Voice API for Linux

Demo Guide

June 2005

05-2342-002

Publication Date: June 2005

Document Number: 05-2342-002

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

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

For **Products and Services Information**, visit the Intel Telecom Products website at:
http://www.intel.com/design/network/products/telecom

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

Contents

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2342-002	June 2005	Demo Descriptions chapter: Added custserv, dpddemo, and d40demo. Also added information about the d4xtools.c toolkit. Running the Demos chapter: Added sections for custserv, dpddemo, and d40demo. Demo Details chapter: Added sections for custserv, dpddemo, and d40demo.
05-2342-001	November 2004	Initial version of document.

About This Publication

The following topics provide information about the *Voice API for Linux Demo Guide*:

Purpose

This publication describes the voice demonstration programs and provides instructions for running the demos on the Linux* operating system.

Applicability

This document version (05-2342-002) is published for Intel® Dialogic® System Release 6.1 for Linux.

This document may also be applicable to later Intel Dialogic system releases on Linux. Check the Release Guide for your software release to determine whether this document is supported.

Intended Audience

This publication is written for users of the voice demonstration programs which may include the following:

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

How to Use This Publication

This publication assumes that you understand computer telephony terms and concepts, and are familiar with the Linux operating system and the C programming language.

The information in this guide is organized as follows:

- Chapter 1, "Demo Descriptions" provides a brief overview of the voice demos.
- Chapter 2, "System Requirements" discusses the hardware and software required to run the demos.
- Chapter 3, "Preparing to Run the Demos" lists the procedures you must follow before running the demos.
- Chapter 4, "Running the Demos" describes the steps to run the demos, the demo options, and how to stop the demo.
- Chapter 5, "Demo Details" provides additional information about the demos, such as the files used by the demos.

Related Information

See the following documents for more information:

- For information about voice library features and guidelines for building applications using Voice software, see the *Voice API Programming Guide*.
- For details on all voice functions, parameters, and data structures in the voice library, see the *Voice API Library Reference*.
- For information about the SRL and programming models supported, see the *Standard Runtime Library API Programming Guide*.
- For information on the system release, system requirements, software and hardware features, supported hardware, and release documentation, see the Release Guide for your system release.
- For details on known problems and late-breaking updates or corrections to the release documentation, see the Release Update. Be sure to check the Release Update for the software release you are using for any updates or corrections to this publication. Release Updates are available on the Telecom Support Resources website at *http://resource.intel.com/telecom/support/documentation/releases/index.htm*

Demo Descriptions 1

This chapter provides a brief description of the voice demonstration programs.

The following demonstration programs are provided with the voice software. These demos are supported on Springware boards only.

cbansr
: Voice application that uses the asynchronous callback programming model. This demo is state-driven but uses event handlers that can be enabled or disabled for specific events on specific devices. The demo illustrates the voice recording and playback feature. You can listen to a prompt, record a message, and play back that message.

pansr
: A multithreaded, multi-channel voice application that uses the asynchronous polled programming model. This demo is state-driven and uses polling to get event information without the use of event handlers or device handlers. The demo illustrates the voice recording and playback feature. You can listen to a prompt, record a message, and play back that message.

custserv
: Synchronous mode, customer service application that transfers calls to the proper service representative based on the caller's input.

dpddemo
: A multi-channel voice application that uses the asynchronous polled programming model. The demo illustrates the voice recording and playback feature. It is similar to the pansr demo, but can accept both dial pulses and DTMF tones.

d40demo
: Synchronous mode, simple order entry application built on a menu system.

The purpose of the demo programs is to show how to use the voice library functions in a voice application. They provide a framework on which to base your applications; they are not intended to be used as real voice applications.

Source code and an executable version of the demo programs are provided, as well as a makefile to compile the source. The source is written in C.

A toolkit of general purpose C routines is also included with the demo programs. This toolkit aids in the development of voice applications. The toolkit source is contained in *d4xtools.c*.

Note: The synchronous programming model is not recommended for production applications; however, it can be used for demo or proof of concept applications. For detailed information on programming models, including advantages and disadvantages of each model, see the *Standard Runtime Library API Programming Guide*.

System Requirements 2

The requirements for running the voice demos are described in the following sections:

2.1 Hardware Requirements

To run a voice demo, you need the following:

- A Springware board that supports the R4 voice API library installed in a computer system (for a list of supported Springware boards, see the release guide for the system release)
- A central office simulator OR a connection to a telephone switch or public telephone network
- For boards with a digital network interface, a T1 or an E1 line connected to your system.
- For boards with an analog interface, an analog line such as a connection to a PBX.
- A telephone

For other system configuration requirements, such as memory requirements, see the release guide for your system release.

The following is an example of a hardware setup for use with cbansr or pansr demo:

- One D/41JCT-LS board
- One central office simulator
- One analog phone

Connect the phone to Line 1A of the simulator and connect Line 2 of the simulator to the desired channel on the back of the board.

2.2 Software Requirements

To run a voice demo, you need the Intel® Dialogic® System Release. Be sure to select the package for Intel Dialogic Boards (Springware boards) when installing the system release. For details on installing the system release, see the software installation guide for your system release.

For a list of operating system requirements and supported compilers, see the release notes for your system release.

Preparing to Run the Demos 3

This chapter provides information on preparations to follow before running a voice demo.

Before you run a voice demo program, ensure that you have performed the following:

1. Adhered to the hardware and software requirements described in Chapter 2, "System Requirements".

2. Installed a Springware board in your computer system. This product should support the R4 API libraries.

3. Started System Services for the board.

4. Connected the board to a central office simulator, telephone switch or the public telephone network.

5. Installed a telephone extension near your computer through which you can dial extensions assigned to the board.

Running the Demos 4

This chapter describes how to run the voice demos. Topics include:

4.1 Starting the Demo

The following sections provide information on starting the voice demos.

- Starting the cbansr demo
- Starting the pansr demo
- Starting the custserv demo
- Starting the dpddemo
- Starting the d40demo

4.1.1 Starting the cbansr demo

To run the cbansr demo, follow these instructions:

1. Open a command prompt window and go to the directory where the demo is located. The default location is the */demos/dx_demos/ansr* directory under INTEL_DIALOGIC_DIR (the environment variable for the directory in which the system release software was installed).

2. At the command prompt, type

   ```
   ./cbansr -<option>
   ```

   For a list of command line options, type

   ```
   ./cbansr -?
   ```

   Note: On Linux, the demo name is case-sensitive and must be entered as shown.

   The demo options are described in Table 1, "Voice Demo Options", on page 15. If you don't specify an option, default options are assumed.

4.1.2 Starting the pansr demo

To run the pansr demo, follow these instructions:

1. Open a command prompt window and go to the directory where the demo is located. The default location is the */demos/dx_demos/ansr* directory under INTEL_DIALOGIC_DIR (the environment variable for the directory in which the system release software was installed).

2. At the command prompt, type

   ```
   ./pansr -<option>
   ```

   For a list of command line options, type

   ```
   ./pansr -?
   ```

   Note: On Linux, the demo name is case-sensitive and must be entered as shown.

   The demo options are described in Table 1, "Voice Demo Options", on page 15. If you don't specify an option, default options are assumed.

4.1.3 Starting the custserv demo

To run the custserv demo, follow these instructions:

1. Open a command prompt window and go to the directory where the demo is located. The default location is the */demos/dx_demos/custserv* directory under INTEL_DIALOGIC_DIR (the environment variable for the directory in which the system release software was installed).

2. At the command prompt, type

   ```
   ./custserv <device name list>
   ```

   where device name list is a list of one or more channels that the demo will use. For example:

   ```
   ./custserv dxxxB1C1 dxxxB1C2
   ```

   Note: On Linux, the demo name is case-sensitive and must be entered as shown.

4.1.4 Starting the dpddemo

To run the dpddemo, follow these instructions:

1. Open a command prompt window and go to the directory where the demo is located. The default location is the */demos/dx_demos/dpddemo* directory under INTEL_DIALOGIC_DIR (the environment variable for the directory in which the system release software was installed).

2. At the command prompt, type

   ```
   ./dpddemo -<option>
   ```

   For a list of command line options, type

   ```
   ./dpddemo -?
   ```

Note: On Linux, the demo name is case-sensitive and must be entered as shown.

The demo options are described in Table 1, "Voice Demo Options", on page 15. If you don't specify an option, default options are assumed.

4.1.5 Starting the d40demo

To run the d40demo, follow these instructions:

1. Open a command prompt window and go to the directory where the demo is located. The default location is in the */demos/dx_demos/d40demo* directory under INTEL_DIALOGIC_DIR (the environment variable for the directory in which the system release software was installed).

2. At the command prompt, type

   ```
   ./d40demo <device name list>
   ```

   where device name list is a list of one or more channels that the demo will use. For example:

   ```
   ./d40demo dxxxB1C1 dxxxB1C2
   ```

   Note: On Linux, the demo name is case-sensitive and must be entered as shown.

4.2 Demo Options

The format for specifying options at the command prompt is:

```
[demo name] -<option1> -<option2> -<optionn>
```

where *n* represents the last option.

Table 1 illustrates options for the cbansr, pansr, and dpddemo demos. (Other demos do not use options.) These options are specified from the command line. If an option is not specified when running a demo, a default value applies.

Notes: ***1.*** The order in which you specify demo options is not important.

2. In the table, N represents an integer.

Table 1. Voice Demo Options

Demo Option	Default Value	Description/Value
-?	N/A	Displays list of command line arguments or options.
-dN	N/A	Specifies the number of the first D/4x board to use. Example: -d1.
-tN	N/A	Specifies the number of the first digital network interface board to use. Example: -t2.

Table 1. Voice Demo Options

Demo Option	Default Value	Description/Value
-nN	N/A	Specifies the number of D/4x channels to use. Values are: 1 to 12. Example -n4.
-f<value>	N/A	Specifies the front end interface. Values are: Analog, T1, and E1. Example: -fAnalog.

Examples

To run the cbansr demo on a D/41JCT-LS board, specify the following:

```
./cbansr -d1 -n4 -fAnalog
```

To run the pansr demo on a D/120JCT-LS board using four channels, specify the following:

```
./pansr -d1 -n4 -fAnalog
```

4.3 Using the Demo

The following topics discuss how to run and use the voice demos:

- Using the cbansr demo
- Using the pansr demo
- Using the custserv demo
- Using the dpddemo
- Using the d40demo

4.3.1 Using the cbansr demo

The following procedure describes the activity after starting the cbansr demo:

1. After the demo starts, a screen is displayed with the status of the channels and calls.
2. Dial an extension number configured for one of the voice channels.
3. After the voice prompt, leave a brief message and hang up the telephone. This message is recorded in *messagen.vox* file, where *n* represents the channel number.
4. Redial the same extension.
5. During the voice prompt, enter the 4-digit access code using the telephone keypad. This access code takes the form *n*234, where *n* is the last digit of the voice channel number.
6. The system plays the previously recorded message.

7. Hang up the telephone.

8. Press Ctrl-C to terminate the demo.

Note: If you do not terminate the demo using Ctrl-C, the driver for the demo does not unload after quitting the program, necessitating a reboot before another demo can be loaded.

4.3.2 Using the pansr demo

The following procedure describes the activity after starting the cbansr demo:

1. After the demo starts, a screen is displayed with the status of the channels and calls.

2. Dial an extension number configured for one of the voice channels.

3. After the voice prompt, leave a brief message and hang up the telephone. This message is recorded in *messagen.vox* file, where *n* represents the channel number.

4. Redial the same extension.

5. During the voice prompt, enter the 4-digit access code using the telephone keypad. This access code takes the form *n*234, where *n* is the last digit of the voice channel number.

6. The system plays the previously recorded message.

7. Hang up the telephone.

8. Press Ctrl-C to terminate the demo.

Note: If you do not terminate the demo using Ctrl-C, the driver for the demo does not unload after quitting the program, necessitating a reboot before another demo can be loaded.

4.3.3 Using the custserv demo

The following procedure describes the activity after starting the custserv demo:

1. After the demo starts, you are presented with the initial menu of options via voice prompt. Select one of the options. For example, press 1 for hardware support, press 2 for software support, or press 3 for service and maintenance.

2. If you press 1 from the main menu, a submenu of options for hardware support is played. Select one of the options. For example, press 1 for PC support.

3. If you press 2 from the main menu, a submenu of options for software support is played. Select one of the options. For example, press 1 for database support.

4. If you press 3 from the main menu, or any number from the submenus, the demo plays a message to the caller. The message simulates a customer service representative and explains that, if this had been a real application, a live representative would have answered.

5. Press Ctrl-C to terminate the demo.

Note: If you do not terminate the demo using Ctrl-C, the driver for the demo does not unload after quitting the program, necessitating a reboot before another demo can be loaded.

4.3.4 Using the dpddemo

The following procedure describes the activity after starting the dpddemo:

1. After the demo starts, a screen is displayed with the status of the channels and calls.
2. Dial an extension number configured for one of the voice channels.
3. After the voice prompt, leave a brief message and hang up the telephone. This message is recorded in *messagen.vox* file, where *n* represents the channel number.
4. Redial the same extension.
5. During the voice prompt, enter the 4-digit access code using the telephone keypad. This access code takes the form *n*234, where *n* is the last digit of the voice channel number.
6. The system plays the previously recorded message.
7. Hang up the telephone.
8. Press Ctrl-C to terminate the demo.

Note: If you do not terminate the demo using Ctrl-C, the driver for the demo does not unload after quitting the program, necessitating a reboot before another demo can be loaded.

4.3.5 Using the d40demo

The d40demo program creates a background process for each channel that was specified as an argument. The main process terminates and you are left with one background process per channel.

Each background process waits to receive a call on its channel. When a call is received, the phone is taken off-hook after one ring, and an introduction message is played. The caller is presented with a menu of four choices:

- place an order by pressing 1
- check an order by pressing 2
- cancel an order by pressing 3
- end the call by pressing *

The caller is prompted for a response. An invalid response prompts the caller to try again. A valid response prompts the caller to complete the action he chose.

4.4 Stopping the Demo

Press Ctrl-C at any time to exit a voice demo. All channels and files are properly closed by the demo.

Demo Details 5

This chapter provides more detail about the voice demos. You do not need this information to run the demos successfully.

5.1 Files Used by cbansr and pansr

Table 2 lists the files that are used by the cbansr and pansr demos.

Table 2. Files Used by the cbansr and pansr Demos

File Name	Purpose
answer.h	Header file
cbansr	Executable file
cbansr.c	Source code for cbansr demo
display.c	Source code for the screen display for cbansr and pansr demos
GOODBYE.VOX	Goodbye voice file
INTRO.VOX	Greeting voice file
INVALID.VOX	Invalid access code voice file
messagen.vox	Recording of your message, where n is an integer that represents the channel number (for example, message1.vox)
pansr	Executable file
pansr.c	Source code for pansr demo

5.2 Files Used by custserv

Table 3 lists the files that are used by the custserv.

Table 3. Files Used by the custserv

File Name	Purpose
custserv	Executable file
custserv.c	Source code
custserv.h	Header file
CUSTSERV.VOX	Greeting voice file

5.3 Files Used by dpddemo

Table 4 lists the files that are used by the dpddemo.

Table 4. Files Used by the dpddemo

File Name	Purpose
answer.h	Header file
display.c	Source code for the screen display
dpddemo	Executable file
dpddemo.c	Source code
example.c	Source code
GOODBYE.VOX	Goodbye voice file
INTRO.VOX	Greeting voice file
INVALID.VOX	Invalid access code voice file

5.4 Files Used by d40demo

Table 5 lists the files that are used by the d40demo.

Table 5. Files Used by the d40demo

File Name	Purpose
d40demo	Executable file
d40demo.c	Source code containing application-specific routines
d4xtools.c	Source code containing system routines
date.h	Header file
digits.h	Header file
menu.h	Header file
prompt.h	Header file
DATE.VOX	Voice file

Table 5. Files Used by the d40demo

File Name	Purpose
DIGITS.VOX	Voice file
PROMPT.VOX	Voice file

5.5 Supplementary Information for d40demo

Supplementary information on the d40demo is given in the following topics:

- Source Code Overview
- Global Variables
- Initialization Routine
- Menu System Routine
- Messaging System Routine

5.5.1 Source Code Overview

The source code to d40demo is located in two modules, *d40demo.c* and *d4xtools.c*. As an introduction to the program, a simplified outline of d40demo is shown:

- Initialization
  - Spawn subprocess for each channel
  - End parent process
  - Initialize interrupt handlers
  - Open channel and files
  - Initialize process variables
- Main Program Body
  - Place channel onhook
  - Wait for one ring
  - Go offhook
  - Play introduction message
  - Start menu routine loop
    - Play menu prompt
    - Get caller response
    - Process caller response
  - End loop

The outline provides a general approach to any voice application, not only d40demo. For example, all voice applications need some type of initialization routine to open the channel, initialize variables, and open files. A voice menu system is also a universal feature found in most voice applications because a caller needs a vocal prompt to instruct him on his choice of actions. These routines are called system routines and are in the *d4xtools.c* module.

Since d40demo is also a working order entry system, the source code contains nonsystem routines that are specific to the order entry application. The *d40demo.c* module contains the routines that manage a simple database, which are called the application specific routines.

The d40demo program uses three system routines:

- Initialization
- Menu System Routine
- Messaging System Routine

5.5.2 Global Variables

Global variables are used by the functions in *d4xtools.c*. These variables are defined in *d4xtools.c*:

cur_menu
: Current Menu. A pointer to the current active menu.

pre_menu
: Previous Menu. A pointer to the previous menu.

idlestate
: Idle State Reentry Environment. The reentry point to the process. Using the **longjmp( )** and **setjmp( )** Linux functions, idlestate is the point where the process sets the hookstate to onhook and waits for the phone to ring.

devhandle
: Device handle. The variable that holds the descriptor for the channel.

Two default DV_TPT arrays are defined in *d40demo.c* and are used by the *d4xtools.c* system routines:

def_rp_tpt
: Default Record and Play. The default DV_TPT array for recording and playing using **dx_rec( )** and **dx_play( )**.

def_dg_tpt
: Default Get Digit. The default DV_TPT array for getting digit responses using the **dx_getdig( )** function.

5.5.3 Initialization Routine

When d40demo is started, the application is initialized by a call in **main( )** to **init_sys( )**. The **init_sys( )** function is located in the *d4xtools.c* module. It performs system initialization by spawning a child process for each channel, configuring the signal handler, opening the channel, initializing the message table, and ending the parent process.

For every argument supplied at the command line, the **init_sys( )** function uses the Linux system call **fork( )** to spawn a child copy of the d40demo process. Any code following the call to **fork( )** is executing in both the child and the parent process. The following code fragment checks if the process is a child process, and if it is, closes file descriptor 0, the standard input:

```
if (rc == 0) {
  close(0);
     .
     .
```

The code that follows the **close( )** function call within the if statement is only executed in the child process. The parent process fails the (rc == 0) comparison and loops back to the while statement. When all the arguments are exhausted, the parent process eliminates itself by a call to the **exit( )** function.

The **signal( )** function calls redirect the signals from the outside world to call the **sigcatch( )** function. The **sigcatch( )** function, located in *d4xtools.c*, is the signal processing routine for each channel. It stops I/O on the channel, sets the hook state to onhook, and exits the process. By intercepting the signals and using **sigcatch( )**, the process exits cleanly if it receives an interrupt, quit, terminate, or hang-up signal from the outside world.

The **strcpy( )** function copies the channel device name to a global variable to give all functions access to the device name.

The channel specified on the command line is opened by the **dx_open( )** function with read and write privileges. The descriptor for the channel is stored in the global variable devhandle.

The message structure is initialized in the for-loop. It scans every entry in the message structure, opens the specified file with read-only privileges, and stores the file descriptor for the open file in the message structure. This code allows an arbitrary number of playback files or devices to be opened. You only need to specify a DL_MSGTBL structure for each playback file or device, and the **init_sys( )** function will open all the files and store the file descriptors.

The final action of a child process is to return the argument number as the return value. *d40demo.c* uses this value when it initializes the order entry database.

5.5.4 Menu System Routine

d40demo is built around a menu system that controls the flow of the program. The menu system is a general concept that is useful for any voice application. Understanding how the menu system works is essential to understanding how d40demo works.

Model of the Menu System

In general, when a call is received by a voice application, all callers are presented with the same initial voice menu. This menu is referred to as the root menu. For example, in a customer service application, the initial prompt might introduce the company and ask the caller to press 1 for sales or 2 for support. The caller listens to the voice prompt and presses 1. The application prompts with another menu, asking the caller to press 1 for Jane, 2 for John, or 3 to hang up. The caller presses 2 and is prompted to record a message to John. When the caller completes recording he is prompted with the same menu asking him to press 1 for Jane, 2 for John, or 3 to hang-up. The caller presses 3 and is disconnected.

From the example above, it can be seen that all menus have the following characteristics. They play voice prompts, require caller input, and transfer control based on caller input.

When input is received from the caller, four things can happen: control is passed to another menu; an action is performed; an action is performed and control is passed to another menu; an error message is played (invalid input).

All menus provide the caller with options. In the example above, the first caller response transferred the caller to another menu. The second response performed an action (recording a message) and prompted the caller with the same menu. In another case, the caller might have been transferred to a different menu.

A menu must obey the following rules:

- A menu must always transfer control to a menu. This can be a different menu or the same menu.
- A menu is not required to perform an action. If the menu does perform an action, it still must pass control to some menu.

Implementation

The menu system is implemented by using three data structures, namely DL_MENU, DL_MNUOPTS, and DL_DATA, and three functions, namely **menu_engine( )**, **gt_data( )**, and **val_menu( )**. These functions are defined in *d4xtools.c*. By defining the structures and using the functions, a complete menu system for any voice application can be implemented.

To see how a menu system can be implemented, examine the code that defines d40demo's main menu as examples. The definition for d40demo's only menu is located in the variables menu_1, menu1_opts, and data_1, which are declared in *d40demo.c*.

DL_MENU Data Structure

The DL_MENU structure defines the menu. It uses a DL_MNUOPTS structure to define the response options and a DL_DATA structure to define the termination conditions for a response.

```
typedef struct DL_MENU {

   DL_MNUOPTS  * me_opttbl;        /* Pointer to menu's option table         */
   DL_MENU     * me_defmenu;       /* Default next menu for this menu        */
   DL_MNUOPTS  * (*me_validfnc)(); /* Pointer to menu's validation function  */
   char        * me_prompt;        /* Pointer to menu's prompt message name  */
   char        * me_retry;         /* Pointer to menu's retry message name   */
   void        (*me_error)();      /* Pointer to menu's error message name   */
   DL_DATA     * me_data;          /* Pointer to Data table for this menu    */

} DL_MENU;
```

The fields are described as follows:

me_opttbl
: Option Table - Pointer to a DL_MNUOPTS structure that defines the options for this menu.

me_defmenu
: Default Menu - Pointer to the default next-menu. The menu system will transfer control to the menu specified in this field after an action has taken place, but only if the mn_nxtmenu structure specified in the option table is NULL. This field can contain a pointer to any valid DL_MENU structure. In addition, the menu.h header file defines two DL_MENU structures

that can be used, DL_PREV and DL_CURR. DL_PREV returns the caller to the previous menu, and DL_CURR returns the caller to the current menu.

me_validfnc
: Validation Function - Pointer to the function that validates the caller's response. This function returns a DL_MNUOPTS structure containing the response that was chosen.

me_prompt
: Prompt Message - Pointer to the voice prompt message for this menu.

me_retry
: Retry Message - Pointer to the retry prompt message for this menu. This message is played if the caller response is not found in the option table.

me_error
: Error Function - Pointer to the function that will handle a fatal error occurring during this menu. Fatal errors occur when a valid response is not given within the maximum number of retries. The maximum number of retries is set by the DL_DATA structure associated with this menu.

me_data
: Data Table - Pointer to a DL_DATA structure that defines the termination conditions for getting data during this menu.

The DL_MENU structure implements the functions a menu must perform. A menu needs to:

- provide options to the caller
- prompt the caller to input an option
- validate the caller input
- perform an action based on input
- provide a prompt if the caller makes a mistake
- provide a way out if a fatal error occurs
- transfer control to some other menu or itself when everything is done

The fields of a DL_MENU structure provide the means to specify all these functions. The root menu for d40demo is defined in a DL_MENU structure with the variable name menu_1. The comments explain what each value indicates.

```
DL_MENU menu_1 = {
           menu1_opts,     /* The option table for menu 1 is in menu1_opts     */
           DM_CURR,        /* Return to current menu after completing action   */
           val_menu,       /* val_menu( ) is the input validation function     */
           "mm_menu",      /* mm_menu is the main menu voice prompt            */
           "mm_retry",     /* mm_retry is the main menu retry prompt           */
           goodbye,        /* goodbye( ) is the error function                 */
           &data_1         /* The termination condition are in data_1          */
           };
```

DL_MNUOPTS Data Structure

The DL_MNUOPTS structure defines the caller's response options for a menu. It specifies the digits a caller can enter and whether the digits indicate an action or another menu. The options for a

menu are implemented as a NULL-terminated array of DL_MNUOPTS structures. An array of DL_MNUOPTS structures allows a menu to have an arbitrary amount of options.

```
typedef struct DL_MNUOPTS {
   char       * mo_rspstr;             /* Response string to this option             */
   void       (*mo_fcnt)();            /* Next function for this option              */
   DL_MENU     * mo_nxtmenu;  /* Next menu for this option                           */
} DL_MNUOPTS;
```

The fields are defined as follows:

mo_rspstr
: Response String - This is the digit string that the caller must press to initiate this action.

mo_fcnt
: Next Function - This is the name of the void function that will be called if the digits in mn_rspstr were received.

mo_nxtmenu
: Next Menu - This is a pointer to the next menu. If the digit specified by mn_rspstr transfers control to another menu, this field points to that menu. If this field is NULL the control is transferred to the menu pointed to by me_defmenu in the DL_MSG structure that contains the option table.

The options available in a menu are:

- transfer control to another menu
- execute a function
- do both
- do neither

All menus must provide a pointer to the next menu at one of two levels. The next menu can be tied to the response and be specified in the mn_nxtmnu field of the DL_MNUOPTS structure. If this field is NULL, the next menu defaults to a higher level. The default next menu for all options in a menu is defined in the me_defmenu field of the DL_MENU structure. The action specified in the mn_fcnt field of the DL_MNUOPTS structure isn't required for every menu option.

The next menu for a response must be specified as either a next menu (mo_nxtmnu) in the DL_MNUOPTS structure or the default menu (mo_fcnt) in the DL_MENU structure for all the options not specifying a next menu.

The menu1_opts variable is the NULL-terminated array for the root menu in *d40demo.c*. It is initialized as:

```
DL_MNUOPTS menu1_opts[] = {
   {"1",enterord,NULL},
   {"2",chekord,NULL},
   {"3",canord,NULL},
   {"*",goodby,NULL},
   {NULL,NULL,NULL}
};
```

All the options for d40demo use the default next menu. The four possible inputs for d40demo (1,2,3, or *) are defined in the menu1_opts array. For example, the first table entry in menu1_opts defines the following: if the caller presses the digit 1, call the void function enterord( ).

DL_DATA Data Structure

The DL_DATA structure defines the termination conditions for getting the response to the menu options. It holds the values for several fields that map to corresponding DV_TPT elements used when getting a caller's response. These are used with the global termination conditions located in def_dg_tpt.

```
typedef struct DL_DATA {
   unsigned short   da_recvdig;      /* Number of digits to receive          */
   unsigned short   da_time;         /* Maximum time limit                   */
   char             * da_digit;      /* Terminating digit string  */
   unsigned short   da_numretry;     /* Number of retries before giving up */
}DL_DATA;
```

The fields are defined as follows:

da_recvdig
: Receive Digits - The maximum number of digits to receive.

da_time
: Time Limit - The amount of time a caller has to respond to the voice prompt. (100 ms units)

da_digit
: Digit String - A string of specific digits, any one of which will terminate the I/O when received.

da_numretry
: Number of Retries - The number of retries the caller has to press a correct digit.

In *d40demo.c*, the variable data_1 defines the termination conditions for the root menu. The comments explain what each value indicates.

```
DL_DATA data_1 ={
          1,     /* Read only 1 digit                        */
          100,   /* Allow 10 seconds for entering it         */
          NULL,  /* No termination digits needed             */
          3      /* Allow 3 retries for entry                */
          };
```

menu_engine( ) function

The **menu_engine( )** function is defined in the *d4xtools.c* module. Its purpose is to control flow of the program by using the menu tables.

```
void   menu_engine( )

{
   DL_MNUOPTS   * rspentry;
   DL_MENU      * tmp;
   char         digbuf[2];

   /* The following is the main menuing system engine. Its purpose
    * is to drive the application as defined by the menu pointed to by
    * cur_menu.
    */
   while(1) {

      /*
       * Execute the response parser
```

```
        */
       rspentry=(DL_MNUOPTS *)gt_data(digbuf,cur_menu->me_validfnc,cur_menu->me_prompt,
                              cur_menu->me_retry, cur_menu->me_error,cur_menu->me_data);

        /*
          * Execute the appropriate actions for the last response
          */

         /* If a function is required execute it   */
        if(rspentry->mo_fcnt != NULL) {
          (void) (*rspentry->mo_fcnt)();
        }

        /*
         * If a new menu has been specified then switch to next menu
         * else switch to the default menu
         */

        if(rspentry->mo_nxtmenu != NULL) {
          /*
           * Make the current menu previous
           * and the previous menu current
           */
         pre_menu = cur_menu;
         cur_menu = rspentry->mo_nxtmenu;

      } else {

         /* Switch to the default menu */
         switch((int)cur_menu->me_defmenu) {
         case NULL:
            fprintf(stderr,"Missing default menu. Menu system error");
            exit(1);

         case (int)DM_PREV:
           /* Swap the current menu with the previous menu */
           tmp = pre_menu;
           pre_menu = cur_menu;
           cur_menu = tmp;
           break;

         case (int)DM_CURR:
           break;

         default:
           pre_menu = cur_menu;
           cur_menu = cur_menu->me_defmenu;
           break;
         }
       }
     }
   }
```

In the while-loop, the first function call is to the system routine **gt_data( )** (see the next section). The **gt_data( )** function plays the voice prompt of the current menu, gets the caller's response, and returns the response as a DL_MENUOPTS structure. The **gt_data( )** function validates the response and plays any error messages if the response is wrong. The value returned is guaranteed to be a valid response.

The **menu_engine( )** function checks the mo_fcnt field of the returned structure for a value and calls the function if one exists.

menu_engine( ) then transfers control to the next menu indicated in the mo_nxtmenu field. If the value of the field is NULL, control is passed to the default menu, specified in the me_defmenu field. A value of DM_PREV in me_defmenu sets the current menu to be the previous menu, and loops back to the **gt_data( )** call at the beginning of the loop. The value of DM_CURR loops back, leaving the current menu the same.

The **menu_engine( )** function relies on other system routines defined in *d4xtools.c* for playing the prompts, handling errors, and validating responses.

gt_data( ) function

The **gt_data( )** function gets the caller's response to a voice prompt.

```
void      * gt_data(digbufp,validfnc,prompt,retry,error,datap)
char      * digbufp;
void      * (*validfnc)();
char      * prompt;
char      * retry;
void      (*error)();
DL_DATA   * datap;

{
   unsigned short    numretry;               /* Number of allowed retries */
   void              * rp;                   /* general return ptr from valid fn  */

   DV_DIGIT          digit;
   DV_TPT            dattpt[MAXTERMS];

   (void)memcpy(dattp,def_dg_tpt,(MAXTERMS*sizeof(DV_TPT)));

   dattpt[DX_MAXDTMF-1].tp_length=datap->da_recvdig;
   dattpt[DX_MAXTIME-1].tp_length=datap->da_time;
   dattpt[DX_DIGMASK-1].tp_length=bld_tdig_msk(datap->)da_digit;

   numretry = datap->da_numretry;            /* Set the retry count  */

   /*
    * Continue to prompt for this input until an acceptable response is
    * entered or the retry count has been exhausted
    */
   do {

      /*
       * Play the prompt message
       */
      playmsg(prompt);
      /*
       * Get the user's response (DTMF digits)
       */
      if(dx_getdig(devhandle, (DV_TPT *)dattpt,&digit,EV_SYNC) == -1) {
        disperr("Failed on get digits");
      }

      /* Copy the digits to our buffer */
     strcpy(digbufp,digit.dg_value);

     check_term();   /* check for fatal errors)
                     /* If a response was given, validate it */

     /* NOSTRICT */
     if ((rp= (*validfnc)(digbufp)) != NULL) {
        return (rp);
```

```
      }

      /* Not a valid response */
      playmsg(retry);

   } while (--numretry);

   /*
    * No valid response within max. # of retries, so call fatal error
    * handler for this menu, if any provided, and then hang up and go idle.
    */
   if (error)
      (*error)();

   longjmp(idlestate,1);

   /* never gets here but makes lint happy */
   return (NULL);
}
```

The **memcpy( )** function call copies the default get digit DV_TPT array, def_dg_tpt, into the local structure, dattpt. The local DV_TPT array is redefined by using the values of the current menu's DL_DATA structure, datap, in the following code fragment:

```
dattpt[DX_MAXDTMF-1].tp_length=datap->da_recvdig;
dattpt[DX_MAXTIME-1].tp_length=datap->da_time;
dattpt[DX_DIGMASK-1].tp_length=bld_tdig_msk(datap->da_digit);
```

The caller's response to the voice prompt is received by the library call to **dx_getdig( )** or **dx_getdigEx( )**. The reason for termination is checked in the call to the system function **check_term( )**.

The call to the function pointed to by validfnc validates the caller's response. If the response is invalid, the validation function returns a failure, the error message is played, and the caller is prompted to input another response. If the caller does not give a correct response within the maximum number of retries, the error message is played, and the process jumps back to the idlestate. The idlestate is defined in **main( )** by the code:

```
main(argc,argv)
   .
   .
   (void)setjmp(idlestate);     /* where idlestate is defined         */
```

When the longjmp(idlestate,1) call is made in **gt_data( )**, the control jumps to the statement directly after the setjmp(idlestate) call in **main( )**.

check_term( )

When an I/O function returns, the **check_term( )** function is used to find out the reason for termination.

```
void check_term( )
{
  if(ATDX_TERMMSK(devhandle)&(TM_MAXNOSIL|TM_LCOFF|TM_PATTERN|TM_USRSTOP)){
     longjmp(idlestat,1);        /*hang up&ready for new call */
  }
  return;
}
```

check_term( ) uses **ATDX_TERMMSK( )** attribute to check the termination type. If the I/O function was terminated for any of the cases, the function calls a **longjmp( )** and returns to idlestate. If none of these terminations occurred, the caller is still on the line, and the response must be validated.

5.5.5 Messaging System Routine

The messaging system provides a way to play voice prompts or digits by specifying the name. It implements the **dx_play( )** function at a higher level. For example, to play the greeting message, a single argument to the **playmsg( )** function is required:

```
playmsg("intro");
```

Implementation

The message playback is implemented by using two data structures, DL_MSGS and DL_MSGTBL, and by the following functions defined in *d4xtools.c*:

- **playmsg( )**
- **plaympmsg( )**
- **buildmsg( )**
- **buildmsg_v( )**
- **builddate( )**
- **gt_groupiott( )**
- **gt_iott( )**
- **gt_numiott( )**
- **gt_pairiott( )**

The DL_MSGTBL structure provides the physical file name, the descriptor, and the device type for a file of messages. The initialization routine, **init_sys( )**, will open the file and provide a file descriptor for each filename provided in the mt_fn field. The message system requires a NULL-terminated array of DL_MSGTBL structures.

```
typedef struct DL_MSGTBL {
   DL_MSGS   mt_msgsp; /* Pointer to a message structure*/
   char      mt_fn;    /* Pointer to file name */
   int         mt_fd;  /* File descriptor*/
   int         mt_typ  /* Type of the storage media*/
} DL_MSGS;
```

The DL_MSGS structure makes the location of the message transparent to the application by using the structure to assign an ASCII name to the message's offset and length. To signify the end of the messages, the message system requires a NULL-terminated DL_MSGS array for each DL_MSGTBL structure.

```
typedef struct dx_msgs {
   char   * ms_msgname;          /* Pointer to message name*/
   long     ms_ofset;     /* Offset of this message - used in a DX_IOTT*/
   unsigned       ms_lngth;      /* Length of this message - used in a DX_IOTT*/
} DL_MSGS;
```

The message system uses the DL_MSGS and DL_MSGTBL structures to store the information needed for a DX_IOTT structure. The **buildmsg( )**, **buildmsg_v( )**, **builddate( )**, **gt_iott( )**, **gt_numiott( )**, **gt_groupiott( )**, and **gt_pairiott( )** functions use the message structures to build the DX_IOTT structure. The **playmsg( )** and **plaympmsg( )** functions use the DX_IOTT structure to play the message. The code for the **playmsg( )** function is:

```
void playmsg(msgp)
char * msgp;
{
     DX_IOTT iott[25];
     DX_IOTT * iottp;

     if (msgp[0] < ' '){            /* if it's an IOTT list         */
        iottp = (DX_IOTT *)msgp;    /* just play it                 */
     }else{                         /* else                         */
        buildmsg(iott,msgp,NULL);   /* build IOTT list for          */
        iottp = iott;               /* named prompt of nmbr         */
     }

     if(dx_play(devhandle,(DX_IOTT *)iottp,(DV_TPT *)def_rp_tpt,NULL) == -1) {
     disperr("Error playing message");
}
     check_term(&csb);     /* abort if hang-up condition */
}
```

The first if statement checks to see if the msgp parameter is either a DX_IOTT structure or an ASCII name. If it is an ASCII name, a call to **buildmsg( )** is used to create a DX_IOTT structure from the DL_MSG structure named by msgp.

The **dx_play( )** function plays the message using the DX_IOTT structure and the default record-play DV_TPT structure located in def_rp_tcb.

Index

S

T