIDATION_SUCCESS	Successful consolidation. Tone set file is written to the disk.
TSF_CONSOLIDATION_TOO _MANY	Too many tone sets were chosen for consolidation. Only 10 tone sets may be consolidated.
TSF_FAILURE	Tone set file operation failed.
TSF_FILE_EXISTS	Filename selected for the save operation already exists.
TSF_FILE_NOT_CLOSED	The file has not been closed, so an open function cannot be performed.
TSF_FILE_NOT_CONSOLIDATED	The file does not have a consolidated tone set.
TSF_FILE_NOT_OPENED	The file has not been opened, so no

Name	Description
	activity can occur.
TSF_INVALID_FILENAME	Filename used is not a valid one.
TSF_INVALID_FILETYPE	Invalid file type
TSF_INVALID_KEY	Key passed is invalid.
TSF_INVALID_PARAMETER	Invalid parameter or null argument passed
TSF_IS_DELETED	The tone set is deleted
TSF_MISSING_ARG	Missing argument
TSF_NO_MEM	The system has no remaining memory.
TSF_READONLY	The file is read-only.
TSF_SUCCESS	Tone set file operation successful.

7. Data Structure Reference

7.1. Data Structures Overview

This chapter provides an alphabetical reference to the data structures used by the learn mode API library and tone set file API library functions.

The following data structures, defined in the *dxxxlib.h*, are used by learn mode API 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

The following data structures, defined in *tsfiolib.h*, are used by tone set file API library functions:

- TONE_INFO – tone information structure
- TONESET_ID – tone set identification structure
- TONESETINFO – tone set information structure
- TSF_CONSOLIDATIONOPTIONS – consolidation options structure
- TSF_FILE_INFO – tone set file information structure

7.2. 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.

Field	Description
	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).
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,

Field	Description	
		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.
	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

Field	Description	
		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
		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

Field	Description	
		range.
	LM_STATISTICAL	Units: Percentage Range: 0 – 0.2 Default: 0.03 Specifies the percentage of extreme samples to exclude from the normal distribution.

7.3. 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

Field	Description
	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 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).

7.4. 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

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

7.5. 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 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.

7.6. 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

Field	Description
	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 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.

7.7. 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.

Parameter	Description
tn_rep_cnt	Length: 2 (unsigned short int) Default: 0 Tone-On/Off Repetition Count: *Not Used for learn mode tone information (always 0).*

7.8. 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.

7.9. TONE_INFO

The TONE_INFO structure is declared as follows:

```
typedef  struct {
   unsigned short tone_id;    /* tone ID */
   TN_DESC        tn_desc;    /* tone description structure */
   TN_AMP         tn_amp      /* tone amplitude structure */
} TONE_INFO;
```

The TONE_INFO structure contains tone template information such as frequency and cadence for a particular tone.

Table 8. TONE_INFO Structure

Field	Description
tone_id	Specifies the tone ID. Possible values are: TID_DIAL_LCL TID_DIAL_INTL TID_DIAL_XTRA TID_BUSY1 TID_RNGBK1 TID_BUSY2 TID_RNGBK2 TID_DISCONNECT TID_FAX1 TID_FAX2
tn_desc	Specifies tone characteristics in the TN_DESC data structure. See TN_DESC data structure for more information.
tn_amp	Specifies tone amplitude boundaries in the TN_AMP data structure. See TN_AMP data structure for more information.

7.10. TONESET_ID

The TONESET_ID structure is declared as follows:

```
typedef  struct {
   unsigned char toneset_name  [TONESET_STRING_SIZE];    /* manufacturer's name */
   unsigned char toneset_model [TONESET_STRING_SIZE];    /* model name */
   unsigned short isConsolidated;                         /* included in build flag */
} TONESET_ID;
```

The TONESET_ID structure stores the name, model, and build state of a tone set. Build state indicates whether the tone set is included in the consolidation.

Table 9. TONESET_ID Structure

Field	Description
toneset_name	Contains the manufacturer name of the phone switch.
toneset_model	Contains the model name of the phone switch.
isconsolidate	Indicates whether the tone set is included in the consolidation of the tones. Possible values are True (1) or False (0).

7.11. TONESETINFO

The TONESETINFO structure is declared as follows:

```
typedef  struct {
   TONE_INFO toneinfo[MAX_TONES];                    /* tone info for all MAX_TONE
tones */
} TONESETINFO;
```

The TONESETINFO structure stores tone information for all tones in a tone set.

Table 10. TONESETINFO Structure

Field	Description
toneinfo	Specifies tone information for all tones in the TONE_INFO structure (MAX_TONES define sets the maximum number of tones in a tone set).

7.12. TSF_CONSOLIDATIONOPTIONS

The TSF_CONSOLIDATIONOPTIONS structure is declared as follows:

```
typedef  struct {
   int dial_tone_id;
   int exclude_default_defs;     /* Use the macros TRUE/FALSE */
} TSF_CONSOLIDATIONOPTIONS;
```

The TSF_CONSOLIDATIONOPTIONS structure is used to set options for the consolidation process.

Table 11. TSF_CONSOLIDATIONOPTIONS Structure

Field	Description
dial_tone_id	Specifies the dial tone ID. Possible values are: TID_DIAL_LCL TID_DIAL_INTL TID_DIAL_XTRA
exclude_default_defs	Indicates whether the default tone definitions are excluded from the consolidation or not. Possible values are True (1) and False (0).

7.13. TSF_FILE_INFO

The TSF_FILE_INFO structure is declared as follows:

```
typedef  struct {
    unsigned short  consolidateflag;   /* TRUE/FALSE - If file contains data   */
    unsigned short  tonesetcount;     /* Total # of TONESET entries           */
    unsigned short  maxtones;         /* Total # of TONE_INFO entries per ts  */
    unsigned short  dnldtscount;      /* Total # of DNLD_TSET entries         */
    unsigned short  gtdtscount;       /* Provisional                          */
    unsigned short  tonesetsize;      /* Size of TONESET structure            */
    unsigned short  dnldsize;         /* Size of Download Toneset structure   */
    unsigned short  gtdtssize;        /* Provisional                          */
    unsigned short  tonesetsinconsolidation; /* Total # of tonesets in build  */
    unsigned char   tid_dt;           /* Tone ID in build for dial tone       */
} TSF_FILE_INFO;
```

The TSF_FILE_INFO structure contains runtime information about an opened tone set file.

Table 12. TSF_FILE_INFO Structure

Field	Description
binarydataflag	Indicates whether the file contains any consolidated tone set information. The only valid information is True = 1 or False = 0.
tonesetcount	Contains the total number of tone set entries.
maxtones	Contains the total number of entries per tone set.
dnldtscount	Contains the total number of consolidated tone sets.
gtdtscount	This field is currently not used and defaults to 0.

Field	Description
tonesetsize	Contains the size of the tone set structure.
dnldsize	Contains the size of the consolidated tone set structure.
gtdtssize	This field is currently not used and defaults to 0.
tonesetinconsolidation	Contains the total number of tone sets in the consolidated build.
tid_dt	Contains the tone set ID in the consolidated build for the default dial tone.

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.

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*).

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.

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

H

I

L

P

S

T