COPYRIGHT NOTICE

Copyright 1998 Dialogic Corporation. All Rights Reserved.

All contents of this document are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation. Every effort is made to ensure the accuracy of this information. However, due to ongoing product improvements and revisions, Dialogic Corporation cannot guarantee the accuracy of this material, nor can it accept responsibility for errors or omissions. No warranties of any nature are extended by the information contained in these copyrighted materials. Use or implementation of any one of the concepts, applications, or ideas described this document or on Web pages maintained by Dialogic-may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not condone or encourage such infringement. Dialogic makes no warranty with respect to such infringement, nor does Dialogic waive any of its own intellectual property rights which may cover systems implementing one or more of the ideas contained herein. Procurement of appropriate intellectual property rights and licenses is solely the responsibility of the system implementer. The software referred to in this document is provided under a Software License Agreement. Refer to the Software License Agreement for complete details governing the use of the software.

All names, products, and services mentioned herein are the trademarks or registered trademarks of their respective organizations and are the sole property of their respective owners. DIALOGIC (including the Dialogic logo), DTI/124, SpringBoard, and Signal Computing System Architecture (SCSA) are registered trademarks of Dialogic Corporation. The following are also trademarks of Dialogic Corporation:

Publication Date: March, 1998
Part Number:05-0774-004
Dialogic Corporation
1515 Route 10
Parsippany NJ 07054

Technical Support
Phone: 973-993-1443
Fax: 973-993-8387
BBS: 973-993-0864
Email: CustEng@dialogic.com

For Sales Offices and other contact information, visit our website at http://www.dialogic.com
Table of Contents

1. System Features ................................................................. 1
   1.1. SCbus Routing Functions ................................................. 1
   1.2. Convenience Functions .................................................. 2
   1.3. File Manipulation Functions ............................................. 2
   1.4. C Language Interfaces ................................................... 4
       1.4.1. Run-Time Linking ...................................................... 4
   1.5. Compatibility Library Functions ....................................... 5
   1.6. The Wait for Events Function .......................................... 17
   1.7. Retrieving Dialogic DLL Version Numbers ......................... 17
   1.8. Dialogic DLL Version Number Functions ......................... 17

2. Voice ................................................................. 31
   2.1. Play and Record Functions ............................................ 31
   2.2. Voice Features .......................................................... 31
       2.2.1. Call Progress Features ............................................. 31
       2.2.2. Tone Features ....................................................... 32
   2.3. Caller ID ................................................................. 34
       2.3.1. Caller ID Overview ................................................ 34
       2.3.2. Caller ID Formats ................................................ 34
       2.3.3. Accessing Caller ID Information ............................. 36
       2.3.4. Error Handling ...................................................... 38
       2.3.5. Enabling Channels to Use the Caller ID Feature .......... 38
       2.3.6. Caller ID Demonstration Programs ......................... 39
   2.4. Caller ID Functions ..................................................... 39
       2.4.1. Caller ID Related References ................................. 58
   2.5. Global Dial Pulse Detection .......................................... 60
       2.5.1. Global Dial Pulse Detection Overview ..................... 60
       2.5.2. Supported Applications .......................................... 61
       2.5.3. Regional DPD Parameters ...................................... 62
       2.5.4. Programming Considerations for Accurate Global DPD 63
       2.5.5. Global DPD Application Programming Interface ........ 63
   2.6. Transaction Record .................................................... 66
       2.6.1. Overview ............................................................ 66
       2.6.2. Supported Voice Boards ....................................... 66
   2.7. Transaction Record Function .......................................... 66
   2.8. Get Feature List Function ............................................. 73
dx_getfeaturelist( ) - returns list of features supported on voice device...........74

3. ProLine/2V ..................................................................................................81
   3.1. ProLine/2V Audio Jacks Overview ......................................................81
   3.2. Line In and Line Out Audio Jack Specifications .....................................82
   3.3. How To Switch the ProLine/2V Channel 2 Input .....................................84
      3.3.1. Cautions .......................................................................................85
      3.3.2. Example .......................................................................................85
   3.4. Using the Audio Jack with the SAMPLE Demonstration Program ..........86

4. ISDN .............................................................................................................87
   4.1. Overview ..............................................................................................87
   4.2. Installation Procedure ...........................................................................87
   4.3. ISDN Features Summary .......................................................................87
   4.4. Diagnostic Tools (The DialView Suite) .....................................................88

5. MSI/SC .........................................................................................................91
   5.1. ms_genringex( ) function .....................................................................91
       ms_genringex( ) - generates distinctive ringing ......................................92

6. DCB/SC .......................................................................................................99
   6.1. Features ...............................................................................................99
   6.2. Notes and Hints ..................................................................................100

7. TAPI/WAVE 2.0 ........................................................................................101
   7.1. TAPI/WAVE 2.0 Requirements ............................................................101
   7.2. TAPI/WAVE 2.0 Summary ..................................................................101
       7.2.1. Telephony Service Provider (TSP) and TAPI ..........................101
       7.2.2. WAVE Device Driver ...............................................................103
   7.3. Installing and Removing the Dialogic TSP and WAVE .........................106
       7.3.1. TAPI/WAVE 2.0 Requirements ...............................................107
       7.3.2. Activate the Dialogic TSP for TAPI ........................................107
       7.3.3. Activate the Dialogic WAVE Driver ........................................108
       7.3.4. Uninstalling the Software .........................................................109

8. Country-Specific Parameters .....................................................................111
   8.1. Country-Specific Parameter System Overview ......................................111
       8.1.1. New Country-Specific Support ..................................................111
       8.1.2. New Custom (Global DPD) Support ..........................................111
       8.1.3. New Generic (Global DPD) Support .........................................112
       8.1.4. Country-Specific Parameter File ...............................................112
   8.2. Supported Countries ...........................................................................112
   8.3. Dialogic Boards Supported by Country ...............................................114
8.4. Firmware Parameter Features ................................................................. 117
8.5. Intelligent Network Interfaces Products ................................................. 118
8.6. Modifying Network Parameters .............................................................. 119
Index ........................................................................................................... 121
1. System Features

C language interfaces to the individual technology DLLs (standard runtime, voice, fax, voice recognition, and network interfaces) are provided.

1.1. SCbus Routing Functions

The functions included in the Voice and Network Library to control the SCbus routing are listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ag_getctinfo( )</td>
<td>Returns information about an analog device connected to the SCbus.</td>
</tr>
<tr>
<td>ag_getxmitslot( )</td>
<td>Returns the SCbus time slot connected to the transmit of an analog channel.</td>
</tr>
<tr>
<td>ag_listen( )</td>
<td>Connects the receive (listen) of an analog channel to an SCbus time slot.</td>
</tr>
<tr>
<td>ag_unlisten( )</td>
<td>Disconnects the receive (listen) of an analog channel from an SCbus time slot.</td>
</tr>
<tr>
<td>dt_getctinfo( )</td>
<td>Returns information for a digital network device time slot.</td>
</tr>
<tr>
<td>dt_getxmitslot( )</td>
<td>Returns the SCbus time slot connected to a digital network device time slot.</td>
</tr>
<tr>
<td>dt_listen( )</td>
<td>Connects the receive (listen) of a digital network device time slot to the SCbus.</td>
</tr>
<tr>
<td>dt_unlisten( )</td>
<td>Disconnects the receive (listen) of a digital network device time slot from the SCbus.</td>
</tr>
<tr>
<td>dx_getctinfo( )</td>
<td>Returns information about a voice device connected to the SCbus.</td>
</tr>
<tr>
<td>dx_getxmitslot( )</td>
<td>Returns the SCbus time slot connected to the transmit of a voice channel.</td>
</tr>
<tr>
<td>dx_listen( )</td>
<td>Connects the receive (listen) of a voice channel to an SCbus time slot.</td>
</tr>
<tr>
<td>dx_unlisten( )</td>
<td>Disconnects the receive (listen) of a voice channel from an SCbus time slot.</td>
</tr>
</tbody>
</table>
### 1.2. Convenience Functions

The following Convenience functions that control SCbus routing are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nr_scroute()</code></td>
<td>Makes a half or full duplex connection between two SCbus devices.</td>
</tr>
<tr>
<td><code>nr_scunroute()</code></td>
<td>Breaks a half or full duplex connection between two SCbus devices.</td>
</tr>
</tbody>
</table>

For more information on these functions, refer to the *SCbus Routing Function Reference for Windows NT*.

**NOTE:** The Voice Library does not include these Convenience functions. Code for these functions is provided in a separate C source file, `sctools.c`, which is installed in the following directory by default:

```
<install drive>:<install directory>\dialogic\sctools
```

### 1.3. File Manipulation Functions

Many file manipulation functions were added to the Voice API. These functions map to “C” runtime functions, and can only be used if the file is opened with the `dx_fileopen()` function. The new functions and the equivalent Microsoft Visual C++ runtime function that they map to are as follows:
1. System Features

<table>
<thead>
<tr>
<th>New File Manipulation Functions</th>
<th>Equivalent Microsoft Visual C++ Runtime Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>int dx_fileopen ( const char * filep, int flags, …)</td>
<td>int _open ( const char * filep, int flags, …)</td>
</tr>
<tr>
<td>int dx_filesopen ( const char * filep, int oflag, int shflag, …)</td>
<td>int _sopen ( const char * filep, int oflag, int shflag, …)</td>
</tr>
<tr>
<td>int dx_filelocking ( int handle, int mode, long bytes)</td>
<td>int _locking ( int handle, int mode, long bytes)</td>
</tr>
<tr>
<td>int dx_filechsize ( int handle, long size)</td>
<td>int _chszie ( int handle, long size )</td>
</tr>
<tr>
<td>int dx_filecommit ( int handle )</td>
<td>int _commit ( int handle )</td>
</tr>
<tr>
<td>int dx_filedup ( int handle )</td>
<td>int _dup ( int handle )</td>
</tr>
<tr>
<td>int dx_filedup2 ( int handle1, int handle2 )</td>
<td>int _dup2 ( int handle1, int handle2 )</td>
</tr>
<tr>
<td>int dx_fileumask ( int pmode)</td>
<td>int _umask ( int pmode )</td>
</tr>
<tr>
<td>long dx_filetell ( int handle)</td>
<td>int _tell ( int handle )</td>
</tr>
<tr>
<td>int dx_filestat ( const char* path, struct _stat * buffer)</td>
<td>int _stat ( const char * path, struct _stat * buffer )</td>
</tr>
<tr>
<td>int dx_filefstat ( int handle, struct _stat * buffer)</td>
<td>int _fstat ( int handle, struct _stat * buffer )</td>
</tr>
<tr>
<td>long dx_filefilelength( int handle )</td>
<td>long _filelength( int handle )</td>
</tr>
<tr>
<td>int dx_fileisatty( int handle)</td>
<td>int _isatty( int handle)</td>
</tr>
<tr>
<td>int dx_filesetmode( int handle, int mode )</td>
<td>int _setmode( int handle, int mode )</td>
</tr>
<tr>
<td>int dx_fileopen_osfhandle(long osfhandle, int flags )</td>
<td>int _open_osfhandle(long osfhandle, int flags)</td>
</tr>
</tbody>
</table>
1.4. C Language Interfaces

Simple C language interfaces in source-code format are provided to each individual technology DLL (standard runtime, voice, fax, voice recognition, and network interfaces). 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. The following libraries are provided for compile-time linking and are installed in the LIB subdirectory under the Dialogic home directory (normally \Program Files\Dialogic):

LIBSRLMT.LIB
LIBDXXMT.LIB
LIBDTIMT.LIB
LIBFAXMT.LIB

1.4.1. Run-Time Linking

Run-time linking resolves the entry points to the Dialogic DLLs when the application is loaded and executed. This allows the application to contain function calls that are not contained in the DLL that resides on the target system.

The following files are provided for run-time linking and are installed in the CLIB subdirectory under the Dialogic home directory (normally \Program Files\Dialogic):

SRLLIB.C and SRLLIB.CPP
DXXXLIB.C and DXXXLIB.CPP
FAXLIB.C and FAXLIB.CPP
DTILIB.C and DTILIB.CPP
MSILIB.C and MSILIB.CPP
CCLIB.C and CCLIB.CPP

To use run-time linking, the application must first call the technology xx_libinit() functions, where xx specifies the technology (sr, dx, fx, dt or vr). All other
1. System Features

Dialogic function calls are the same as when using compile-time linking. The technology `xx_libinit()` functions provided are as follows:

- `dx_libinit()` - Initializes the Voice Library DLL
- `fx_libinit()` - Initializes the Fax Library DLL
- `sr_libinit()` - Initializes the Standard Runtime Library DLL
- `dt_libinit()` - Initializes the Network Interface Library DLL
- `vr_libinit()` - Initializes the Voice Recognition Library DLL

The `xx_libinit()` function calls the `LoadLibrary()` function to load a specific Dialogic technology DLL. If the DLL does not exist, all its functions are set up as default Not Implemented Functions. If the DLL does exist, the `xx_libinit()` function performs a series of `GetProcAddress()` function calls to set up the address pointers for the functions.

### Library Interfaces

<table>
<thead>
<tr>
<th>Technology</th>
<th>.C and .CPP Files</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voice</td>
<td>DXXXLIB</td>
<td>DXXXLIB.H</td>
</tr>
<tr>
<td>Fax</td>
<td>FAXLIB</td>
<td>FAXLIB.H</td>
</tr>
<tr>
<td>Standard Runtime</td>
<td>SRLLIB</td>
<td>SRLLIB.H</td>
</tr>
<tr>
<td>Network Interfaces</td>
<td>CCLIB</td>
<td>CCLIB.H</td>
</tr>
<tr>
<td></td>
<td>DTILIB</td>
<td>DTILIB.H</td>
</tr>
<tr>
<td></td>
<td>MSILIB</td>
<td>MSILIB.H</td>
</tr>
<tr>
<td>Voice Recognition</td>
<td>VRXXSLIB</td>
<td>VRXXSLIB.H</td>
</tr>
</tbody>
</table>

The Network Interfaces include the MSI/SC and T-1 interfaces. Throughout this Release Reference, these interfaces are referred to as the Network DLL.

1.5. Compatibility Library Functions
dx_libinit initializes the Voice Library DLL

Name: dx_libinit ( flags )
Inputs: unsigned short flags
• Specifies the programming model
Returns: 0 if success
-1 if failure
Includes: srllib.h
dxxxlib.h

Description

The dx_libinit() function initializes the Voice Library DLL by loading and resolving all entry points in LIBDXXMT.DLL.

This function has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| flags     | This flag has two possible values:  
            DLGC_MT - Specify if using a multi-threaded or window callback model.  
            DLGC_ST - Specify if using the single threaded model. |

Cautions

The sr_libinit() function must be called prior to using the dx_libinit() functions.
1. System Features

### Example

```c
/*$ dx_libinit() example $*/
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>

int InitDevices()
{
  DWORD dwfilever, dwprodver;

  /*************************************************************************
  * Initialize all the DLLs required. This will cause the DLLs to be
  * loaded and entry points to be resolved. Entry points not resolved
  * are set up to point to a default not implemented function in the
  * 'C' library. If the DLL is not found all functions are resolved
  * to not implemented.
  ********************************************************************************/

  if (sr_libinit(DLGC_MT) == -1) {
    /* Must be already loaded, only reason if sr_libinit() was already called */
  }

  /*************************************************************************
  * Call technology specific dx_libinit() functions to load Voice DLL */
  if (dx_libinit(DLGC_MT) == -1) {
    /* Must be already loaded, only reason if dx_libinit() was already called */
  }

  /*************************************************************************
  * Voice library initialized so all other Voice functions may be called
  * as normal. Display the version number of the DLL
  ********************************************************************************/
  dx_GetDllVersion(&dwfilever, &dwprodver);
  printf("File Version for Voice DLL is %d.%02d\n", 
            HIWORD(dwfilever), LOWORD(dwfilever));
  printf("Product Version for Voice DLL is %d.%02d\n",
            HIWORD(dwprodver), LOWORD(dwprodver));

  /* Now open all the Voice devices */
}
```

### Errors

The `dx_libinit()` function fails if the library has already been initialized. For example, if you try to make a second call to `sr_libinit()`, it fails.

### See Also

- `fx_libinit()`
- `sr_libinit()`
### fx_libinit initializes the Fax Library DLL

**Name:** fx_libinit (flags)

**Inputs:**
- unsigned short flags

**Description**

The `fx_libinit()` function initializes the Fax Library DLL by loading and resolving all entry points in `LIBFAXMT.DLL`.

This function has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| flags     | This flag has two possible values:  
|           | DLGC_MT - Specify if using a multi-threaded or window callback model.  
|           | DLGC_ST - Specify if using the single threaded model. |

**Returns:**
- 0 if success
- -1 if failure

**Includes:**
- srllib.h
- dxxxlib.h

### Description

The `fx_libinit()` function initializes the Fax Library DLL by loading and resolving all entry points in `LIBFAXMT.DLL`.

### Cautions

The `sr_libinit()` function must be called prior to using the `fx_libinit()` functions.
1. System Features

Example

/* $fx_libinit()$ example */

#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int InitDevices()
{
    DWORD dwfilever, dwprodver;

    /*************************************************************************
    * Initialize all the DLLs required. This will cause the DLLs to be
    * loaded and entry points to be resolved. Entry points not resolved
    * are set up to point to a default not implemented function in the
    * 'C' library. If the DLL is not found all functions are resolved
    * to not implemented.
    *************************************************************************/
    if (sr_libinit(DLGC_MT) == -1) {
        /* Must be already loaded, only reason if sr_libinit() was already called */
    }
    /* Call technology specific dx_libinit() functions to load Voice DLL */
    if (dx_libinit(DLGC_MT) == -1) {
        /* Must be already loaded, only reason if dx_libinit() was already called */
    }
    /* Call technology specific fx_libinit() functions to load VFX Fax DLL */
    if (fx_libinit(DLGC_MT) == -1) {
        /* Must be already loaded, only reason if dx_libinit() was already called */
    }
    /*************************************************************************
    * Fax library initialized so all other VFX functions may be called as normal.
    * Display the version number of the DLL
    *************************************************************************/
    fx_GetDllVersion(&dwfilever, &dwprodver);
    printf("File Version for FAX DLL is %d.%02d\n",
               HIWORD(dwfilever), LOWORD(dwfilever));
    printf("Product Version for FAX DLL is %d.%02d\n",
               HIWORD(dwprodver), LOWORD(dwprodver));
    /* Now open all the Voice devices */
}

Errors

The $fx_libinit()$ function fails if the library has already been initialized. For example, if you try to make a second call to $sr_libinit()$, it fails.
See Also

- `dx_libinit()`
- `sr_libinit()`
1. System Features

**sr_libinit**

*initializes the Standard Runtime Library DLL*

<table>
<thead>
<tr>
<th>Name:</th>
<th>sr_libinit (flags)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>unsigned short flags</td>
</tr>
<tr>
<td>Returns:</td>
<td>0 if success -1 if failure</td>
</tr>
<tr>
<td>Includes:</td>
<td>srllib.h</td>
</tr>
</tbody>
</table>

**Description**

The **sr_libinit()** function initializes the Standard Runtime Library DLL by loading and resolving all entry points in **LIBSRLMT.DLL**.

This function has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| flags     | This flag has two possible values:  
  DLGC_MT - Specify if using a multi-threaded or window callback model.  
  DLGC_ST - Specify if using the single threaded model. |

**Cautions**

The **sr_libinit()** function must be called prior to using any other technology specific **xx_libinit()** functions.
Example

```c
/*$ sr_libinit( ) example $*/
#include <windows.h>
#include <srllib.h>
int InitDevices( )
{
  DWORD dwfilever, dwprodver;
  /************************************************************************
  * Initialize all the DLLs required. This will cause the DLLs to be
  * loaded and entry points to be resolved. Entry points not resolved
  * are set up to point to a default not implemented function in the
  * `C` library. If the DLL is not found all functions are resolved
  * to not implemented.
  ************************************************************************/
  if (sr_libinit(DLGC_MT) == -1) {
    /* Must be already loaded, only reason if sr_libinit( ) was already called */
  }
  /*************************************************************************
  * SRL library initialized so all other SRL functions may be called as normal.
  * Display the version number of the DLL
  *************************************************************************/
  sr_GetDllVersion(&dwfilever, &dwprodver);
  printf("File Version for SRL is %d.%02d\n",
         HIWORD(dwfilever), LOWORD(dwfilever));
  printf("Product Version for SRL is %d.%02d\n",
         HIWORD(dwprodver), LOWORD(dwprodver));
  /*************************************************************************
  * Call technology specific xx_libinit( ) functions */
  }
}
```

Errors

The `sr_libinit( )` function fails if the library has already been initialized. For example, if you try to make a second call to `sr_libinit( )`, it fails.

See Also

- `dx_libinit( )`
- `fx_libinit( )`
1. System Features

**dt_libinit**

_initializes the Network Library DLL_

<table>
<thead>
<tr>
<th>Name:</th>
<th>dt_libinit (flags)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>unsigned short flags</td>
</tr>
<tr>
<td>Returns:</td>
<td>0 if success</td>
</tr>
<tr>
<td></td>
<td>-1 if failure</td>
</tr>
<tr>
<td>Includes:</td>
<td>srllib.h</td>
</tr>
<tr>
<td></td>
<td>dtlib.h</td>
</tr>
<tr>
<td></td>
<td>msilib.h</td>
</tr>
<tr>
<td></td>
<td>cclib.h</td>
</tr>
</tbody>
</table>

**Description**

The *dt_libinit*() function initializes the Network Library DLL and resolves all entry points in the LIBDTIMT.DLL.

This function has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>This flag has two possible values:</td>
</tr>
<tr>
<td></td>
<td>DLGC_MT - Specify if using a multi-threaded or window callback model.</td>
</tr>
<tr>
<td></td>
<td>DLGC_ST - Specify if using the single threaded model.</td>
</tr>
</tbody>
</table>

**Cautions**

The *sr_libinit*() function must be called prior to using the *dt_libinit*() function.

**Example**

```c
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <dtilib.h>

int InitDevices() {
    // Example code...
} 
```
{ DWORD dwfilever, dwprodver;
  //
  // Initialize all the DLLs required. This will cause the DLLs to be
  // loaded and entry points to be resolved. Entry points not resolved
  // are set up to point to a default not implemented function in the
  // 'C' library. If the DLL is not found all functions are resolved
  // to not implemented.
  if (sr_libinit(DLGC_MT) == -1) {
    // Must be already loaded, only reason if sr_libinit() was
    // already called
  }
  //
  // Call technology specific dt_libinit() functions to load Network DLL
  //
  if (dt_libinit(DLGC_MT) == -1) {
    // Must be already loaded, only reason if dx_libinit() was
    // already called
  }
  //
  // Network library initialised so all other DTI/ISDN/MSI functions may be called
  // as normal. Display the version number of the DLL
  //
  dt_GetDllVersion(&dwfilever, &dwprodver);
  printf("File Version for network DLL is %d.%02d\n",
         HIWORD(dwfilever), LOWORD(dwfilever));
  printf("Product Version for network DLL is %d.%02d\n",
         HIWORD(dwprodver), LOWORD(dwprodver));
  //
  // Now open all the network devices
  //

Errors

The dt_libinit() function fails if the library has already been initialized. For example, if you try to make a second call to dt_libinit(), it fails.

See Also

- dx_libinit()
- fx_libinit()
- vr_libinit()
1. System Features

vr_libinit initializes the Voice Recognition Library DLL

<table>
<thead>
<tr>
<th>Name</th>
<th>vr_libinit (flags)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>unsigned short flags</td>
</tr>
<tr>
<td>Returns</td>
<td>0 if success</td>
</tr>
<tr>
<td></td>
<td>-1 if failure</td>
</tr>
<tr>
<td>Includes</td>
<td>srllib.h</td>
</tr>
<tr>
<td></td>
<td>dxxxlib.h</td>
</tr>
</tbody>
</table>

**Description**

The `vr_libinit()` function initializes the Voice Recognition Library DLL and resolves all entry points in the LIBVRXMT.DLL.

This function has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>This flag has two possible values:</td>
</tr>
<tr>
<td></td>
<td>DLGC_MT - Specify if using a multi-threaded or window callback model.</td>
</tr>
<tr>
<td></td>
<td>DLGC_ST - Specify if using the single threaded model.</td>
</tr>
</tbody>
</table>

**Cautions**

The `sr_libinit()` function must be called prior to using the `vr_libinit()` function.

**Example**

```c
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <vrxxxlib.h>

int InitDevices()
{
    DWORD dwfilever, dwprodver;
```
// Initialize all the DLLs required. This will cause the DLLs to be
// loaded and entry points to be resolved. Entry points not resolved
// are set up to point to a default not implemented function in the
// C library. If the DLL is not found all functions are resolved
// to not implemented.
//
if (sr_libinit(DLGC_MT) == -1) {
    // Must be already loaded, only reason if sr_libinit() was
    // already called
}
//
// Call technology specific dx_libinit() functions to load Voice DLL
//
if (dx_libinit(DLGC_MT) == -1) {
    // Must be already loaded, only reason if dx_libinit() was
    // already called
}
//
// Call technology specific vr_libinit() functions to load VR/160 DLL
//
if (vr_libinit(DLGC_MT) == -1) {
    // Must be already loaded, only reason if vr_libinit() was
    // already called
}
//
// VR/160 library initialized so all other VR/160 functions may be called
// as normal. Display the version number of the DLL

vr_GetDllVersion(&dwfilever, &dwprodver);
printf("File Version for VR/160 DLL is %d.%02d\n",
        HIWORD(dwfilever), LOWORD(dwfilever));
printf("Product Version for VR/160 DLL is %d.%02d\n",
        HIWORD(dwprodver), LOWORD(dwprodver));

//
// Now open all the Voice Recog devices


---

**Errors**

The `vr_libinit()` function fails if the library has already been initialized. For example, if you try to make a second call to `vr_libinit()`, it fails.

**See Also**

- `dt_libinit()`
- `dx_libinit()`
- `fx_libinit()`
1. System Features

1.6. The Wait for Events Function

If you use the `sr_waitevt()` function to retrieve events, we recommend that new applications pass 0 as a parameter when using the following functions:

- `sr_getevtdatap()`
- `sr_getevtdev()`
- `sr_getevtlen()`
- `sr_getevttype()`

1.7. Retrieving Dialogic DLL Version Numbers

Dialogic uses the standard Windows resource file mechanism to version-stamp all Dialogic DLLs. An application can use the standard Win32 `GetFileVersionInfo()` to retrieve the version number. The Dialogic C interface libraries also provide the following Convenience functions for retrieving DLL Version Numbers, which are built on top of Win32 functions.

- `dx_GetDllVersion()` - Returns the Voice DLL Version Number
- `fx_GetDllVersion()` - Returns the Fax DLL Version Number
- `sr_GetDllVersion()` - Returns the Standard Runtime Library DLL Version
- `dt_GetDllVersion()` - Returns the Network DLL Version Number
- `vr_GetDllVersion()` - Returns the Voice Recognition DLL Version Number

1.8. Dialogic DLL Version Number Functions

This section contains a description of the Dialogic DLL Version Number functions. Each of these functions returns the file version number and product version number. The file version number specifies the version of the DLL. The product version number specifies the version of the software release that includes the DLL. Each function returns both version numbers in hexadecimal format. For example, if the DLL version is 4.13, the function returns it as 0x0004000D. If the product version is 11.3, the function returns it as 0x000b0003. In each case, the high word represents the major number, and the low word represents the minor number.
dx_GetDllVersion returns the Voice DLL Version Number

Name: dx_GetDllVersion (dwfileverp, dwprodverp)

Inputs:
- LPDWORD dwfileverp • Voice DLL Version Number
- LPDWORD dwprodverp • Product version of this release

Returns:
0 if success
-1 if failure

Includes:
dxxxlib.h
srllib.h

dx_GetDllVersion

description

The dx_GetDllVersion( ) function returns the Voice DLL Version Number for the file and product.

This function has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwfileverp</td>
<td>pointer to where to return file version information</td>
</tr>
<tr>
<td>dwprodverp</td>
<td>pointer to where to return product version information</td>
</tr>
</tbody>
</table>

cautions

None.

dexample

/*$ dx_GetDllVersion( ) example $*/
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>

int InitDevices( )
{
    DWORD dwfilever, dwprodver;

    /*-----------------------------------------------*/

1. System Features

* Initialize all the DLLs required. This will cause the DLLs to be
  loaded and entry points to be resolved. Entry points not resolved
  are set up to point to a default not implemented function in the
  "C" library. If the DLL is not found all functions are resolved
  to not implemented.

************************************************************************/

if (sr_libinit(DLGC_MT) == -1) {
    /* Must be already loaded, only reason if sr_libinit( ) was already called */
}
/* Call technology specific dx_libinit( ) functions to load Voice DLL */
if (dx_libinit(DLGC_MT) == -1) {
    /* Must be already loaded, only reason if dx_libinit( ) was already called */
}
/*************************************************************************/

* Voice library initialized so all other Voice functions may be called
  as normal. Display the version number of the DLL

************************************************************************不多

dx_GetDllVersion(&dwfilever, &dwprodver);
printf("File Version for Voice DLL is %d.%02d\n",
    HIWORD(dwfilever), LOWORD(dwfilever));
printf("Product Version for Voice DLL is %d.%02d\n",
    HIWORD(dwprodver), LOWORD(dwprodver));

/* Now open all the Voice devices */
}

■ Errors

None.

■ See Also

- fx_GetDIIVersion()
- sr_GetDIIVersion()
- dt_GetDIIVersion()
- vt_GetDIIVersion()
fx_GetDllVersion returns the Fax DLL Version Number

Name: fx_GetDllVersion (dwfileverp, dwprodverp)

Inputs:
- LPDWORD dwfileverp • Fax DLL Version Number
- LPDWORD dwprodverp • Product version of this release

Returns:
0 if success
-1 if failure

Includes:
srllib.h
dxxlib.h
taxlib.h

Description
The fx_GetDllVersion() function returns the Fax DLL Version Number for the file and product.

This function has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwfileverp</td>
<td>pointer to where to return file version information</td>
</tr>
<tr>
<td>dwprodverp</td>
<td>pointer to where to return product version information</td>
</tr>
</tbody>
</table>

Cautions
None.
1. System Features

Example

```c
/*$ fx_GetDllVersion() example $*/
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int InitDevices() {
    DWORD dwfilever, dwprodver;

    /************************************************************************
    * Initialize all the DLLs required. This will cause the DLLs to be
    * loaded and entry points to be resolved. Entry points not resolved
    * are set up to point to a default not implemented function in the
    * "C" library. If the DLL is not found all functions are resolved
    * to not implemented.
    ************************************************************************/

    if (sr_libinit(DLGC_MT) == -1) {
        /* Must be already loaded, only reason if sr_libinit() was already called */
    }
    /* Call technology specific dx_libinit( ) functions to load Voice DLL */
    if (dx_libinit(DLGC_MT) == -1) {
        /* Must be already loaded, only reason if dx_libinit() was already called */
    }
    /* Call technology specific fx_libinit( ) functions to load VFX Fax DLL */
    if (fx_libinit(DLGC_MT) == -1) {
        /* Must be already loaded, only reason if dx_libinit() was already called */
    }

    /*************************************************************************
    * Fax library initialized so all other VFX functions may be called as normal.
    * Display the version number of the DLL
    *************************************************************************/
    fx_GetDllVersion(&dwfilever, &dwprodver);
    printf("File Version for FAX DLL is %d.%02d\n", HIWORD(dwfilever), LOWORD(dwfilever));
    printf("Product Version for FAX DLL is %d.%02d\n", HIWORD(dwprodver), LOWORD(dwprodver));

    /* Now open all the Voice devices */
}
```

Errors

None.
See Also

- dx_GetDllVersion()
- sr_GetDllVersion()
- dt_GetDllVersion()
- vr_GetDllVersion()
## sr_GetDllVersion

**Name:** sr_GetDllVersion (dwfileverp, dwprodverp)

**Inputs:**
- LPDWORD dwfileverp
- LPDWORD dwprodverp

**Returns:**
- 0 if success
- -1 if failure

**Includes:**
- srllib.h

### Description

The `sr_GetDllVersion()` function returns the Standard Runtime Library DLL Version Number for the file and product.

This function has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwfileverp</td>
<td>pointer to where to return file version information</td>
</tr>
<tr>
<td>dwprodverp</td>
<td>pointer to where to return product version information</td>
</tr>
</tbody>
</table>

### Cautions

None.
Dialogic System Software and SDK for Windows NT DNA Release Reference

Example

/*$ sr_GetDllVersion() example */
#include <windows.h>
#include <srllib.h>

int InitDevices()
{
    DWORD dwfilever, dwprodver;

    /**************************************************************************
    * Initialize all the DLLs required. This will cause the DLLs to be
    * loaded and entry points to be resolved. Entry points not resolved
    * are set up to point to a default not implemented function in the
    * "C" library. If the DLL is not found all functions are resolved
    * to not implemented.
    **************************************************************************/

    if (sr_libinit(DLGC_MT) == -1) {
        /* Must be already loaded, only reason if sr_libinit() was already called */
    }

    /**************************************************************************
    * SRL library initialized so all other SRL functions may be called as normal.
    * Display the version number of the DLL
    **************************************************************************/

    printf("File Version for SRL is %d.%02d\n",
          HIWORD(dwfilever), LOWORD(dwfilever));
    printf("Product Version for SRL is %d.%02d\n",
          HIWORD(dwprodver), LOWORD(dwprodver));

    /**************************************************************************
    * Call technology specific xx_libinit() functions */

}

Errors

None.

See Also

- dx_GetDllVersion()
- fx_GetDllVersion()
- dt_GetDllVersion()
- vr_GetDllVersion()
1. System Features

**dt_GetDllVersion**  
*returns the Network DLL Version Number*

<table>
<thead>
<tr>
<th>Name:</th>
<th>dt_GetDllVersion (dwfileverp, dwprodverp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>LPDWORD dwfileverp</td>
</tr>
<tr>
<td></td>
<td>LPDWORD dwprodverp</td>
</tr>
<tr>
<td></td>
<td>• Network DLL Version Number</td>
</tr>
<tr>
<td></td>
<td>• Product version of this release</td>
</tr>
<tr>
<td>Returns:</td>
<td>0 if success</td>
</tr>
<tr>
<td></td>
<td>-1 if failure</td>
</tr>
<tr>
<td>Includes:</td>
<td>srllib.h</td>
</tr>
<tr>
<td></td>
<td>dxxxlib.h</td>
</tr>
<tr>
<td></td>
<td>dtlib.h</td>
</tr>
</tbody>
</table>

### Description

The **dt_GetDllVersion()** function returns the Network DLL Version Number for the file and product.

This function has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwfileverp</td>
<td>pointer to where to return file version information</td>
</tr>
<tr>
<td>dwprodverp</td>
<td>pointer to where to return product version information</td>
</tr>
</tbody>
</table>

### Cautions

If using older DLL’s with no version number stamps, a default version 4.10 (0x0004000A) is returned.

### Example

```c
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <dtlib.h>

int InitDevices()
{
    DWORD dwfilever, dwprodver;
```
// Initialize all the DLLs required. This will cause the DLLs to be
// loaded and entry points to be resolved. Entry points not resolved
// are set up to point to a default not implemented function in the
// 'C' library. If the DLL is not found all functions are resolved
// to not implemented.

if (sr_libinit(DLGC_MT) == -1) {
    // Must be already loaded, only reason if sr_libinit() was
    // already called
}

// Call technology specific dt_libinit() functions to load Network DLL
if (dt_libinit(DLGC_MT) == -1) {
    // Must be already loaded, only reason if dt_libinit() was
    // already called
}

// Network library initialised so all other DTI/ISDN/MSI functions may be called
// as normal. Display the version number of the DLL

// Print file version
dt_GetDllVersion(&dwfilever, &dwprodver);
printf("File Version for network DLL is %d.%02d\n",
        HIWORD(dwfilever), LOWORD(dwfilever));

// Print product version
printf("Product Version for network DLL is %d.%02d\n",
        HIWORD(dwprodver), LOWORD(dwprodver));

// Now open all the network devices

---

**Errors**

None.

**See Also**

- `dx_GetDllVersion()`
- `fx_GetDllVersion()`
- `sr_GetDllVersion()`
- `vr_GetDllVersion()`
1. System Features

vr_GetDllVersion returns the Voice Recognition DLL Version Number

**Name:** vr_GetDllVersion (dwfileverp, dwprodverp)

**Inputs:**
- LPDWORD dwfileverp • Voice Recognition DLL Version Number
- LPDWORD dwprodverp • Product version of this release

**Returns:**
- 0 if success
- -1 if failure

**Includes:**
srlib.h
dxxxlib.h
vrxxxlib.h

---

**Description**

The vr_GetDllVersion() function returns the Voice Recognition DLL Version Number for the file and product.

This function has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwfileverp</td>
<td>pointer to where to return file version information</td>
</tr>
<tr>
<td>dwprodverp</td>
<td>pointer to where to return product version information</td>
</tr>
</tbody>
</table>

**Cautions**

If using older DLL’s with no version number stamps, a default version 4.10 (0x0004000A) is returned.
Example

```c
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <vrxxxlib.h>

int InitDevices()
{
    DWORD dwfilever, dwprodver;
    
    // Initialize all the DLLs required. This will cause the DLLs to be
    // loaded and entry points to be resolved. Entry points not resolved
    // are set up to point to a default not implemented function in the
    // 'C' library. If the DLL is not found all functions are resolved
    // to not implemented.
    //
    if (sr_libinit(DLGC_MT) == -1) {
        // Must be already loaded, only reason if sr_libinit() was
        // already called
    }
    
    // Call technology specific dx_libinit() functions to load Voice DLL
    //
    if (dx_libinit(DLGC_MT) == -1) {
        // Must be already loaded, only reason if dx_libinit() was
        // already called
    }
    
    // Call technology specific vr_libinit() functions to load VR/160 DLL
    //
    if (vr_libinit(DLGC_MT) == -1) {
        // Must be already loaded, only reason if dx_libinit() was
        // already called
    }
    
    // VR/160 library initialised so all other VR/160 functions may be called
    // as normal. Display the version number of the DLL
    //
    vr_GetDllVersion(&dwfilever, &dwprodver);
    printf("File Version for VR/160 DLL is %d.%02d\n",
            HIWORD(dwfilever), LOWORD(dwfilever));

    printf("Product Version for VR/160 DLL is %d.%02d\n",
            HIWORD(dwprodver), LOWORD(dwprodver));

    //
    // Now open all the Voice Recog devices
    //
}
```
1. System Features

■ Errors

None.

■ See Also

- dt_GetDllVersion()
- dx_GetDllVersion()
- fx_GetDllVersion()
- sr_GetDllVersion()
2. Voice

2.1. Play and Record Functions

Play and record functions included in the Voice Library are listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dx_playiottdata()</td>
<td>plays voice data from multiple sources</td>
</tr>
<tr>
<td>dx_playvox()</td>
<td>plays a single vox file</td>
</tr>
<tr>
<td>dx_playwav()</td>
<td>plays a single wave file</td>
</tr>
<tr>
<td>dx_recioottdata()</td>
<td>records voice data to multiple destinations</td>
</tr>
<tr>
<td>dx_mrecioottdata()</td>
<td>records voice data from two channels to a single file, device or memory. The dx_mrecioottdata() function supports the Transaction Record feature as described in Section 2.6. Transaction Record.</td>
</tr>
<tr>
<td>dx_recvox()</td>
<td>records voice data to a single vox file</td>
</tr>
<tr>
<td>dx_recwav()</td>
<td>records voice data to a single wave file</td>
</tr>
</tbody>
</table>

For a detailed description of the wave file functions, refer to the Voice Software Reference for Windows NT.

2.2. Voice Features

2.2.1. Call Progress Features

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

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

The recommended setting for the Call Analysis Parameter structure (DX_CAP) ca_pamd_spdval field is PAMD_ACCU. Previously, the ca_pamd_spdval field could be set to PAMD_FULL or PAMD_QUICK. This field can now also be set to PAMD_ACCU, which does the most accurate evaluation; it detects live voice as accurately as PAMD_FULL but is more accurate than PAMD_FULL (although slightly slower) in detecting an answering machine. Use the setting PAMD_ACCU when accuracy is more important than speed.

2.2.2. Tone Features

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

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

Dial Pulse Detection

DTMF Length and Interdigit Timing

New channel parameters are available for setting DTMF length and interdigit delay for dialing. The software reference currently documents these parameters for use only at the board level. These new parameters can be set with the dx_setparm() function by specifying one of the following values for parm. To set the DTMF length, set parm = DXCH_TTDATA. To set the DTMF interdigit delay, set parm = DXCH_T_IDD. Specify the duration using the valuep parameter in units of 10 ms.

Dial Pulse Detection (DPD) Digit Type Reporting

As shown in the following table, two new defines have been provided for identifying the Dial Pulse Detection digit type, depending upon where the digit type is retrieved.
2. Voice

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

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

**Other Defines for Digit Type Reporting**

Several new defines have been added for obtaining the DV_DIGIT dg_type (digit type) from the digit buffer. These defines contain “_ASCII” extensions as shown in the following table.

<table>
<thead>
<tr>
<th>Digit Type</th>
<th>Defines for dg_type from</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Digit Buffer</td>
</tr>
<tr>
<td>DTMF</td>
<td>DG_DTMF_ASCII</td>
</tr>
<tr>
<td>DPD</td>
<td>DG_DPD_ASCII</td>
</tr>
<tr>
<td>MF</td>
<td>DG_MF_ASCII</td>
</tr>
<tr>
<td>GTD (user defined)</td>
<td>DG_USER1_ASCII</td>
</tr>
<tr>
<td></td>
<td>DG_USER2_ASCII</td>
</tr>
<tr>
<td></td>
<td>DG_USER3_ASCII</td>
</tr>
<tr>
<td></td>
<td>DG_USER4_ASCII</td>
</tr>
<tr>
<td></td>
<td>DG_USER5_ASCII</td>
</tr>
</tbody>
</table>

Use the above defines to identify the digit type from the value returned. If you get the digit from the DV_DIGIT dg_type digit buffer using dx_getdig(), you should use the digit type define that has the “_ASCII” extension. Otherwise, if you get the digit from the DX_EBLK event queue data (cst_data) through a
DE_DIGITS Call Status Transition event, you should use the digit type define without the “ASCII” extension.

2.3. Caller ID

2.3.1. Caller ID Overview

Caller Identification (Caller ID) is the Bellcore ServiceMark for a feature that allows local telephone companies to provide a service enabling the subscriber to receive the caller’s phone number (Directory Number), possibly the name of the caller, and other information about the call. A similar service is available in other countries. This feature will support all countries who use the Bellcore CLASS specification, and where the product is approved for connection (US, Canada and Singapore at this time). The Caller ID information is transmitted using Frequency Shift Keying (FSK) to the subscriber from the service provider, the telephone company Central Office (CO), at 1200 baud.

2.3.2. Caller ID Formats

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

Caller ID formats currently supported are:

- Custom Local Area Signaling Services (CLASS) is a standard published by Bellcore:
  - Single Data Message (SDM) format
  - Multiple Data Message (MDM) format
- Analog Calling Line Identity Presentation (ACLIP) is a standard used in Singapore published by the Telecommunications Authority of Singapore:
  - Single Data Message Format (SDMF)
  - Multiple Data Message Format (MDMF)
- Calling Line Identity Presentation (CLIP) is a standard used in the United Kingdom published by British Telecommunications (BT)
Caller ID information is received from the Central Office between the first and second ring for CLASS and ACLIP, and before the first ring for CLIP.

CLASS is the only format used by applicable high density boards.

**NOTES:**
1. Throughout the discussion of Caller ID in this Release Reference, ACLIP SDMF is referred to as SDM, and ACLIP MDMF is referred to as MDM.
2. ACLIP and CLIP are not supported on the D/160SC-LS, LSI/81SC or LSI/161SC boards.

CLASS and ACLIP Caller ID information is supported as sent by the service provider in the following format types:

- **Single Data Message (SDM) format Caller ID information:**
  - Frame header (indicating SDM format type)
  - Calling line’s Directory Number (DN)
  - Date
  - Time

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

**NOTE:** One or more of the Caller ID features may be available from your service provider. Contact your service provider to determine the Caller ID options available from your Central Office.

CLIP Caller ID information is supported as sent by the service provider in the following format type:
Dialogic System Software and SDK for Windows NT DNA Release Reference

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

**NOTE:** One or more of the above Caller ID features may be available from your service provider. Contact your service provider to determine Caller ID options available from your Central Office.

### 2.3.3. Accessing Caller ID Information

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

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

- For CLIP, the Caller ID information is received from the service provider before the first ring. Set the ring event in the application to occur on or after the first ring. The ring event indicates reception of the CLIP Caller ID information from the Central Office.

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

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

Voice API functions and parameters for Caller ID are described in detail later in this chapter.

To determine if Caller ID information has been received from the Central Office (CO), before issuing a `dx_gtcallid()` or `dx_gtextcallid()` Caller ID function, check the event data in the event block. When the ring event is received, the event data field in the event block is bitmapped and indicates that Caller ID information is available when bit 0 (LSB) is set to 1; see the function code examples in this document. For details on the event block, refer to the Voice Software Reference.

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

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

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

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

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

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

- `dx_setevtmsk()` voice function
- `dx_getevt()` voice function
• dx_gtcallid( ) Caller ID function

2.3.4. Error Handling

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

• If the Caller ID function completes successfully, the buffer contains the Caller ID information.

• If the Caller ID function completes unsuccessfully, an error code is returned that indicates the reason for the error.

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

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

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

2.3.5. Enabling Channels to Use the Caller ID Feature

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

Caller ID parameter for dx_setparm():

38
2. Voice

### 2.3. Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| DXCH_CALLID      | Default: DX_CALLIDDISABLE  
Enable or disable Caller ID for the channel as specified in **dx_setparm()**.  
Valid values are:  
DX_CALLIDDISABLE  
DX_CALLIDENABLE    |

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

#### 2.3.6. Caller ID Demonstration Programs

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

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

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

#### 2.4. Caller ID Functions

This section describes the following Caller ID functions:

- `dx_gtcallid()` Returns the calling line Directory Number
- `dx_gtextcallid()` Returns the requested Caller ID message
- `dx_wtcallid()` Waits for rings and reports Caller ID
dx_gtcallid  returns the calling line Directory Number

<table>
<thead>
<tr>
<th>Name</th>
<th>int dx_gtcallid (chdev, bufferp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>int chdev</td>
</tr>
<tr>
<td></td>
<td>unsigned char</td>
</tr>
<tr>
<td></td>
<td>*bufferp</td>
</tr>
<tr>
<td>Returns</td>
<td>0 success</td>
</tr>
<tr>
<td></td>
<td>-1 error return code</td>
</tr>
<tr>
<td>Includes</td>
<td>srllib.h</td>
</tr>
<tr>
<td></td>
<td>dxxxlib.h</td>
</tr>
<tr>
<td>Category</td>
<td>Caller ID</td>
</tr>
<tr>
<td>Mode</td>
<td>synchronous</td>
</tr>
</tbody>
</table>

Description

The dx_gtcallid( ) function returns the calling line Directory Number (DN) sent by the Central Office.

This function has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chdev:</td>
<td>Channel device handle.</td>
</tr>
<tr>
<td>Bufferp:</td>
<td>Pointer to where to return calling line Directory Number (DN).</td>
</tr>
</tbody>
</table>

On successful completion, a NULL terminated string containing the caller’s phone number (DN) is placed in the buffer.

**NOTE:** Non-numeric characters (punctuation, space, dash) may be included in the number string. The string may not be suitable for dialing without modification.
2. Voice

Caller ID information is available for the call from the moment the ring event is generated (if the ring event is set to occur on or after the second ring (CLASS, ACLIP) or set to occur on or after the first ring (CLIP)) until either of the following occurs:

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

## Cautions

To allow the reception of Caller ID information from the Central Office before answering a call (application channel goes off-hook):

- In CLASS and ACLIP, set the ring event to occur on or after the second ring.
- In CLIP, set the ring event to occur on or after the first ring.

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

## Example

```c
/* dx_gtcallid() example */

#include <windows.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
/* Dialogic Includes */
#include "srllib.h"
#include "dxxxlib.h"

int main()
{
    int numRings = 2; /* In the US */
    int ringtimeout = 20; /* 20 seconds */
    int chdev; /* Channel descriptor */
    unsigned short parmval;
    unsigned char buffer[81];
```
/ Open channel /
if ((chdev=dx_open("dxxxB1C1", NULL)) == -1) {
    /* process error */
    exit(0);
}

/* Enable the caller id functionality */
parmval = DX_CALLIDENABLE;
if (dx_setparm(chdev, DXCH_CALLID, (void *)&parmval) == -1) {
    /* process error */
    exit(0);
}

/******************************************************************
* Set the number of rings required for a RING event to permit
* receipt of the caller id information. In the US, caller id
* information is transmitted between the first and second rings
******************************************************************/
parmval = numRings; /* 2 in the US */
if (dx_setparm(chdev, DXCH_RINGCNT, &parmval) == -1) {
    /* process error */
    exit(0);
}

/* Put the channel onhook */
if (dx_sethook(chdev, DX_ONHOOK, EV_SYNC) == -1) {
    /* process error */
    exit(0);
}

/* Wait for 2 rings and go offhook (timeout after 20 seconds) */
if (dx_wtring(chdev, numRings, DX_OFFHOOK, ringTimeout) == -1) {
    /* process error */
}

/* Get just the caller id */
if (dx_getcallid(chdev, buffer) == -1) {
    /* Can check the specific error code */
    if (ATDV_LASTERR(chdev) == EDX_CLIDBLK) {
        printf("Caller ID information blocked \n\n");
    } else if (ATDV_LASTERR(chdev) == EDX_CLIDOOGA) {
        printf("Caller out of area \n\n");
    } else {
        /* Or print the pre-formatted error message */
        printf("Error: %s \n", ATDV_ERRMSGP(chdev));
    }
} else {
    printf("Caller ID = %s\n", buffer);
}

/******************************************************************
* If the message is an MDM (Multiple Data Message), then
* additional information is available.
* First get the frame and check the frame type. If Class MDM,
* get and print additional information from submessages.
*******************************************************************/
if (dx_gtextcallid(chdev, CLIDINFO_FRAMETYPE, buffer) != -1) {
2. Voice

if(buffer[0] == CLASSFRAME_MDM) {
    /* Get and print the date and time */
    if (dx_gtextcallid(chdev, MCLASS_DATETIME) == -1) {
        /* process error */
        printf("Error: %s\n", ATDV_ERRMSGP(chdev));
    } else {
        printf("Date/Time = %s\n", buffer);
    }

    /* Get and print the caller name */
    if (dx_gtextcallid(chdev, MCLASS_NAME) == -1) {
        /* process error */
        printf("Error: %s\n", ATDV_ERRMSGP(chdev));
    } else {
        printf("Caller Name = %s\n", buffer);
    }

    /* Get and print the Dialed Number */
    if (dx_gtextcallid(chdev, MCLASS_DDN) == -1) {
        /* process error */
        printf("Error: %s\n", ATDV_ERRMSGP(chdev));
    } else {
        printf("Dialed Number = %s\n", buffer);
    }
}
else {
    printf("Submessages not available - not an MDM message\n");
}
}

dx_close(chdev);
return(0);
Errors

If this function returns -1 to indicate failure, use \texttt{ATDV\_LASTERR()} and \texttt{ATDV\_ERRMSGP()} to retrieve one of the following error reasons:

- EDX\_BADPARM: Invalid parameter
- EDX\_BUSY: Channel is busy
- EDX\_CLIDBLK: Caller ID is blocked or private or withheld (other information may be available using dx\_gtextcallid( ))
- EDX\_CLIDINFO: Caller ID information not sent or Caller ID information invalid
- EDX\_CLIDOOA: Caller ID is out of area (other information may be available using dx\_gtextcallid( ))
- EDX\_SYSTEM: Operating system error - check errno

See Also

- dx\_gtextcallid( )
- dx\_wtcallid( )
2. Voice

dx_gtextcallid  returns the requested Caller ID message

**Name:** int dx_gtextcallid (chdev, infotype, bufferp)

**Inputs:**
- int chdev  • Channel device handle
- int infotype  • Message Type ID
- unsigned char *bufferp  • Pointer to where to return the requested Caller ID message

**Returns:**
- 0 success
- -1 error return code

**Includes:**
- srllib.h
- dxxlib.h

**Category:** Caller ID

**Mode:** synchronous

---

**Description**

The `dx_gtextcallid()` function returns the requested Caller ID message by specifying the Message Type ID. The application can issue this function as many times as required to get the desired Caller ID messages (such as date and time, calling line subscriber name, reason why Caller ID is not available). The formatting and content of the Caller ID messages documented in this Release Reference are based on the published telecommunication standards. The actual formatting and content of the data returned depend on the implementation and level of service provided by the originating and destination Central Offices.

**NOTE:** For CLASS and ACLIP, do not use Multiple Data Message Type IDs with Caller ID information in Single Data Message format.

This function has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chdev:</td>
<td>Channel device handle.</td>
</tr>
<tr>
<td>Infotype:</td>
<td>The Message Type ID for the specific Caller ID information to receive. (Message Type IDs for CLASS, ACLIP and CLIP are listed on the following pages.)</td>
</tr>
<tr>
<td>bufferp:</td>
<td>Pointer to where to return the requested Caller ID message.</td>
</tr>
</tbody>
</table>
**Parameter Description**

All returns are NULL terminated.

**Common Message Types**

The following standard Message Types are available for:

- CLASS (Single Data Message)
- CLASS (Multiple Data Message)
- ACLIP (Single Data Message)
- ACLIP (Multiple Data Message)
- CLIP

All returns are NULL terminated.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition/Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIDINFO_CMPLT</td>
<td>All Caller ID information as sent from the CO (maximum of 258 bytes; includes header and length byte at the beginning). Can produce EDX_CLIDINFO error.</td>
</tr>
<tr>
<td>CLIDINFO_GENERAL</td>
<td>Date and time (20 bytes - formatted with / and : characters; padded with spaces). Caller phone number or reason for absence (20 bytes; padded with spaces). Caller name or reason for absence (variable length ≥0; not padded). Can produce EDX_CLIDINFO error. See Figure 1. Format of General Caller ID Information.</td>
</tr>
<tr>
<td>CLIDINFO_CALLID</td>
<td>Caller ID (phone number); can produce EDX_CLIDINFO, EDX_CLIDOAA, and EDX_CLIDBLK errors.</td>
</tr>
<tr>
<td>CLIDINFO_FRAMETYPE</td>
<td>Indicates Caller ID frame (does not apply to CLIP). Values (depending upon service type): CLASSFRAME_SDM, CLASSFRAME_MDM, ACLIPFRAME_SDM</td>
</tr>
</tbody>
</table>
2. Voice

ACLIPFRAME_MDM
Can produce EDX_CLIDINFO error

<table>
<thead>
<tr>
<th>Date and Time (20 bytes)</th>
<th>Phone Number (20 bytes)</th>
<th>Name (variable length&gt;0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0123456789012345678901234567890123456789012345678901234567890123456789</td>
<td>04/04 10:11bbbbbbbbbbbbb</td>
<td>2019933000bbbbbbbbbb</td>
</tr>
<tr>
<td>04/04 10:11bbbbbbbbbbbbb</td>
<td>2019933000bbbbbbbbbb</td>
<td>P</td>
</tr>
<tr>
<td>04/04 10:11bbbbbbbbbbbbb</td>
<td>P</td>
<td>O</td>
</tr>
<tr>
<td>04/04 10:11bbbbbbbbbbbbb</td>
<td>O</td>
<td>I</td>
</tr>
</tbody>
</table>

b=blank  O=null  O=Out of area  P=Private

Figure 1. Format of General Caller ID Information

Message Types for CLASS (Multiple Data Message)
See Common Message Types for the standard Message Types that can also be used. The following Message Types can produce an EDX_CLIDINFO error. All returns are NULL terminated.
**Value** | **Definition/Returns**
---|---
MCLASS_DATETIME | Date and Time (as sent by CO without format characters / and :)
MCLASS_DN | Calling line directory number (digits only)
MCLASS_DDN | Dialed number (digits only)
MCLASS_ABSENCE1 | Reason for absence of Caller ID (only available if caller name is absent): O = out of area, P = private
MCLASS_REDIRECT | Call forward: 0 = universal; 1 = busy; 2 = unanswered
MCLASS_QUALIFIER | L = long distance call
MCLASS_NAME | Calling line subscriber name
MCLASS_ABSENCE2 | Reason for absence of name (only available if caller name is absent): O = out of area, P = private

**Message Types for ACLIP (Multiple Data Message)**

See *Common Message Types* for the standard Message Types that can also be used. The following Message Types can produce an EDX_CLIDINFO error. All returns are NULL terminated.
## 2. Voice

### Value | Definition/Returns
---|---
MACLIP_DATETIME | Date and Time (as sent by CO without format characters / and :)
MACLIP_DN | Calling line directory number (digits only)
MACLIP_DDN | Dialed number (digits only)
MACLIP_ABSENCE1 | Reason for absence of Caller ID (only available if caller name is absent): O = out of area, P = private
MACLIP_REDIRECT | Call forward: 0 = universal; 1 = busy; 2 = unanswered
MACLIP_QUALIFIER | L = long distance call
MACLIP_NAME | Calling line subscriber name
MACLIP_ABSENCE2 | Reason for absence of name (only available if caller name is absent): O = out of area, P = private

### Message Types for CLIP

See *Common Message Types* for the standard Message Types that can also be used. The following Message Types can produce an EDX_CLIDINFO error. All returns are NULL terminated.
### Definition/Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIP_DATETIME</td>
<td>Date and Time (as sent by CO without format characters / and :)</td>
</tr>
<tr>
<td>CLIP_DN</td>
<td>Calling line directory number (digits only)</td>
</tr>
<tr>
<td>CLIP_DDN</td>
<td>Dialed number (digits only)</td>
</tr>
<tr>
<td>CLIP_ABSENCE1</td>
<td>Reason for absence of Caller ID (only available if caller name is absent): O = out of area, P = private</td>
</tr>
<tr>
<td>CLIP_NAME</td>
<td>Calling line subscriber name</td>
</tr>
<tr>
<td>CLIP_ABSENCE2</td>
<td>Reason for absence of name (only available if caller name is absent): O = out of area, P = private</td>
</tr>
<tr>
<td>CLIP_CALLTYPE</td>
<td>1 = voice call, 2 = ring back when free call, 129 = message waiting call</td>
</tr>
<tr>
<td>CLIP_NETMSG</td>
<td>Network Message System status: number of messages waiting</td>
</tr>
</tbody>
</table>

By passing the proper Message Type ID, the `dx_gtextcallid()` function can be used to retrieve the desired message(s). For example:

- CLIDINFO_CMPLT can be used to get the complete Caller ID frame including header, length, sub-message(s) as sent by the CO
- CLIDINFO_GENERAL can be used to get messages including date and time (formatted), caller’s Directory Number (DN), and name
- CLIDINFO_CALLID can be used to get caller’s Directory Number (DN)
- CLIDINFO_FRAMETYPE can be used to determine the type of Caller ID frame (for example: CLASS SDM or CLASS MDM, ACLIP SDM or ACLIP MDM)
- MCLASS_DDN can be used to get the dialed number for CLASS MDM (digits only)
- MACLASS_DDN can be used to get the dialed number for ACLIP MDM (digits only)
- CLIP_NAME can be used to get the calling line subscriber name for CLIP
• MACLIP_NAME can be used to get the calling line subscriber name for ACLIP

Caller ID information is available for the call from the moment the ring event is generated (if the ring event is set to occur on or after the second ring (CLASS, ACLIP) or set to occur on or after the first ring (CLIP)) until either of the following occurs:

• If the call is answered (the application channel goes off-hook), the Caller ID information is available to the application until the call is disconnected (the application channel goes on-hook).

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

■ Cautions

To allow the reception of Caller ID information from the central office before answering a call (application channel goes off-hook):

• For CLASS and ACLIP, set the ring event to occur on or after the second ring.

• For CLIP, set the ring event to occur on or after the first ring.

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

CLASS and ACLIP: Do not use Multiple Data Message Type IDs with Caller ID information in Single Data Message format.

Make sure the buffer size is large enough to hold the Caller ID message(s) returned by this function.
Example

```c
/* dx_gtextcallid() example to obtain all available Caller ID information */

#include <windows.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

/* Dialogic Includes */
#include "srllib.h"
#include "dxxxlib.h"

int main()
{
    int numRings = 2; /* In the US */
    int ringTimeout = 20; /* 20 seconds */
    int chdev; /* Channel descriptor */
    unsigned short parmval;
    unsigned char buffer[81];

    /* Open channel */
    if ((chdev=dx_open("dxxxB1C1", NULL)) == -1) {
        /* process error */
        exit(0);
    }

    /* Enable the caller id functionality */
    parmval = DX_CALLIDEnable;
    if (dx_setparm(chdev, DXCH_CALLID, (void *) &parmval) == -1) {
        /* process error */
        exit(0);
    }

    /******************************************************************
    * Set the number of rings required for a RING event to permit
    * receipt of the caller id information. In the US, caller id
    * information is transmitted between the first and second rings
    ******************************************************************/
    parmval = numRings; /* 2 in the US */
    if (dx_setparm(chdev, DXCH_RINGCNT, &parmval) == -1) {
        /* process error */
        exit(0);
    }

    /* Put the channel onhook */
    if (dx_sethook(chdev, DX_ONHOOK, EV_SYNC) == -1) {
        /* process error */
        exit(0);
    }

    /******************************************************************
    * Wait for 2 rings and go offhook (timeout after 20 seconds) */
    if (dx_wtring(chdev, numRings, DX_OFFHOOK, ringTimeout) == -1) {
        /* process error */
    }

    return 0;
}
```
2. Voice

if (dx_gtextcallid(chdev, CLIDINFO_FRAMETYPE, buffer) != -1) {
    if (buffer[0] == CLASSFRAME_MDM) {
        /* Get and print the Caller ID */
        if (dx_gtextcallid(chdev, MCLASS_DN, buffer) != -1) {
            printf("Caller ID = %s\n", buffer);
        }
        /* This is another way to obtain Caller ID (regardless of frame type)*/
        else if (dx_gtextcallid(chdev, CLIDINFO_CALLID, buffer) != -1) {
            printf("Caller ID = %s\n", buffer);
        }
        else {
            /* print the reason for the Absence of Caller ID */
            printf("Caller ID not available: %s\n", ATDV_ERRMSGP(chdev));
        }
    }
    /* Get and print the Caller Name */
    if (dx_gtextcallid(chdev, MCLASS_NAME, buffer) != -1) {
        printf("Caller Name = %s\n", buffer);
    }
    /* Get and print the Date and Time */
    if (dx_gtextcallid(chdev, MCLASS_DATETIME, buffer) != -1) {
        printf("Date/Time = %s\n", buffer);
    }
    /* Get and print the Dialed Number */
    if (dx_gtextcallid(chdev, MCLASS_DDN) != -1) {
        printf("Dialed Number = %s\n", buffer);
    }
    else {
        printf("Submessages not available - not an MDM message\n");
        /* Get just the caller id */
        if (dx_gtextcallid(chdev, CLIDINFO_CALLID, buffer) != -1) {
            printf("Caller ID = %s\n", buffer);
        }
        else {
            /* print the reason for the absence of caller id */
            printf("Caller ID not available: %s\n", ATDV_ERRMSGP(chdev));
        }
    }
}

if (dx_gtextcallid(chdev, CLIDINFO_GENERAL, buffer) != -1) {
    printf("Date/Time, Caller Number, and Caller ID = %s\n", buffer);
}

else {
    /* Print out the error message */
    printf("Error: %s\n", ATDV_ERRMSGP(chdev));
}

}
Errors

If this function returns -1 to indicate failure, use ATDV_LASTERR( ) and ATDV_ERRMSGP( ) to retrieve one of the following error reasons:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDX_BADPARM</td>
<td>Invalid parameter</td>
</tr>
<tr>
<td>EDX_BUSY</td>
<td>Channel is busy</td>
</tr>
<tr>
<td>EDX_CLIDBLK</td>
<td>Caller ID is blocked or private or withheld</td>
</tr>
<tr>
<td></td>
<td>(infotype = CLIDINFO_CALLID)</td>
</tr>
<tr>
<td>EDX_CLIDINFO</td>
<td>Caller ID information not sent, sub-message(s)</td>
</tr>
<tr>
<td></td>
<td>requested not available or Caller ID information invalid</td>
</tr>
<tr>
<td>EDX_CLIDOAA</td>
<td>Caller ID is out of area (infotype =</td>
</tr>
<tr>
<td></td>
<td>CLIDINFO_CALLID)</td>
</tr>
<tr>
<td>EDX_SYSTEM</td>
<td>Operating system error - check errno</td>
</tr>
</tbody>
</table>

All Message Types (infotype) can produce an EDX_CLIDINFO error. Message Type CLIDINFO_CALLID can also produce EDX_CLIDOAA and EDX_CLIDBLK errors.

See Also

- dx_gtcallid( )
- dx_wtcallid( )
2. Voice

**dx_wtcallid** waits for rings and reports Caller ID

<table>
<thead>
<tr>
<th>Name:</th>
<th>int dx_wtcallid (chdev, nrings, timeout, bufferp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>int chdev</td>
</tr>
<tr>
<td></td>
<td>int nrings</td>
</tr>
<tr>
<td></td>
<td>short timeout</td>
</tr>
<tr>
<td></td>
<td>unsigned char *bufferp</td>
</tr>
</tbody>
</table>

| Returns: | 0 success                                      |
|          | -1 error return code                           |

| Includes: | srllib.h                                        |
|           | dxxxlib.h                                       |

| Category: | Caller ID                                       |
| Mode:     | synchronous                                    |

**Description**

The **dx_wtcallid()** function waits for rings and reports Caller ID, if available. Using this function is equivalent to using the voice functions **dx_setevtmsk()** and **dx_getevt()**, and the Caller ID function **dx_getcallid()** to return the caller's Directory Number (DN).

This function has the following parameters:
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chdev:</td>
<td>Channel device handle.</td>
</tr>
<tr>
<td>Nrings:</td>
<td>Number of rings to wait before answering.  &lt;br&gt;Valid values:  &lt;br&gt;[ \geq 1 ] (Note: Minimum 2 for CLASS and ACLIP)</td>
</tr>
<tr>
<td>timeout:</td>
<td>Maximum length of time to wait for a ring:  &lt;br&gt;Valid values (0.1-second units):  &lt;br&gt;[ \geq 0 ]  &lt;br&gt;-1 waits forever; never times out  &lt;br&gt;If timeout is set to zero and a ring event does not already exist, the function returns immediately.</td>
</tr>
<tr>
<td>Bufferp:</td>
<td>Pointer to where to return calling line Directory Number (DN).</td>
</tr>
</tbody>
</table>

On successful completion, a NULL terminated string containing the caller’s phone number (DN) is placed in the buffer.

**NOTE:** Non-numeric characters (punctuation, space, dash) may be included in the number string. The string may not be suitable for dialing without modification.

Caller ID information is available for the call from the moment the ring event is generated (if the ring event is set to occur on or after the second ring (CLASS, ACLIP), or set to occur on or after the first ring (CLIP)) until either of the following occurs:

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

### Cautions

`dx_wtcallid()` changes the event enabled on the channel to DM_RINGS.
2. Voice

Example

/*$ dx_wtcallid() example $*/
#include <srllib.h>
#include <dxoxlib.h>

unsigned char buffer[21];      /* char buffer */
int rc;                        /* value returned by function */
int chdev;                     /* channel descriptor */
unsigned short parmval;        /* Parameter value */

/* open channel */
if ((chdev = dx_open("dxxxB1C1", NULL) == -1) {
    /* process error */
}

/* Enable Caller ID */
parmval = DX_CALLIDENABLE;
if (dx_setparm(chdev, DXCH_CALLID, (void *)&parmval) == -1) {
    /* process error */
}

/* sit and wait for two rings on this channel - no timeout */
if (dx_wtcallid(chdev,2,-1,buffer) == -1) {
    printf("Error waiting for ring (with Caller ID): 0x%x\n", ATDV_LASTERR(chdev));
    /* process error */
}
printf("Caller ID = %s\n", buffer);

Errors

If this function returns -1 to indicate failure, use ATDV_LASTERR() and
ATDV_ERRMSGP() to retrieve one of the following error reasons:

- **EDX_BADPARM** Invalid parameter
- **EDX_BUSY** Channel is busy
- **EDX_CLIDBLK** Caller ID is blocked or private or withheld
  (other information may be available using dx_gtextcallid() )
- **EDX_CLIDINFO** Caller ID information not sent, sub-message(s) requested
  not available or Caller ID information invalid
- **EDX_CLIDOAA** Caller ID is out of area
  (other information may be available using dx_gtextcallid() )
- **EDX_SYSTEM** Operating system error - check errno
- **EDX_TIMEOUT** Time out limit is reached
2.4.1. Caller ID Related References

Sample application programs are provided that demonstrate this feature using the Dialogic API (SAMPLE.EXE) and TAPI (TALKER32.EXE). These programs are located in the SAMPLES\VOICE and SAMPLES\VOICE\TALKER32 subdirectories, respectively, under the Dialogic home directory (normally C:\Program Files\Dialogic\).

Caller ID is supported on the following boards:

- D/41ESC
- ProLine/2V
- D/21H, D/41H
- VFX/40ESCplus
- D/160SC-LS
- LSI/81SC
- LSI/161SC

It is **not** supported on D/21D, D/41D, DIALOG/4, VFX/40, VFX/40E, VFX/40SC, or VFX/40ESC boards.

**NOTE:** ACLIP and CLIP are not supported on the D/160SC-LS, LSI/81SC or LSI/161SC boards.

Before developing an application that requires CLASS, ACLIP or CLIP Caller ID information, contact your service provider and request the following appropriate detailed specifications:
Bellcore Documents for CLASS:
Information Exchange Management
Bellcore
445 South St., Room 2J-125
P. O. Box 1910
Morristown, NJ 07962-1910
Phone: 201-829-4785

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

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

- TAS TS PSTN1 A-CLIP: 1994
- Bellcore specification TR-NWT-000030 (see Bellcore address above)
2.5. Global Dial Pulse Detection

2.5.1. Global Dial Pulse Detection Overview

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

• Global DPD does not require a leading “0” to train the DPD algorithm. The algorithm is adaptive and can train on any DPD digit it encounters, with the greatest accuracy produced from training on a digit that has 5 or more pulses.

• Global DPD can be performed simultaneously with DTMF detection. The application can determine whether the digit detected is a DTMF or DPD digit.

• Global DPD can be performed simultaneously with Global Tone Detection (GTD). For example, the application can use GTD to monitor for disconnect tones (dial tone or busy) simultaneously with DPD.

• Global DPD supports pulse-digit cut-through during a voice playback, with the correct digit returned in the digit buffer. Global DPD uses echo cancellation, which provides more accurate reporting of digits during voice playback.
2. Voice

- The application can enable Global DPD and Volume Control. (Previously, there was a restriction that DPD digits had to be sent to the event queue instead of the digit queue if Volume Control was enabled.)

**Global Dial Pulse Detection (GDPD) is supported on the following boards:**

- D/41ESC
- ProLine/2V
- D/21H, D/41H
- VFX/40ESCalplus
- D/160SC-LS-IDPD
- D/300SC-E1-75-IDPD
- D/320SC-IDPD
- D/240SC-T1-IDPD
- D/300SC-E1-120-IDPD

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

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

Global DPD is **available on all** boards supported except the: D/21D, D/41D, DIALOG/4, VFX/40, VFX/40E, VFX/40SC, or VFX/40ESC boards.

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

**2.5.2. Supported Applications**

The following applications are supported by the Global DPD feature:

- Analog applications using the loop start telephone interface on the D/41ESC (and other low density boards) or D/160SC-LS-IDPD voice boards.
• Digital applications using the D/300SC-E1-75-IDPD or D/320SC-IDPD voice board.

2.5.3. Regional DPD Parameters

One reason for calling this Dial Pulse Detection implementation Global DPD is that the detection algorithm supports 8 pulse-per-second (PPS) to 22 PPS telephones. Dialogic is continuously qualifying its Dial Pulse Detection algorithm against dial pulse data collected from different parts of the world to improve DPD accuracy for the telephone systems and telephones in different regions. When appropriate, Dialogic issues downloaded parameters to improve the accuracy of DPD in a given region of the world, whether it is a part of a country, a whole country, or a collection of countries.

Customized Global DPD download parameters are provided for several countries, as noted in the Country-Specific Parameters. As more regions are qualified for customized Global DPD, additional region-specific support will be released.

Support for a generic 10 pulse-per-second (PPS) Global DPD is provided for other countries where customized support is not available.

You must install the Country-Specific Parameters and select a country to obtain support for Global DPD. You can specify "install country specific parameters" when you install the System Release software. The process for configuring depends on whether you are using the Intel or Digital AlphaServer platform. For Intel, select Country Specific Configuration from the Dialogic program group. For the Digital AlphaServer, select the Country button from the main screen of the Dialogic Configuration Manager (DCM).

Global DPD is provided depending upon the country selected in the Country-Specific Configuration program and the Dialogic boards in the system. If Global DPD is supported for the selected country and the Dialogic boards in your system, the Country-Specific Configuration program displays whether it is generic or customized Global DPD.
2.5.4. Programming Considerations for Accurate Global DPD

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

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

2.5.5. Global DPD Application Programming Interface

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

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

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

Programming Procedure

1. Define a data structure of type DV_DIGIT (the DV_DIGIT structure is specified in the DXDIGIT.H file).
2. Enable DPD on the desired channels using the `dx_setdigtyp()` function. For more information, refer to the Voice Software Reference. For new calls you must use the D_DPDZ mask that initializes the DPD detector for new calls.

3. Execute the `dx_getdig()` function to collect and transfer the digits to the user’s digit buffer. The digits are stored in the `dg_value` field of the DV_DIGIT structure with the corresponding digit types stored in the `dg_type` field of the DV_DIGIT structure. The following values distinguish the digit types that are returned in the `dg_type` field.

<table>
<thead>
<tr>
<th>Digit Type</th>
<th>Digit Buffer</th>
<th>Event Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTMF</td>
<td>DG_DTMF_ASCII</td>
<td>DG_DTMF</td>
</tr>
<tr>
<td>DPD</td>
<td>DG_DPD_ASCII</td>
<td>DG_DPD</td>
</tr>
<tr>
<td>MF</td>
<td>DG_MF_ASCII</td>
<td>DG_MF</td>
</tr>
<tr>
<td>GTD</td>
<td>DG_USER1_ASCII</td>
<td>DG_USER1</td>
</tr>
<tr>
<td></td>
<td>DG_USER2_ASCII</td>
<td>DG_USER2</td>
</tr>
<tr>
<td></td>
<td>DG_USER3_ASCII</td>
<td>DG_USER3</td>
</tr>
<tr>
<td></td>
<td>DG_USER4_ASCII</td>
<td>DG_USER4</td>
</tr>
<tr>
<td></td>
<td>DG_USER5_ASCII</td>
<td>DG_USER5</td>
</tr>
</tbody>
</table>

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

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

void main(int argc, char **argv)
{
  int dev;                          /* Dialogic device handle */
  DV_DIGIT dig;
  DV_TPT tpt;

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

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

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

  /* collect 3 digits from the user */
  if (dx_getdig(dev, &tpt, &dig, EV_SYNC) == -1) {
    /* error, display error message */
    printf("dx_getdig error %d, %s\n", ATDV_LASTERR(dev), ATDV_ERRMSGP(dev));
  } else {
    /* display digits received and digit type */
    printf("Received \n", dig.dg_value);
    printf("Digit type is ");
    /*
    * digit types have 0x30 ORed with them strip it off
    * so that we can use the DG_xxx equates from the header files
    */
    switch ({(dig.dg_type[0] & 0x00f})
    {
      case DG_DTMF:
        printf("DTMF\n");
        break;
      case DG_DPD:
        printf("DPD\n");
        break;
      default:
        printf("Unknown, \n", (dig.dg_type[0] & 0x00f));
        break;
    }
  }

  /*
  * continue processing call
  */
}
2.6. Transaction Record

2.6.1. Overview

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

2.6.2. Supported Voice Boards

The Transaction Record feature is compatible with the following Dialogic hardware:

- D/80SC
- D/160SC
- D/240SC
- D/320SC
- D/160SC-LS
- D/240SC-T1
- D/300SC-E1
- D/240SC-2T1
- D/300SC-2E1
- D/480SC-2T1
- D/600SC-2E1

**NOTE:** Transaction Record does not work with the D/41E or the D/41ESC boards.

2.7. Transaction Record Function

To implement the Transaction Record feature, the function `dx_mreciottdata()` was added to the Dialogic Voice Library. The new function is an extension of the `dx_reciottdata()` function (see the *Voice Software Reference for Windows NT*).
2. Voice

dx_mreciottdata  records voice data from two channels

Name:  dx_mreciottdata (devd, iotp, tptp, xpb, mode, sc_tsinfop)

Inputs:
- int  devd  • Dialogic channel descriptor
- DX_IOTT *iotp  • Pointer to I/O transfer table
- DV_TPT *tptp  • Pointer to termination control block
- DX_XPB *xpb  • Pointer to I/O transfer parameter block
- USHORT *mode  • Switch to set audible tone, or DTMF termination
- SC_TSINFO *sc_tsinfop  • Pointer to time slot information structure

Returns:  0 success
          -1 error return code

Includes: srllib.h
          dxxxlib.h

Category: I/O

Mode:  synchronous

Description

The dx_mreciottdata( ) function records voice data from two SCbus time slots. The data may be recorded to a combination of data files, memory or custom devices. This function has the following parameters:
Parameter  Description
devd: Specifies the valid Dialogic voice channel descriptor on which
the recording is to occur. The channel descriptor may be that
associated with either of the two SCbus transmit time slots or a
third device also connected to the SCbus.
Iotp: Pointer to the I/O TransferTable Structure, DX_IOTT. Specifies
the order of the voice data and the media on which it will be
recorded.
Tptp: Points to the Termination Parameter Table Structure, DV_TPT,
which specifies the termination conditions for recording. See the
Voice Software Reference for Windows NT for a complete list
of termination conditions.
Xpb: Points to a DX_XPB structure and specifies parameter values
for I/O data transfer. See the Voice Software Reference for
Windows NT for a complete list of parameters and valid values.
Mode: Specifies the attributes of the recording mode. One or more of
the following values can be specified:
  0  standard play mode
  RM_TONE  Transmits a 200ms tone before initiating
           record. If this mode is not selected, no
           tone is transmitted (default).
  RM_DTMFTERM  Terminate record on receiving any
                DTMF signal (leading edge) regardless
                of mode setting when tone was built.
Sc_tsinfop: Points to an SC_TSINFO structure and specifies the SCbus
transmit time slot values of the two time slots being recorded.

NOTE: Both RM_TONE and RM_DTMFTERM can be specified by ORing
the two values.

NOTE: When using RM_TONE bit for tone-initiated record, each time slot
must be "listening" to the transmit time slot of the recording channel;
the alert tone can only be transmitted on the recording channel’s
transmit time slot.

The structure for SC_TSINFO is as follows:

typedef struct {
2. Voice

```c
unsigned long sc_numts; /* Number of time slots in array */
long *sc_tsarrayp; /* Pointer to array of SCbus time slots */
```

where `sc_numts` should be set to 2 for channel recording and `sc_tsarrayp` should point to an array of two long integers, specifying the two SCbus transmit time slots from which to record.

After `dx_mreciotdata()` is called, recording continues until one of the following occurs:

- `dx_stopch()` is called on the channel whose device handle is specified in the `devd` parameter
- the data requirements specified in the DX_IOTT structure are fulfilled
- one of the conditions for termination specified in the DV_TPT structure is satisfied
- a DTMF digit is detected, if RM_DTMFTERM has been specified in the `mode` parameter

### Cautions

- All files specified in the DX_IOTT structure are of the file format specified in DX_XPB.
- All files recorded will have the same data encoding and rate as DX_XPB.
- When recording VOX files, the data format is specified in DX_XPB rather than through the `dx_setparm()` function.
- Voice data files that are specified in the DX_IOTT structure must be opened with the O_BINARY flag.
- When using MSI/SC stations for Transaction Recording, make sure a full duplex connection is established. You must issue an `ms_listen()` even though the MSI/SC station is used only for transmitting.
- Because the DSP sums the PCM values of the two SCbus time slots before processing them during transaction recording, all voice-related terminating conditions or features such as DTMF detection, Automatic Gain Control (AGC), and sample rate change will apply to both time slots. In other words,
for terminating conditions specified by a DTMF digit, either time slot containing the DTMF digit will stop the recording. Also, maximum silence length requires simultaneous silence from both time slots to meet the specification.

- If both time slots transmit a DTMF digit at the same time, the recording will contain an unintelligible result.

- The Transaction Record feature may not detect a DTMF digit over a dial tone.

- Since this application programming interface (API) uses `dx_listen()` to connect the channel to the first specified time slot, any error returned from `dx_listen()` will terminate the API with the error indicated. Refer to `dx_listen()` in the *Voice Software Reference for Windows NT* for the explanation of the errors.

- The API will connect the channel to the time slot specified in `sc_tsarrayp[0]` and remain connected after the function has been completed. Both `sc_tsarrayp[0]` and `sc_tsarrayp[1]` must be within the range 0 to 1023. No checking is done to verify that `sc_tsarrayp[0]` or `sc_tsarrayp[1]` has been connected to a valid channel.

- Upon termination of the `dx_mreciotdata()` function, the recording channel continues to listen to the first time slot (pointed to by `sc_tsarray[0]`).

- The application should check for a TDX_RECORD event with T_STOP event data after executing a `dx_stopch()` function during normal and transaction recording. This will ensure that all data is written to the disk.

- The recording channel can only detect a loop current drop on a physical analog front end that is associated with that channel. If you have a configuration where the recording channel is not listening to its corresponding front end, you will have to design the application to detect the loop current drop and issue a `dx_stopch()` to the recording device. The recording channel hook state should be off-hook while the recording is in progress.


**Example**

```c
#include <windows.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <srllib.h>
#include <dxxxlib.h>

#define MAXLEN 10000

main()
{
  int devh1, devh2, devh3;
  short fd;
  DV_TPT tpt;
  DX_IOTT iott[2];
  DX_XPB xpb;
  SC_TSINFO tsinfo;
  long scts;
  long tslots[32];
  char basebufp[MAXLEN];

  /* open two voice channels */
  if ((devh1 = dx_open("dxxxB1C1", NULL)) == -1) {
    printf("Could not open dxxxB1C1\n");
    exit (1);
  }

  if ((devh2 = dx_open("dxxxB1C2", NULL)) == -1) {
    printf("Could not open dxxxB1C2\n");
    exit (1);
  }

  if ((devh3 = dx_open("dxxxB1C3", NULL)) == -1) {
    printf("Could not open dxxxB1C3\n");
    exit (1);
  }

  if ((fd = dx_fileopen("file.vox", O_CREAT | O_RDWR | O_BINARY)) == -1) {
    printf("File open error\n");
    exit (1);
  }

  /* Get channels' external time slots * and fill in tslots[] array */
  tsinfo.sc_numts = 1;
  tsinfo.sc_tsrarrayp = &scts;
  if (dx_getxmitslot (devh1, &tsinfo) == -1) { /* Handle error */ }
  tslots[0] = scts;
  if (dx_getxmitslot (devh2, &tsinfo) == -1) { /* Handle error */ }
  tslots[1] = scts;

  /* Set up SC_TSINFO structure */
```
Dialogic System Software and SDK for Windows NT DNA Release Reference

tsinfo.sc_numts = 2;
tsinfo.sc_tsarrayp = &tslots[0];

/* Set up DX_XPB structure */
    xpb.wFileFormat = FILE_FORMAT_VOX;
xpb.wDataFormat = 0;
xpb.wSamplesPerSec = 0;
xpb.wBitsPerSample = 0;

/* Set up DX_TPT structure */
    dx_clrtpt (tpt,1);
tpt.tp_type = IO_EOT;
tpt.tp_termno = DX_MAXDTMF;
tpt.tp_length = 1;
tpt.tp_flags = TF_MAXDTMF;

/* Set up DX_IOTT structure */
    iott[0].io_fhandle = fd;
iott[0].io_type = IO_DEV;
iott[0].io_offset = 0;
iott[0].io_length = MAXLEN;
iott[0].io_offset = IO_EOT;

/* And record from both voice channels */
    if (dx_mreciottdata(devh3, &iott[0], &tpt, fxpb' RM_TONE, &tsinfo) == -1) { 
        printf("Error recording from dxxxB1C1 and dxxxB1C2\n"); 
        printf("error = %s\n", ATDV_ERRMSGP(devh1)); 
        exit(2); 
    }

/* Display termination condition value */
    printf("The termination value = %d\n", ATDX_TERMMSK(devh1));

/* And close three voice channels */
    if (dx_close(devh3) == -1) { 
        printf("Error closing devh3 \n");
        printf("errno = %d\n", errno);
        exit(3); 
    }
    if (dx_close(devh2) == -1) { 
        printf("Error closing devh2\n");
        printf("errno = %d\n", errno);
        exit(3); 
    }
    if (dx_close(devh1) == -1) { 
        printf("Error closing devh1\n");
        printf("errno = %d\n", errno);
        exit(3); 
    }
    if (dx_fileclose(fd) == -1) { 
        printf("File close error \n");
        exit(1); 
    }

/* And finish */
    return;
2. Voice

■ Errors

If this function returns -1 to indicate failure, use ATDV_LASTERR and ATDV_ERRMSGP to retrieve one of the following error reasons:

- EDX_BADDEV Invalid device handle
- EDX_BADIOTT Invalid DX_IOTT entry
- EDX_BADPARM Invalid parameter passed
- EDX_BADTPT Invalid DV_TPT entry
- EDX_BUSY Busy executing I/O function
- EDX_SYSTEM Operating system error - check errno

■ See Also

- dx_rec( )
- dx_play( )
- dx_reciofdata( )
- dx_playiofdata( )

2.8. Get Feature List Function

The following function call is new to the Voice library. This function can be used to check whether a particular voice device is enabled with the Syntellect patent license (that is, whether it is an “STC” board) as well as other features.
dx_getfeaturelist returns list of supported features

<table>
<thead>
<tr>
<th>Name:</th>
<th>int dx_getfeaturelist(chdev, feature_tablep)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>int chdev FEATURE_TABLE *feature_tablep</td>
</tr>
<tr>
<td></td>
<td>• voice channel handle • pointer to features information structure</td>
</tr>
<tr>
<td>Returns:</td>
<td>0 on success -1 on error</td>
</tr>
<tr>
<td>Includes:</td>
<td>dxxxxlib.h</td>
</tr>
<tr>
<td>Mode:</td>
<td>Synchronous</td>
</tr>
</tbody>
</table>

Description

The dx_getfeaturelist() function returns list of features supported on voice device. This function is available for use on all Dialogic voice devices.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chdev:</td>
<td>Specifies the valid voice channel handle obtained when the channel was opened using dx_open().</td>
</tr>
<tr>
<td>feature_tablep:</td>
<td>Specifies a pointer to the data structure FEATURE_TABLE that contains the bitmasks of various features.</td>
</tr>
</tbody>
</table>

On return from the function, the FEATURE_TABLE structure contains the relevant information and is declared as follows:

```c
typedef struct feature_table {
    unsigned short ft_play;
    unsigned short ft_record;
    unsigned short ft_tone;
    unsigned short ft_e2p_brd_cfg;
    unsigned short ft_fax;
    unsigned short ft_front_end;
    unsigned short ft_misc;
    unsigned short ft_rfu[8];
} FEATURE_TABLE;
```

Features reported by each member of the FEATURE_TABLE structure are defined in dxxlib.h. To determine what features are enabled on the voice device,
“bitwise AND” the returned bitmask with the following defines. For further information, refer to the *Example* section.

- **ft_play** contains a bitmask that informs you of the play features supported on the specified voice device.
  - FT_ADPCM
  - FT_PCM
  - FT_ALAW
  - FT_ULAW
  - FT_LINEAR
  - FT_ADSI
  - FT_DRT6KHZ
  - FT_DRT8KHZ
  - FT_DRT11KHZ

- **ft_record** contains a bitmask that informs you of the record features supported on the specified voice device.
  - FT_ADPCM
  - FT_PCM
  - FT_ALAW
  - FT_ULAW
  - FT_LINEAR
  - FT_ADSI
  - FT_DRT6KHZ
  - FT_DRT8KHZ
  - FT_DRT11KHZ

- **ft_tone** contains a bitmask that informs you of the tone features supported on the specified voice device.
  - FT_GTDENABLED
  - FT_GTGENABLED
  - FT_CADENCE_TONE

- **ft_e2p_brd_cfg** contains a bitmask that informs you of the board configuration features supported on the specified voice device.
  - FT_DPD
  - FT_SYNTELLECT

- **ft_fax** contains a bitmask that informs you of the fax features supported on the specified voice device.
  - FT_FAX
Dialogic System Software and SDK for Windows NT DNA Release Reference

FT_VFX40
FT_VFX40E
FT_VFX40E_PLUS

- ft_frontend contains a bitmask that informs you of the front end features supported on the specified voice device.
  - FT_ANALOG
  - FT_EARTH_RECALL

- ft_misc contains a bitmask that informs you of miscellaneous features supported on the specified voice device.
  - FT_CALLERID

- ft_rfu is reserved for future use.

Cautions

This function will fail if an invalid voice channel handle is specified.

Example

```c
#include <stdio.h>
#include <windows.h>
#include "srllib.h"
#include "dxxxlib.h"

void main(int argc, char ** argv)
{
    char chname[32] = "dxxxB1C1";
    int dev;
    FEATURE_TABLE feature_table;

    if ((dev = dx_open(chname, 0)) == -1) {
        printf("Error opening \"%s\"\n", chname);
        exit(1);
    }

    if (dx_getfeaturelist(dev, &feature_table) == -1) {
        printf("%s: Error %d getting featurelist\n", chname, ATDV_LASTERR(dev));
        exit(2);
    }

    printf("%s: Play Features:-\n", chname);
    if (feature_table.ft_play & FT_ADPCM) {
        printf("ADPCM ");
    }
    if (feature_table.ft_play & FT_PCM) {
        printf("PCM ");
    }
```
2. Voice

```c
if (feature_table.ft_play & FT_ALAW) {
    printf("ALAW ");
}
if (feature_table.ft_play & FT_ULAW) {
    printf("ULAW ");
}
if (feature_table.ft_play & FT_LINEAR) {
    printf("LINEAR ");
}
if (feature_table.ft_play & FT_ADSI) {
    printf("ADSI ");
}
if (feature_table.ft_play & FT_DRT6KHZ) {
    printf("DRT6KHZ ");
}
if (feature_table.ft_play & FT_DRT8KHZ) {
    printf("DRT8KHZ ");
}
if (feature_table.ft_play & FT_DRT11KHZ) {
    printf("DRT11KHZ");
}
printf("%

%s: Record Features:-
", chname);
if (feature_table.ft_record & FT_ADPCM) {
    printf("ADPCM ");
}
if (feature_table.ft_record & FT_PCM) {
    printf("PCM ");
}
if (feature_table.ft_record & FT_ALAW) {
    printf("ALAW ");
}
if (feature_table.ft_record & FT_ULAW) {
    printf("ULAW ");
}
if (feature_table.ft_record & FT_LINEAR) {
    printf("LINEAR ");
}
if (feature_table.ft_record & FT_ADSI) {
    printf("ADSI ");
}
if (feature_table.ft_record & FT_DRT6KHZ) {
    printf("DRT6KHZ ");
}
if (feature_table.ft_record & FT_DRT8KHZ) {
    printf("DRT8KHZ ");
}
if (feature_table.ft_record & FT_DRT11KHZ) {
    printf("DRT11KHZ");
}
printf("%

%s: Tone Features:-
", chname);
if (feature_table.ft_tone & FT_GTDENABLED) {
    printf("GTDEnabled ");
}
if (feature_table.ft_tone & FT_GTGENABLED) {
    printf("GTGENabled ");
}
if (feature_table.ft_tone & FT_CADENCE_TONE) {
    printf("CADENCE_TONE");
}
```

printf("\n\n%s: E2P Board Configuration Features:\n", chname);
if (feature_table.ft_e2p_brd_cfg & FT_DPD) {
  printf("DPD ");
}
if (feature_table.ft_e2p_brd_cfg & FT_SYNTELLECT) {
  printf("SYNTELLECT");
}

printf("\n\n%s: FAX Features:\n", chname);
if (feature_table.ft_fax & FT_FAX) {
  printf("FAX ");
}
if (feature_table.ft_fax & FT_VFX40) {
  printf("VFX40 ");
}
if (feature_table.ft_fax & FT_VFX40E) {
  printf("VFX40E ");
}
if (feature_table.ft_fax & FT_VFX40E_PLUS) {
  printf("VFX40E_PLUS");
}

printf("\n\n%s: FrontEnd Features:\n", chname);
if (feature_table.ft_front_end & FT_ANALOG) {
  printf("ANALOG ");
}
if (feature_table.ft_front_end & FT_EARTH_RECALL) {
  printf("EARTH_RECALL");
}

printf("\n\n%s: Miscellaneous Features:\n", chname);
if (feature_table.ft_misc & FT_CALLERID) {
  printf("CALLERID");
}
printf("\n");
dx_close(dev);
Errors

If the function returns -1, use the SRL Standard Attribute function ATDV_LASTERR( ) to obtain the error code or use ATDV_ERRMSGP( ) to obtain a descriptive error message. One of the following error codes may be returned:

<table>
<thead>
<tr>
<th>Equate</th>
<th>Returned When</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDX_BADParm</td>
<td>Parameter error</td>
</tr>
<tr>
<td>EDX_SH_BADEXTTS</td>
<td>SCbus time slot is not supported at current clock rate</td>
</tr>
<tr>
<td>EDX_SH_BADINDX</td>
<td>Invalid Switch Handler index number</td>
</tr>
<tr>
<td>EDX_SH_BADTYPE</td>
<td>Invalid local time slot channel type (voice, analog, etc.)</td>
</tr>
<tr>
<td>EDX_SH_CMDBLOCK</td>
<td>Blocking command is in progress</td>
</tr>
<tr>
<td>EDX_SH_LIBBSY</td>
<td>Switch Handler library busy</td>
</tr>
<tr>
<td>EDX_SH_LIBNOTINIT</td>
<td>Switch Handler library uninitialized</td>
</tr>
<tr>
<td>EDX_SH_MISSING</td>
<td>Switch Handler is not present</td>
</tr>
<tr>
<td>EDX_SH_NOCLK</td>
<td>Switch Handler clock fallback failed</td>
</tr>
<tr>
<td>EDX_SYSTEM</td>
<td>Windows NT System Error</td>
</tr>
</tbody>
</table>

See Also

- dx_getctinfo( )
3. ProLine/2V

3.1. ProLine/2V Audio Jacks Overview

The ProLine/2V voice board is a two-line voice board with audio jacks and a 2/3-size form factor (PC-XT height).

The ProLine/2V Line In and Line Out audio jacks allow application developers to record and play back prompts or other voice files using equipment such as a tape recorder or CD player without having to go through a telephony interface.

The ProLine/2V audio jacks use a “line level” interface, which is designed to take advantage of the line level interfaces commonly provided on most home audio equipment (see Figure 1). The line level interface works over channel 2 only, not channel 1.

- **Line Out**: The ProLine/2V Line Out interface is used to listen to prompts or voice files that are played over channel 2, which is typically done to verify the quality of the recording during application development. The Line Out interface can directly drive a tape deck or amplifier Line In or Aux In (even some headphones can be driven directly), as well as amplified speakers that have a “line level” input (not a “mic level” input).

  Whenever a prompt or voice file is played on channel 2, it always gets transmitted to the Line Out audio interface. It also gets transmitted to the channel 2 telephony interface (trunk) if the trunk is off-hook.

- **Line In**: Many microphones by themselves cannot record directly into a line level interface without amplification. However, tape decks that provide a Mic In jack and a Line Out or Aux Out jack usually contain a suitable microphone amplifier. (The microphone can be connected to the Mic In jack of the tape deck, and the Line Out or Aux Out jack from the tape deck can be connected to the Proline/2V Line In jack. Some decks require that you be in record mode for this to work.)

  ProLine/2V Line In audio jack supports a channel parameter that redirects voice channel 2 record operations from the RJ-11 jack to the Line In audio
jack. (See Section 2.3, *How To Switch the ProLine/2V Channel 2 Input Between the RJ-11 and Audio Jack Line In* for information on using the parameter.)

Channel 2 must be on-hook to record with the Line In audio jack. When you enable the Line In audio jack for recording, it disables the channel 2 telephony interface (trunk) and prevents channel 2 from going off-hook.

The application can record from channel 2 using either the analog telephony interface RJ-11 jack (trunk) or the line level Line In audio jack. The telephony interface is used for typical voice processing interactions with a user connected to the trunk (such as voice mail operations). The audio jacks are used for application development activity that is “off-line,” or outside the day-to-day operation of the voice processing server (such as recording voice prompts). The jacks are not intended for transmitting over or receiving voice from the trunk.

In other words, the audio jacks cannot be used to monitor or interact with calls in progress. The Line In and Line Out jacks do not connect to the trunk interface—they cannot receive any audio from the channel 2 trunk or transmit over the channel 2 trunk. Also, when you are using the Line Out interface to listen to a voice file played on channel 2, the channel 2 trunk should not be used for call processing because the audio being played will also be output to the trunk if the trunk is off hook.

### 3.2. Line In and Line Out Audio Jack Specifications

The ProLine/2V audio jack specifications are as follows:

<table>
<thead>
<tr>
<th>Audio Jack</th>
<th>Specification</th>
<th>Possible Connection</th>
</tr>
</thead>
</table>
| Line In     | Mini stereo plug (3.5 mm) > 2K ohm impedance * | • Microphone and amplifier  
• Tape deck  
• CD-ROM player |
| Line Out    | Mini stereo plug (3.5 mm) 1.5K ohm impedance * | • Speaker and amplifier           |

* amplification required
NOTE: Either stereo or monaural jacks can be used with the ProLine/2V audio jacks, although the ProLine/2V will only record and play monaural sound.

Figure 2 shows common stereo equipment connected to the ProLine/2V audio jacks. A description of these connections follows the figure.

**Figure 2. Sample ProLine/2V Audio Jack Connections**

After switching off the power to the stereo equipment and to the PC containing the ProLine/2V, make the connections as follows:

- Connect the speaker to one of the Line Out (may also be labelled as “speaker”) plugs or screw terminals on the stereo receiver/amplifier.

- Connect the stereo receiver/amplifier Line In plug or screw terminal (may also be labeled as “microphone” or “auxiliary in”) to the ProLine/2V Line Out audio jack.

- Plug the microphone into one of the microphone plugs on the tape deck.

- Connect the tape deck Line Out plugs or screw terminals to the ProLine/2V Line In audio jack.

- As you make the connections, you may need to pay attention to “left” and “right” plug and terminal labels depending on the number of conductors in the connecting cables and the types of plugs used. You may also need plug...
adapters if your equipment has a mix of screw terminals, mini stereo plugs and jacks, RCA plugs and jacks, and standard stereo plugs and jacks.

### 3.3. How To Switch the ProLine/2V Channel 2 Input

By default, the channel 2 input is connected to the RJ-11 jack. Use the following procedure to switch the input to record from the audio jack:

- Enable the audio jack Line In on channel 2.
- Record on channel 2.
- Disable the audio jack Line In on channel 2.

DXCH_AUDIOLINEIN is a new channel parameter for the `dx_setparm()` function that allows you to enable or disable the ProLine/2V audio jack Line In on voice channel 2. By specifying DXCH_AUDIOLINEIN as the value for `parm` and setting `valuep` to point to DX_LINEINENABLE or DX_LINEINDISABLE, you can enable or disable the audio jack Line In.

<table>
<thead>
<tr>
<th>Function</th>
<th>Channel</th>
<th>Parameter</th>
<th>Setting Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dx_setparm()</code></td>
<td>dev = 2</td>
<td><code>parm = DXCH_AUDIOLINEIN</code></td>
<td><code>valuep = (pointer to a setting below)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DX_LINEINENABLE</td>
<td>Enables the Line In audio jack (line level interface) for recording on channel 2, and disables the RJ-11 jack (analog telephony interface) on channel 2; also prevents channel 2 from going off-hook.</td>
</tr>
<tr>
<td>DX_LINEINDISABLE</td>
<td>Enables the Line In audio jack (line level interface) on channel 2 and enables the RJ-11 jack (analog telephony interface) for recording on channel 2; also allows channel 2 to go off-hook (default value).</td>
</tr>
</tbody>
</table>

Refer to the *Voice Software Reference* for information on the `dx_setparm()` function.
3. ProLine/2V

3.3.1. Cautions

Keep the following in mind when developing applications that use the ProLine/2V audio jacks:

- The audio Line In channel parameter can only be used with voice channel 2 on a ProLine/2V board.
- Channel 2 must be on hook when the audio Line In channel parameter is set to enable or disable the Line In audio jack, or else the command is rejected and an error sent back.
- Applications cannot record through the RJ-11 jack for channel 2 while the Line In audio jack is enabled.
- The setting of the audio Line In channel parameter stays in effect until it is reset to its default value or the card is reset or re-downloaded. For this reason, your application should reset the audio Line In channel parameter to its default value in the following situations:
  - Following the completion of recording through the Line In audio jack.
  - Whenever there is an error during a recording through the Line In audio jack.
  - Prior to exiting your application.

3.3.2. Example

```c
/*$ dx_setparm( ) example for Audio Jack Enablement  */
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>

/* Before enabling the audio input jack, verify that channel 2 is on-hook. */
int EnableAudioJack(int devhandle)
{
    short parm = DX_LINEINENABLE;
    if (dx_setparm(devhandle, DMCH_AUDIOLINEIN, &parm) == 0) {
        // success
        return 0;
    } else {
        // error
        MessageBox(NULL, "Error enabling Audio Input Jack", "Error", MB_ICONSTOP);
        return -1;
    }
}
```
3.4. Using the Audio Jack with the SAMPLE Demonstration Program

The SAMPLE application program located in the SAMPLES\VOICE\ directory under the Dialogic home directory (normally C:\Program Files\Dialogic\) demonstrates the ProLine/2V audio jack feature.

The audio jack is supported in the SAMPLE demonstration program so that you can see how to record and play voice prompts using the Line In and Line Out audio jacks as explained in Section 3.1. ProLine/2V Audio Jacks Overview. You can use this demonstration program to record the voice prompts for your application. The source code is provided along with the executable to help you develop applications that take advantage of the ProLine/2V audio jacks.
4. ISDN

4.1. Overview

Initial Release of the ISDN Primary Rate software for the following:

- System Release 4.25SC Production for Windows NT
- System Software and SDK for Windows NT 96.11
- System Software and SDK for Windows NT 97.01
- System Software and SDK for Windows NT 97.05

The following Dialogic boards are supported: D/240SC-T1, DTI/240SC, D/300SC-E1.

The integrated D/240SC-T1 and D/300SC-E1 boards have voice processing and ISDN connectivity. The DTI/240SC and DTI/300SC boards have ISDN connectivity without voice processing.

4.2. Installation Procedure

The ISDN software is not installed as part of the Typical installation option. To install ISDN, run the installation program from the System Release Software and SDK CD-ROM and choose the **Custom** install on the **Setup Type** dialog.

4.3. ISDN Features Summary

ISDN technology offers the benefits inherent in digital connectivity such as fast connection and fast Dialed Number Identification Service (DNIS) and Automatic Number Identification (ANI) acquisition.
The following ISDN features are supported:

- **Vari A Bill**  
  A flexible billing option enabling a customer to modify the charge for a call while the call is in a stable state, for example, between answer and disconnect. This feature is available from the AT&T network only.

- **ANI on demand**  
  Allows the user to request a Caller ID number to identify the origin of the call, when necessary.

- **User to user information**  
  An information element included in setup, connect, or disconnect messages.

- **Call by Call service selection**  
  This feature allows the user to access different services, such as an 800 line or WATS line, on a per call basis.

### 4.4. Diagnostic Tools (The DialView Suite)

DialView is a suite of tools to help developers test and debug their ISDN applications. The DialView suite includes:

- ISDN Network Emulation Firmware (NT1 and NE1)
- ISDN Diagnostic Program (*isdiag*)
- PRI Trace Utility (*pritrace*)

The ISDIAG Utility Program assists the ISDN line installer in verifying operation of the ISDN line. The program also serves as a troubleshooting aid for isolating the fault area when a problem occurs. ISDIAG can also be used by system developers to examine and verify the functionality of their application via the command line and tracing features.

**NOTE:** ISDIAG is not intended as an application tester when installed in a system using NT emulation software (*isnt1.fwl* or *dtint1.fwl*, T1 only).
4. **ISDN**

To start the program, type "isdiag parm1 parm2 parm3 parm4 parm5." Parm1 is the board number, parm2 is the channel time slot number, parm3 is the interface type (t for T-1 and e for E-1), parm4 is the bus type (S for SCbus) and parm5 is optional to indicate voice support (v for voice).

After the channel number and bus mode are selected, the program will automatically configure the system and display the first level menu. At this level, the user can choose the following actions:

- Set call parameter
- Request calling party number (ANI)
- Send maintenance request
- Display information
- Drop call
- Make call
- Play
- Record
- Stop play/record
- Start or stop trace
- Restart and wait call
- ESC exit

The PRI trace utility program (*pritrace.exe*) translates the recorded binary ISDN trace file (*filename.log*) into a formatted text file (*filename.res*) for easy reading. The binary trace file is generated using the **cc_StartTrace()** and **cc_StopTrace()** functions.
5. **MSI/SC**

5.1. **ms_genringex( ) function**

The `ms_genringex( )` function may be used to generate distinctive ringing:
The `ms_genringex()` function generates distinctive ringing to a station. The function will terminate when the phone goes off hook or the specified number of rings has been generated.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devh:</td>
<td>The station device handle.</td>
</tr>
<tr>
<td>Len:</td>
<td>The number of cycles to ring a station. A maximum value of 255 is allowed.</td>
</tr>
</tbody>
</table>
| Mode:     | The operation mode  
For synchronous mode, EV_SYNC must be specified as the third parameter. The function will return only on termination of ringing due to an error, off hook, or completion of ring cycles.  
For asynchronous mode, EV_ASYNC must be specified as the third parameter. The function will return on initiation of ringing or on error. To get the completion status, a termination event is generated. |
| Cadid:    | Cadence ID for distinctive ringing. Range: 0–8. See ms_setbdparm( ) MSG_DISTINCTRNG for information on initializing distinctive ring and assigning cadence IDs. Set cadid to MS_RNG_DEFAULT to use the default ring for that station. Rings generated either by the ms_genring( ) function or by ms_genringex( ) function when cadid is set to MS_RNG_DEFAULT will use the default ring cadence. |
NOTES:  1. This function is only supported on MSI/SC-R boards.

2. `ms_genringex()` will fail when executed on a station currently off hook. The error returned is E_MSBADRNGSTA.

3. Dialogic recommends specifying a length, or cycle, of at least two rings to make sure that at least one full ring cycle is generated.

4. A distinctive ring will “stick” to a station and become that station’s default ring. If you generate a distinctive ring on a station by setting the cadid parameter to a cadence ID, the specified distinctive ring becomes the default ring cadence for that station. Future rings generated either by the `ms_genringex()` function when cadid is set to MS_RNG_DEFAULT or by the `ms_genring()` function will use the default ring cadence. The default ring cadence is either the last distinctive ring that was generated on the station, or if none, the board-level ring cadence.

5. `ms_genringex()` will fail when specifying an invalid cadid or if distinctive ring has not been initialized with the `ms_setbrdparm()` MSG_DISTINCTRNG parameter. The error returned is E_MSBADRNGCAD.

6. A glare condition occurs when two parties seize the same line for different purposes. Although very rare, if glare occurs in your application, the function returns successfully. However, it is followed by the event MSEV_NORING. The data associated with the event is E_MSBADRNGSTA, indicating that the station was off hook when the ring was attempted.

7. When you ring a station, a built-in (non-modifiable) 500 ms ring is “splashed” to the station immediately before its ring cadence begins. The splash may make the beginning of the ring cadence sound slightly different from the rest of the cadence. This ring splash serves as a fast way to produce a ring at the station and thus to reduce the glare window. Otherwise, glare could occur when a ring starts in the off-time (non-ringing) portion of the ring cycle (where there is no notification that the phone is being rung) and a person picks up the (silent) phone expecting to get dial tone and instead is connected with a caller.
In asynchronous mode, 0 indicates that the function was initiated while AT_FAILURE indicates error. For successful completion of ringing, MSEV_RING will be returned. MSEV_NORING will be returned if the ring is not successful. The event data for MSEV_RING is as follows:

- **MSMM_RNGOFFHK**  Solicited off hook detected
- **MSMM_TERM**     Ringing terminated

In synchronous mode, AT_FAILURE indicates failure and a positive value (>0) indicates the reason for termination. Reasons for termination are:

- **MSMM_RNGOFFHK**  Solicited off hook detected
- **MSMM_TERM**     Ringing terminated

## Cautions
This function fails when:

- The board is not an MSI/SC-R board.
- The device handle is invalid.

## Example

Synchronous mode:

```c
/* $ ms_genringex() example */
#include <windows.h>
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int dev1;    /* Station device descriptor */
int rc;      /* Return code */
MS_CADENCE cadence;
BYTE pattern;

/* Open board 1, station 1 device */
if ((dev1 = ms_open("msiB1C1",0)) == -1) {
    printf( "Cannot open MSI B1, station 1: errno=%d", errno);
    exit(1);
}
/*
* Setup distinctive cadence

```c
// First distinctive cadence

cadence.cadid = 1; // First distinctive cadence

cadence.cadlength = MS_RNGA_CADLEN;

pattern = MS_RNGA_TWOSEC;

cadence.cadpattern = &pattern; // Pattern (secs) : 2 on 4 off

/* Set 1st ring cadence to MS_RNGA_TWOSEC */
if (ms_setbrdparm(devh, MSG_DISTINCTRNG, (void *)&cadence) == -1) {
    printf("Error setting board parameter : %s\n", ATDV_ERRMSGP(devh));
    exit(1);
}

/* Continue processing */
/* Generate ringing using distinctive ring 1 */
if ((rc =ms_genringex(dev1, 10, EV_SYNC, 1)) == -1) { // process error */
    /* If timeout, process the condition */
    if (rc == MSMM_TERM) {
        printf("Station not responding");
    }

    /* Continue Processing */

/* Done processing - close device */
if (ms_close(dev1) == -1) {
    printf("Cannot close device msiB1C1. errno = \d", errno);
    exit(1);
}
```

Asynchronous mode:

```c
#include <windows.h>
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  dev1;      /* Station dev descriptor */
int  srlmode;   /* SRL mode indicator */
MS_CADENCE cadence;
BYTE pattern;

/* Open board 1, station 1 device */
if ((dev1 = ms_open("msiB1C1", 0)) == -1) {
    printf("Cannot open MSI B1, station 1: errno=\d", errno);
    exit(1);
}

/* Set SRL to run in polled mode */
srlmode = SR_POLLMODE;
if (sr_setparm(SRL_DEVICE, SR_MODEID, (void *)&srlmode) == -1) {
    /* process error */
}

/* Set up handler function to handle play completion */
```
if (sr_enbhdlr(dev1,MSEV_RING,sig_hdlr) == -1) {
  /* process error */
}

/* Setup distinctive cadence 1 */
cadence.cadid = 1; /* First distinctive cadence */
cadence.cadlength = MS_RNGA_CADLEN;
pattern = MS_RNGA_TWOSER;
cadence.cadpattern = &pattern; /* Pattern (secs) : 2 on 4 off */

/* Set 1st ring cadence to MS_RNGA_TWOSER */
if (ms_setbrdparm(devh, MSG_DISTINCTRNG, (void *)&cadence)) == -1) {
  printf("Error setting board parameter : %s\n",
         ATDV_ERRMSGP(devh));
  exit(1);
}
/* Continue processing */
/* Generate asynchronous ringing using distinctive ring 1 */
if ((rc =ms_genringex(dev1,10,EV_ASYNC,1)) == -1) {
  /* process error */
}

/* Use sr_waitevt to wait for the completion of ms_genring().
   On receiving the completion event, MSEV_RING, control is
   transferred to the handler function previously established
   using sr_enbhdlr(). */

/* Continue processing */
/* Done processing - close device */
if (ms_close(dev1) == -1) {
  printf("Cannot close device msiB1C1. errno = %d", errno);
  exit(1);
}
/* Continue processing */

int sig_hdlr() {
  int dev = sr_getevtdev();
  unsigned short *sigtype = (unsigned *)sr_getevtdatap();
  if (sigtype != NULL) {
    switch (*sigtype) {
      case MSMM_TERM:
        printf("Station does not answer");
        return 0;
      case MSMM_RNGOFFHK:
        printf("Station offhook detected\n");
        return 0;
      default:
        return 1;
    }
  }
}
/*
 * Continue processing
 */

**Errors**

If the function does not complete successfully, it will return AT_FAILURE to indicate error. Use the Standard Attribute function ATDV_LASTERR() to obtain the applicable error value(s).

Error defines can be found in dtilib.h or msilib.h.

**See Also**

- ms_genring()
- ms_setbrdparm()
- ms_setevtsk()
6. DCB/SC

6.1. Features

This Package supports the Dialogic DCB/SC product series, consisting of the following boards:

| DCB/320SC  | 32 conferencing resources | 1 DSP |
| DCB/640SC  | 64 conferencing resources  | 2 DSP’s |
| DCB/960SC  | 96 conferencing resources  | 3 DSP’s |

- SCbus access for building high density systems.
- Each product in the DCB/SC series contains one, two, or three DSP(s). Each Digital Signal Processor (DSP) supports up to 32 conferees, for a maximum of 96 (3 sets of 32) conferencing resources.
- Conferencing applications written for the MSI/SC board are easily portable to the DCB/SC board.
- Active talker status determines who is talking in a conference at any given time.
- On-board DTMF detection with tone clamping allows conferees using touch-tone to enter DTMF digits without disturbing the other conferees.
- Monitoring allows many parties to monitor a single conference.
- Coaching allows one conferee (the coach) to speak to a specific conferee (the pupil) without any of the other conferees hearing the coach.
- Individual volume control allows a participant to vary the receive volume via DTMF digits.

Refer to the Dialogic Audio Conferencing Software Reference for Windows NT for more information on product features.
6.2. Notes and Hints

If multiple handles are opened on a single DSP, a digit detection event is received for each handle.
7. TAPI/WAVE 2.0

7.1. TAPI/WAVE 2.0 Requirements

TAPI/WAV requires that you have installed Windows NT 4.x or a later version of Windows NT, which includes TAPI 2.0.

You must activate the Dialogic Telephony Service Provider (TSP) and WAVE driver according to the instructions found in section 7.2.

7.2. TAPI/WAVE 2.0 Summary

The following is a summary of relevant features. For additional details, see the Dialogic TAPI and WAVE Support Reference in the Telephony Service Provider (TSP) Configuration on-line help. The Dialogic Telephony Service Provider (TSP) Configuration is available from the Configure button on the Telephony Drivers tab of the Control Panel Telephony applet. The on-line help is available from the Help button or you can open the D41MT.HLP file installed in the SYSTEM32 subdirectory under Windows.

7.2.1. Telephony Service Provider (TSP) and TAPI

- The Dialogic Telephony Service Provider (TSP) supports the Microsoft Windows Telephony Applications Programming Interface (TAPI) Version 2.0. (See the lineInitializeEx() function in the “Programming Considerations” section for an item regarding the TAPI version being used.)

- The Dialogic TSP supports the following boards, which contain on-board analog front ends, in a stand-alone configuration. (The SCbus and time slot routing are not supported on SCbus boards.)
  - ProLine/2V
  - D/21H, D/41H
  - D/21D, D/41D, DIALOG/4
  - D/41E, D/41ESC
  - VFX/40, VFX/40E, VFX/40SC, VFX/40ESC, VFX/40ESCplus
• D/160SC-LS
• D/240SC
• D/320SC

The Dialogic TSP supports up to 128 channels.

The Dialogic Telephony Service Provider (TSP) Configuration is available from the Configure button on the Telephony Drivers tab of the Control Panel Telephony applet. The TSP Configuration allows you to configure board parameters, call parameters, tone parameters, as well as optional parameters for PBX features when supported by the PBX. For details, see the on-line help (the D41MT.HLP file is installed in the SYSTEM32 subdirectory under Windows).

The Dialogic TALKER32 demonstration program (a TAPI-based GUI demonstration of selected Dialogic board features). The program is installed in the Dialogic SAMPLES folder. For details, see the on-line help provided with the program.

Each channel of the voice device is a TAPI line device with respect to device capabilities and applicable functions.

The boards are not modeled as phone devices.

The Dialogic TSP supports only one terminal (TAPI.DLL maintains terminal IDs, so that messages can be sent to different terminals).

The Dialogic TSP is installable as a standard Windows service provider.

No device-specific extensions are implemented in this version of the TSP; device extensions are disabled.

**Conferencing Support**: The Dialogic Telephony Service Provider (TSP) supports conferencing on TAPI with the telephone switch handling Dialogic devices as regular analog telephones. The conferencing capabilities are strictly defined and limited by the telephone switching equipment. (Note that the TSP does not support the MSI board or its conferencing capabilities.) The following functions are supported:

• lineAddToConference()  
• lineCompleteTransfer() with LINETRANSFERMODE_CONFERENCE  
• lineGetConfRelatedCalls()  
• linePrepareAddToConference()
7. TAPI/WAVE 2.0

- `lineSetupConference()`

**NOTES:**
1. The `lineRemoveFromConference` function controls the operation of the telephone switching equipment and is dependent upon the capabilities of that equipment; it does not control the Dialogic voice devices and hence is not supported by the Dialogic TSP:

2. **Caller ID:** The Dialogic Telephony Service Provider (TSP) supports Caller ID on TAPI. Caller ID is always enabled. The Call Parameters tab in the TSP Advanced Configuration dialog allows you to select when the service provider will Report Incoming Calls. The default is after the second ring. This will work for most Caller ID services. If you set the parameter to 1 ring, you will not be able to obtain the Caller ID from a service that transmits the Caller ID information between the first and second rings, as with CLASS (North America) and ACLIP (Singapore) Caller ID service.

3. **Global Dial Pulse Detection (GDPD):** Global DPD can be enabled in TAPI only on those boards that support Global DPD by setting the `dwDigitModes` parameter to `LINEDIGITMODE_PULSE` in the `lineMonitorDigits()` or `lineGatherDigits()` functions.

4. If your board does not have an IDPD-suffix and you wish to enable Global DPD, you must order a separate Global DPD enablement diskette from Dialogic.

7.2.2. WAVE Device Driver

- WAVE is supported on all voice boards that are supported.

- The Dialogic D/41D, D/21D, DIALOG/4, VFX/40, VFX/40E, VFX/40SC, and VFX/40ESC boards support the WAVE API (see the following table for details).

- The WAVE driver detects and supports up to 128 channels.

- Each channel of the voice device is a WAVE device with respect to device capabilities and applicable functions.

- The WAVE Device Driver is installable as a standard Windows installable driver.
• The WAVE Device Driver uses the callback functions supplied by MMSYSTEM.DLL to send asynchronous completion, line event, and phone event messages to MMSYSTEM.DLL.

• The actual number of Dialogic WAVE devices present on the system can be viewed through the Devices tab of the Control Panel Multimedia applet. These devices are displayed when the boards have been started and removed when the boards are stopped.

• The system will try to use Dialogic boards to play system sounds if there is no standard audio card installed.

• **Waveform Data Formats Directly Supported by Dialogic Boards for WAVE Audio Files:** Waveform audio data formats for WAVE files are directly supported by the Dialogic boards as indicated in the following table (all formats are monophonic).

<table>
<thead>
<tr>
<th>Digitizing Method (resolution)</th>
<th>ADPCM$^1$ (4-bit)</th>
<th>PCM (8-bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mu-Law$^2$</td>
</tr>
<tr>
<td>Sampling Rate (KHz)</td>
<td>6 8 6 8 11</td>
<td>6 8 11</td>
</tr>
<tr>
<td>Bit Rate (Kbps)</td>
<td>24 32 48 64 88</td>
<td>48 64 88</td>
</tr>
<tr>
<td>D/21D</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>D/41D</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>DIALOG/4</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>VFX/40</td>
<td>X X X X</td>
<td>X X</td>
</tr>
<tr>
<td>VFX/40E</td>
<td>X X X X</td>
<td>X X</td>
</tr>
<tr>
<td>VFX/40SC</td>
<td>X X X X</td>
<td>X X</td>
</tr>
</tbody>
</table>
### 7. TAPI/WAVE 2.0

<table>
<thead>
<tr>
<th>VFX/40ESC</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFX/40ESC+</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ProLine/2V</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D/21H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D/41H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D/41E</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D/41ESC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HD Voice§</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**X** = Supported (Non-Standard Waveform Data Format)

**+** = Supported (Standard Waveform Data Format)

1. `WAVE_FORMAT_DIALOGIC_OKI_ADPCM`
2. `WAVE_FORMAT_MULAW`
3. `WAVE_FORMAT_ALAW`
4. `WAVE_FORMAT_PCM`

**Standard Waveform Data Formats Indirectly Supported by Using the ACM:**

The following table shows which Dialogic boards directly support the 11 KHz PCM Linear “standard” Waveform data format. These Dialogic boards also indirectly support other standard Waveform data formats (11 KHz, 22 KHz, and 44 KHz, 8- and 16-bit mono and stereo PCM Linear) through use of the Audio Compression Manager (ACM) and the Microsoft PCM Converter, which can convert between the different standard formats “on-the-fly.” See the topic on “Waveform Data Formats” in the Dialogic TSP on-line help for additional information on the converter (the D41MT.HLP file is installed in the SYSTEM32 subdirectory under Windows).

Currently, no CODEC is available for the ACM to indirectly support standard formats on Dialogic boards that support only non-standard formats (D/41D, D/21D, DIALOG/4, VFX/40, VFX/40E, VFX/40SC, and VFX/40ESC).
### Standard Waveform Audio Data Formats

<table>
<thead>
<tr>
<th>Method</th>
<th>Resolution</th>
<th>Sampling Rate (KHz)</th>
<th>Bit Rate (Kbps)</th>
<th>PCM Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8-Bit Mono or Stereo</td>
<td>11.025</td>
<td>22.05</td>
<td>44.1</td>
</tr>
<tr>
<td></td>
<td>16-Bit Mono or Stereo</td>
<td>11.025</td>
<td>22.05</td>
<td>44.1</td>
</tr>
<tr>
<td>VFX/40ESC+</td>
<td>+</td>
<td>88.2</td>
<td>176.4</td>
<td>352.8</td>
</tr>
<tr>
<td>ProLine/2V</td>
<td>+</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D/21H</td>
<td>+</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D/41H</td>
<td>+</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D/41E</td>
<td>+</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D/41ESC</td>
<td>+</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HD Voice†</td>
<td>+</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

+ = Supported Directly on Dialogic Board in Monophonic format
X = Supported Indirectly on these Dialogic Boards Through the Audio Compression Manager and the Microsoft PCM Converter
† High-density voice boards: D/80SC, D/240SC, D/320SC, D/240SC-T1, D/300SC-E1, D/160SC-LS

### 7.3. Installing and Removing the Dialogic TSP and WAVE

These instructions apply only to systems on which you are going to develop or use a TAPI/WAVE 2.0 application program. They describe how to install and remove the WAVE driver and the Dialogic Telephony Service Provider (TSP), which allow Windows telephony applications to access the Dialogic hardware.
7. TAPI/WAVE 2.0

- After you have successfully configured, installed, and started Dialogic boards for the first time, you must perform this procedure once on any system where you are going to develop or use a TAPI application program.

- You should also perform this procedure any time after you have started the Dialogic boards following a change to the Dialogic board configuration (add, remove, disable or enable boards).

7.3.1. TAPI/WAVE 2.0 Requirements

- You must have installed Windows NT 4.x or a later version of Windows NT, which includes TAPI 2.0.

- You must have installed the Dialogic boards and the Dialogic software release and confirmed that the boards are functional.

- The Dialogic boards must be started using the Set Dialogic Service Start Up Mode.

7.3.2. Activate the Dialogic TSP for TAPI

1. Run the Telephony applet in the Control Panel:
   A. Click Start.
   B. Point to Settings.
   C. Click Control Panel.
   D. Double-click Telephony.

2. Click the Telephony Drivers tab.

3. If the Dialogic D41MT Service Provider for NT does not appear in the Telephony Drivers tab, do the following to add it:
   A. Click Add to display the Add Driver dialog.
   B. Click the **Dialogic D41MT Service Provider for NT** to select it (if it is not shown, restart your PC and try again).
   C. Click Add. The Dialogic TSP Configuration dialog is displayed.

4. Click **Detect Boards** to update the list of Dialogic board channels (dxxxB1C1 is board 1 channel 1). The list is updated to match the
configuration that was most recently started. You can click Help to access on-line help for the TSP configuration.

5. Click OK to close the Dialogic TSP Configuration dialog. The Dialogic TSP should now appear in the Telephony Drivers tab. Note that the Configure button can be used to redisplay the Dialogic TSP Configuration dialog.

6. Click OK to exit the Telephony applet.

7.3.3. Activate the Dialogic WAVE Driver

1. Run the Multimedia applet in the Control Panel:
   A. Click Start.
   B. Point to Settings.
   C. Click Control Panel.
   D. Double-click Multimedia.

1. Click the Devices tab.

2. Double click the Audio Devices item to expand it.

3. If the Audio for Dialogic WAVE Driver does not appear, do the following to add it:
   A. Click Add.
   B. Select Unlisted or Updated Driver and click OK.
   C. Click Browse.
   D. Select the Dialogic\LIB directory under the Dialogic home directory (normally C:\Program Files), and click OK.
   E. Click OK.
   F. Click Restart System Now.

Do the following to update the Dialogic WAVE configuration after adding, removing, disabling, or enabling Dialogic boards, or to view the actual number of Dialogic WAVE devices present on the system:

1. Run the Control Panel Multimedia applet.
2. Click the Devices tab.
3. Double click the Audio Devices item to expand it.
4. Select Audio for Dialogic WAVE Driver.
5. Click Properties.
6. Click Settings.
7. **Click OK.**

### 7.3.4. Uninstalling the Software

If you are going to uninstall the Dialogic software, you must perform the following procedure prior to the uninstall to de-activate the Dialogic TSP and WAVE driver and to make sure that the uninstall removes these components:

**Deactivate the Dialogic TSP for TAPI**

1. Run the Control Telephony applet.
2. Click the Telephony Drivers tab.
3. Select **Dialogic D41MT Service Provider for NT**.
4. Click Remove, and then click Yes to confirm.
5. Click Don’t Restart Now.

**Deactivate the Dialogic WAVE Driver**

1. Run the Control Panel Multimedia applet.
2. Click the Devices tab.
3. Double click the Audio Devices item to expand it.
4. Select **Audio for Dialogic WAVE Driver**.
5. Click Remove and click Yes to confirm.
6. Click Don’t Restart Now.
8. Country-Specific Parameters

8.1. Country-Specific Parameter System Overview

The following features are new:

8.1.1. New Country-Specific Support

Added country-specific support for the following countries:

<table>
<thead>
<tr>
<th>Country</th>
<th>Country Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greece</td>
<td>GR</td>
</tr>
<tr>
<td>Ireland</td>
<td>IE</td>
</tr>
<tr>
<td>Morocco</td>
<td>MA</td>
</tr>
</tbody>
</table>

8.1.2. New Custom (Global DPD) Support

Added customized Global DPD support for ProLine/2V, D/41E, D/41E-IDPD, D/41H, D/21H and VFX/40ESCplus, D/160SC-LS-IDPD, D/300SC-E1-75-IDPD and D/320SC-IDPD boards in the following countries:

<table>
<thead>
<tr>
<th>Country</th>
<th>Country Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argentina</td>
<td>AR</td>
</tr>
<tr>
<td>Brazil</td>
<td>BR</td>
</tr>
<tr>
<td>Colombia</td>
<td>CO</td>
</tr>
<tr>
<td>Japan</td>
<td>JP</td>
</tr>
<tr>
<td>India</td>
<td>IN</td>
</tr>
<tr>
<td>Mexico</td>
<td>MX</td>
</tr>
<tr>
<td>Venezuela</td>
<td>VE</td>
</tr>
</tbody>
</table>

If you select one of these countries, Global DPD will use customized country-specific parameters.
Global DPD will work only on DPD-enabled boards. You must order a separate Global DPD enablement package from Dialogic to enable Global DPD on any Dialogic board (except for boards with the “IDPD” suffix which are already enabled).

8.1.3. New Generic (Global DPD) Support

Added generic 10 pulse-per-second (PPS) Global DPD as a common default to be used by supporting Global DPD boards when customized Global DPD parameters do not exist.

8.1.4. Country-Specific Parameter File

The following features are supported:

- Pre-configured parameter files for the specific board configurations in specific countries.
- Specific firmware parameter features that can be downloaded from the parameter files.
- Intelligent Network Interfaces Products support.

8.2. Supported Countries

The following countries are supported:
8. Country-Specific Parameters

<table>
<thead>
<tr>
<th>Country</th>
<th>Country Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argentina</td>
<td>AR</td>
</tr>
<tr>
<td>Australia</td>
<td>AU</td>
</tr>
<tr>
<td>Austria</td>
<td>AT</td>
</tr>
<tr>
<td>Belgium</td>
<td>BE</td>
</tr>
<tr>
<td>Brazil</td>
<td>BR</td>
</tr>
<tr>
<td>Colombia</td>
<td>CO</td>
</tr>
<tr>
<td>Denmark</td>
<td>DK</td>
</tr>
<tr>
<td>Finland</td>
<td>FI</td>
</tr>
<tr>
<td>France</td>
<td>FR</td>
</tr>
<tr>
<td>Germany</td>
<td>DE</td>
</tr>
<tr>
<td>Greece</td>
<td>GR</td>
</tr>
<tr>
<td>Hungary</td>
<td>HU</td>
</tr>
<tr>
<td>India</td>
<td>IN</td>
</tr>
<tr>
<td>Ireland</td>
<td>IE</td>
</tr>
<tr>
<td>Israel</td>
<td>IL</td>
</tr>
<tr>
<td>Italy</td>
<td>IT</td>
</tr>
<tr>
<td>Japan</td>
<td>JP</td>
</tr>
<tr>
<td>Korea</td>
<td>KR</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>LU</td>
</tr>
<tr>
<td>Malaysia</td>
<td>MY</td>
</tr>
<tr>
<td>Mexico</td>
<td>MX</td>
</tr>
<tr>
<td>Morocco</td>
<td>MA</td>
</tr>
<tr>
<td>Netherlands</td>
<td>NL</td>
</tr>
<tr>
<td>New Zealand</td>
<td>NZ</td>
</tr>
<tr>
<td>Norway</td>
<td>NO</td>
</tr>
<tr>
<td>Poland</td>
<td>PL</td>
</tr>
<tr>
<td>Portugal</td>
<td>PT</td>
</tr>
<tr>
<td>Singapore</td>
<td>SG</td>
</tr>
<tr>
<td>South Africa</td>
<td>ZA</td>
</tr>
<tr>
<td>Spain</td>
<td>ES</td>
</tr>
<tr>
<td>Sweden</td>
<td>SE</td>
</tr>
<tr>
<td>Switzerland</td>
<td>CH</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>UK</td>
</tr>
<tr>
<td>United States</td>
<td>US</td>
</tr>
<tr>
<td>Venezuela</td>
<td>VE</td>
</tr>
</tbody>
</table>
8.3. Dialogic Boards Supported by Country

The following table lists the parameter files included for the specific board configurations associated with each country. The board families are:

D/xxD Boards: D/21D, D/41D, DIALOG/4;  D/xxE Boards: D/41E, D/41ESC, VFX/40, VFX/40E, VFX/40SC, VFX/40ESC, VFX/40ESCplus;  D/xxH Boards: D/21H, D/41H;  D/160SC Boards: D/160SC-LS;  D/240SC Boards: D/240SC-T1, D/240SC-2T1, D/480SC-2T1, D/300SC Boards: D/300SC-E1;  ProLine Boards: ProLine/2V.

<table>
<thead>
<tr>
<th>Country</th>
<th>Country Code</th>
<th>File Name</th>
<th>Boards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argentina</td>
<td>AR</td>
<td>ar_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ar_41h.prm</td>
<td>D/xxH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ar_p2v.prm</td>
<td>ProLine</td>
</tr>
<tr>
<td>Australia</td>
<td>AU</td>
<td>au_300.prm</td>
<td>D/300SC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>au_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>au_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Austria</td>
<td>AT</td>
<td>at_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Belgium</td>
<td>BE</td>
<td>be_300.prm</td>
<td>D/300SC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Brazil</td>
<td>BR</td>
<td>br_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>br_41h.prm</td>
<td>D/xxH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>br_p2v.prm</td>
<td>ProLine</td>
</tr>
<tr>
<td>Switzerland</td>
<td>CH</td>
<td>ch_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ch_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Colombia</td>
<td>CO</td>
<td>co_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>co_41h.prm</td>
<td>D/xxH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>co_p2v.prm</td>
<td>ProLine</td>
</tr>
<tr>
<td>Germany</td>
<td>DE</td>
<td>de_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>de_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Denmark</td>
<td>DK</td>
<td>dk_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dk_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Spain</td>
<td>ES</td>
<td>es_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>es_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Finland</td>
<td>FI</td>
<td>fi_300.prm</td>
<td>D/300SC</td>
</tr>
</tbody>
</table>
## 8. Country-Specific Parameters

<table>
<thead>
<tr>
<th>Country</th>
<th>Country Code</th>
<th>File Name</th>
<th>Boards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finland</td>
<td>FI</td>
<td>fi_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fi_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>France</td>
<td>FR</td>
<td>fr_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fr_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Greece</td>
<td>GR</td>
<td>gr_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Hungary</td>
<td>HU</td>
<td>hu_300.prm</td>
<td>D/300SC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hu_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td>Ireland</td>
<td>IE</td>
<td>ie_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Israel</td>
<td>IL</td>
<td>il_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td>India</td>
<td>IN</td>
<td>in_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in_41h.prm</td>
<td>D/xxH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in_p2v.prm</td>
<td>ProLine</td>
</tr>
<tr>
<td>Italy</td>
<td>IT</td>
<td>it_300.prm</td>
<td>D/300SC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>it_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>it_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Japan</td>
<td>JP</td>
<td>jp_160.prm</td>
<td>D/160SC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jp_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jp_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jp_41h.prm</td>
<td>D/xxH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jp_p2v.prm</td>
<td>ProLine</td>
</tr>
<tr>
<td>Korea</td>
<td>KR</td>
<td>kr_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>LU</td>
<td>lu_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Morocco</td>
<td>MA</td>
<td>ma_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td>Mexico</td>
<td>MX</td>
<td>mx_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mx_41h.prm</td>
<td>D/xxH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mx_p2v.prm</td>
<td>ProLine</td>
</tr>
<tr>
<td>Malaysia</td>
<td>MY</td>
<td>my_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Netherlands</td>
<td>NL</td>
<td>nl_300.prm</td>
<td>D/300SC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nl_41d.prm</td>
<td>D/xxD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nl_41e.prm</td>
<td>D/xxE</td>
</tr>
<tr>
<td>Norway</td>
<td>NO</td>
<td>no_300.prm</td>
<td>D/300SC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_41d.prm</td>
<td>D/xxD</td>
</tr>
</tbody>
</table>
You can choose to install the Country Specific Parameter file when you install the System Release software. The files for the country you select during installation are placed in the same subdirectory where the Dialogic firmware files reside (by default, the `<install drive>:\<install directory>\dialogic\data subdirectory`). The Dialogic Configuration file is automatically updated.
8. Country-Specific Parameters

8.4. Firmware Parameter Features

Depending upon your platform, Country Specific Parameters are selected in a different way. If you are using Intel, select Country Specific Configuration from the Dialogic System Software program group. If you are using Digital AlphaServer, click the Country button on the main window of the Dialogic Configuration Manager (DCM). Parameter feature names are stored in the Dialogic Configuration file to selectively download parameters from the Country Specific Parameter files.

The following features are supported:

- **PROT_MERCURYCAS** - The board will use the Mercury Channel Associated signaling instead of the default BT CallStream signaling. This feature is used only for E-1 digital frontend boards in the United Kingdom (such as D/300SC-E1).

- **SIG_ER** - The D/41ESC board will use Earth Recall signaling instead of the default Hook Flash. This feature is used mostly in European countries.

- **PPS_20** - The board will use 20PPS for pulse dialing instead of the default 10PPS. This feature is used only in Japan.

- **RXGAIN_P1** - The D/41ESC or D/160SC-LS board will have a positive receive gain of +1 dB instead of the default value of 0 dB. This feature is currently available in the United States and Japan only.

- **RXGAIN_P2** - The D/41ESC or D/160SC-LS board will have a positive receive gain of +2 dB instead of the default value of 0 dB. This feature is currently available in the United States and Japan only.

- **RXGAIN_P3** - The D/41ESC or D/160SC-LS board will have a positive receive gain of +3 dB instead of the default value of 0 dB. This feature is currently available in the United States and Japan only.

- **RXGAIN_N1** - The D/41ESC or D/160SC-LS board will have a negative receive gain of -1 dB instead of the default value of 0 dB. This feature is currently available in the United States and Japan only.

- **RXGAIN_N2** - The D/41ESC or D/160SC-LS board will have a negative receive gain of -2 dB instead of the default value of 0 dB. This feature is currently available in the United States and Japan only.
• RXGAIN_N3 - The D/41ESC or D/160SC-LS board will have a negative receive gain of -3 dB instead of the default value of 0 dB. This feature is currently available in the United States and Japan only.

8.5. Intelligent Network Interfaces Products

The features listed below can be selected using the Country Specific Configuration program. This program is accessed differently depending on which platform you are using. If Intel, select Country Specific Configuration from the Dialogic System Software program group. If Digital AlphaServer, click the Country button on the main window of the Dialogic Configuration Manager (DCM).

• Clear Channel TS16 is supported. This feature allows the use of time slot 16 for data on E-1 interface boards. The Country Specific Configuration program will add one of the following lines to your configuration.

  FEATURES = TS16_CLEAR
  Selects Clear Channel Time Slot 16 (CCTS16) for E-1 interface boards (DTI/300SC, DTI/301SC, D/300SC-E1); ignores E1 signaling received from the network on time slot 16 and transmits FFH.

  FEATURES = TS16_SIG
  The E-1 interface board will use the default of E-1 signaling on time slot 16.

• Selectable Dual-Tone Resolution is supported. This feature includes a download parameter that changes the dual-tone resolution for D/41ESC voice boards. By default a dual-tone resolution of 62.5 Hz is used. You may select a dual-tone resolution of 125 Hz instead. A dual-tone resolution of 62.5 hz means that the 2 tones in the dual tone must be at least 62.5 Hz apart to be detected by Global Tone Detection as dual tone.

NOTE: This parameter applies to D/41ESC installations only.

  The following line is automatically added to your configuration to change the dual tone frequency to 125 Hz:

  FEATURES = FREQRES_LOW
8. Country-Specific Parameters

8.6. Modifying Network Parameters

When modifying the default network parameters, ensure that you perform the following:

- If you have Country-Specific Parameter software installed, modify your parameter file which is located in the \<install drive>\<install directory>\dialogic\data. The file name is in the form of xx_240.prm or xx_300.prm, where xx is the country code. For example, the parameter file for Australia is au_240.prm or au_300.prm.

- If you do not have Country-Specific Parameter software installed, edit the SPANDTI.PRM file which is located in the \<install drive>\<install directory>\dialogic\data directory.
Dialogic System Software and SDK for Windows NT DNA Release Reference
Index

A
accessing Caller ID information
ACLIP, 36
CLASS, 36
CLIP, 36
ACLIP, 34
accessing Caller ID information, 36
MDMF, 34
Message Types, 49
Multiple Data Message (MDM)
Format, 35
SDMF, 34
Single Data Message (SDM)
Format, 35
ag_getctinfo( ), 1
ag_getxmitslot( ), 1
ag_listen( ), 1
ag_unlisten( ), 1
Analog Calling Line Identity
Presentation, 34
ANI on demand, 88
audio jacks, 81
C
C language interfaces, 4
c_a_pamd_spdval field, 32
cadence, distinctive ringing, 92
Call Analysis Parameter structure
DX_CAP, 32
Call by call service selection, 88
call progress features
PerfectCall, 31
Positive Answering Machine
Detection (PAMD), 31
Positive Voice Detection (PVD), 31
silence detection, 31
Caller ID, 34
accessing information
ACLIP, 36
CLASS, 36
common Message Types, 46
demonstration programs, 39
enabling channels, 38
error handling, 38
formats, 34
ACLIP, 34
MDMF, 34
SDMF, 34
CLASS, 34
MDM, 34
SDM, 34
CLIP, 34
Caller ID functions, 39
dx_gtcid( ), 40
dx_gtextcid( ), 45
dx_wtcid( ), 55
Caller ID related references, 58
Calling Line Identity Presentation, 34
CLASS, 34
accessing Caller ID information, 36
MDM, 34
Message Types, 48
Multiple Data Message (MDM)
Format, 35
SDM, 34
Single Data Message (SDM)
Format, 35
CLIP, 34
accessing Caller ID information, 36
Message Types, 49
common Message Types, 46
CLASS, ACLIP, and CLIP, 46
compatibility library functions, 5
dt_libinit( ), 13
dx_libinit( ), 6
fx_libinit( ), 8
sr_libinit( ), 11
vr_libinit( ), 15
Compelled Tone Protocol (R2MF), 32
country-specific parameter system, 111
countries supported, 112
Dialogic boards supported, 114
features carried forward, 112
firmware parameter features, 117
Intelligent Network Interfaces
products, 118
modifying network parameters, 119
new country support, 111
new GDPD support, 111
new generic 10 pulse-per-second
GDPD support, 112
Custom Local Area Signaling Services, 34
cut-through
Global DPD, 60
D
DCB/SC, 99
DG_DPD
Dial Pulse Detection
new defines, 32
dg_type, 32, 64
Dialogic Global Dial Pulse Detection, 60
Dialogic DLL Version Number
functions, 17
dt_GetDllVersion( ), 25
dx_GetDllVersion( ), 18
fx_GetDllVersion( ), 20
sr_GetDllVersion( ), 23
vr_GetDllVersion( ), 27
Dialogic TSP and WAVE
installing and removing, 106
uninstalling, 109
Dialogic TSP for TAPI
activating, 107
deactivating, 109
Dialogic WAVE driver
activating, 108
deactivating, 109
digit type recording
Dial Pulse Detection, 32
new defines for dg_type( ), 33
distinctive ring, 92
DPD. See Dial Pulse Detection. See dial
pusle detection
dt_getctinfo( ), 1
dt_GetDllVersion( ), 25
### Index

- `dt_getxmitslot()`, 1
- `dt_libinit()`, 13
- `dt_listen()`, 1
- `dt_unlisten()`, 1
- DTMF length and interdigit timing, 32
- `DV_DIGIT`, 32
- `DX_CAP`  
  - Call Analysis Parameter structure, 32
  
- `dx_filechsize()`, 3
- `dx_filecommit()`, 3
- `dx_filedup()`, 3
- `dx_filedup2()`, 3
- `dx_filefilelength()`, 3
- `dx_fileisatty()`, 3
- `dx_filelocking()`, 3
- `dx_fileopen()`, 3
- `dx_fileopen_osfhandle()`, 3
- `dx_filesetmode()`, 3
- `dx_filesopen()`, 3
- `dx_filestart()`, 3
- `dx_filestat()`, 3
- `dx_filetell()`, 3
- `dx_fileumask()`, 3
- `dx_getctinfo()`, 1
- `dx_GetDllVersion()`, 18
- `dx_getfeaturelist()`, 74
  - cautions, 74
  - description, 74
  - errors, 79
  - example, 76
- `dx_getxmitslot()`, 1
- `dx_gcallid()`, 40
- `dx_gtextcallid()`, 45
- `dx_libinit()`, 6
- `DX_LINEINDISABLE`, 84
- `DX_LINEINENABLE`, 84
- `dx_listen()`, 1
- `dx_mreciottdata()`, 31, 67
- `dx_playiottdata()`, 31
- `dx_playvox()`, 31
- `dx_playwav()`, 31
- `dx_recioottdata()`, 31
- `dx_recvox()`, 31
- `dx_recwav()`, 31
- `dx_setparm()`  
  - DTMF length and interdigit timing, 32
  - enable or disable audio jack, 84
  - enabling Caller ID channels, 38
- `dx_unlisten()`, 1
- `dx_wtcallid()`, 55
- `DXCH_AUDIOLINEIN`, 84

**E**

- enabling channels  
  - Caller ID, 38
- enabling Global DPD, 61
- error handling  
  - Caller ID, 38
F

FEATURE_TABLE, 74

file manipulation functions, 2

dx_filechsize( ), 3

dx_filecommit( ), 3

dx_filedup( ), 3

dx_filedup2( ), 3

dx_filefilelength( ), 3

dx_fileisatty( ), 3

dx_filelocking( ), 3

dx_fileopen( ), 3

dx_fileopen_osfhandle( ), 3

dx_filesetmode( ), 3

dx_filesopen( ), 3

dx_filestart( ), 3

dx_filestat( ), 3

dx_filetell( ), 3

dx_fileumask( ), 3

file Version Number, 17

ft_e2p_brd_cfg field

FEATURE_TABLE, 75

ft_fax table field

FEATURE_TABLE, 75

ft_frontend field

FEATURE_TABLE, 76

ft_misc field

FEATURE_TABLE, 76

ft_play field

FEATURE_TABLE, 75

ft_record field

FEATURE_TABLE, 75

ft_rfu field

FEATURE_TABLE, 76

ft_tone field

FEATURE_TABLE, 75

function

dx_getfeaturelist( ), 74

fx_GetDllVersion( ), 20

fx_GetFileVersionInfo( )

Win32, 17

fx_libinit( ), 8

G

GDPD. See Dialogic Global Dial Pulse Detection

Global Dial Pulse Detection

application programming for accuracy, 63

programming procedure, 63

programming for accuracy, 63

Global Tone Detection (GTD), 32

Global Tone Generation (GTD), 32

I

ISDN

benefits, 87

diagnostic tools, 88

features, 87

installation, 87

L

library interfaces, 5

Line In audio jack, 81

Line Out audio jack, 82

M

Message Type ID, 45

Message Types for ACLIP (multiple data message), 49

Message Types for CLASS (multiple data message), 48

Message Types for CLIP, 49

ms_genringex( ), 92
Index

MSI/SC distinctive ring, 92
Multiple Data Message (MDM) Format, 34
   ACLIP, 35
   CLASS, 35
Multiple Data Message Format (MDMF), 34

N
nr_scroute( ), 2
nr_scunroute( ), 2

P
PAMD. See Positive Answering Machine Detection
PerfectCall, 31
PerfectDigit (DTMF signaling and MF signaling), 32
play and record functions, 31
   dx_mreciottdata( ), 31
   dx_playiottdata( ), 31
   dx_playvoxy( ), 31
   dx_playwav( ), 31
   dx_recriottdata( ), 31
   dx_recvoxy( ), 31
   dx_recwav( ), 31
Positive Answering Machine Detection, 31, 32
Positive Voice Detection, 31
product version number, 17
ProLine/2V audio jack demonstration program, 86
ProLine/2V audio jack specifications, 82
ProLine/2V audio jacks, 81
ProLine/2V channel 2 input switching, 84

R
regional DPD parameters, 62
related references
   Caller ID, 58
retrieving Dialogic DLL Version Numbers, 17
ring cadence, distinctive, 92
run-time linking, 4

S
SCbus routing functions, 1
   ag_getctinfo( ), 1
   ag_getxmitslot( ), 1
   ag_listen( ), 1
   ag_unlisten( ), 1
   dt_getctinfo( ), 1
   dt_getxmitslot( ), 1
   dt_listen( ), 1
   dt_unlisten( ), 1
   dx_getctinfo( ), 1
   dx_getxmitslot( ), 1
   dx_listen( ), 1
   dx_unlisten( ), 1
silence detection, 31
Single Data Message (SDM) Format, 34
   ACLIP, 35
   CLASS, 35
Single Data Message Format (SDMF), 34
   sr_GetDllVersion( ), 23
   sr_libinit( ), 11
   sr_waitevt( ), 17
station distinctive ring, 92
Station functions
  ms_genringex(), 92
system features, 1

T
Talker32, 101
talk-off rejection, 63
TAPI and WAVE
  relevant features, 101
  requirements, 101, 107
  Telephony Service Provider (TSP)
    and TAPI, 101
  WAVE device driver, 103
Telephony Service Provider (TSP) and
  TAPI, 101
tone features
  Compelled Tone Protocol (R2MF), 32
  Global Tone Detection (GTD), 32
  Global Tone Generation (GTD), 32
  PerfectDigit (DTMF signaling and
    MF signaling), 32
training
  Global DPD, 60
Transaction Record, 66
  supported voice boards, 66
transaction record function, 67

U
User-to-user information, 88

V
Vari-A-Bill, 88
Version Number
  Dialogic DLL, 17
  file, 17
  product, 17

voice, 31
voice features, 31
vr_GetDllVersion(), 27
vr_libinit(), 15

W
wait for events function, 17
WAVE device driver, 103