EK-DTC01-RM-003

DECtalk DTC01

Programmer Reference Manual

Prepared by Educational Services
of
Digital Equipment Corporation

1st Edition, December 1983
2nd Edition, June 1984
3rd Edition, March 1985

The information in this document is subject to change without notice. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. Compliance with the FCC Class B technical requirements is dependent upon the use of interconnecting cables specified in the User/Installation manual. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following methods.

- Reorient the receiving antenna.
- Relocate the computer with respect to the receiver.
- Move the computer away from the receiver.
- Plug the computer into a different outlet, so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the booklet *How to Identify and Resolve Radio/TV Interference Problems*, prepared by the Federal Communications Commission, helpful. This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-00345-4.

UNIX and System V are trademarks of AT&T Bell Laboratories.

Touch-Tone and AT&T are trademarks of American Telephone and Telegraph Company.

Zeus is a trademark of Zilog, Inc.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts.

DIGITAL™	DECwriter	RSTS
DEC	DIBOL	RSX
DECmate	LA	ULTRIX-32
DECnet	MASSBUS	UNIBUS
DECsystem-10	PDP	VAX
DECSYSTEM-20	P/OS	VMS
DECtalk	Professional	VT
DECUS	Rainbow	Work Processor

TELEPHONE COMPANY AND FCC REQUIREMENTS AND RESPONSIBILITIES

FCC regulations require that you provide your local telephone company business office with the following information before you connect DECtalk to the telephone network.

- The particular lines(s) to which terminal equipment will be connected (by telephone number)
- The make, model number, and FCC registration number (label on bottom of unit)
- The ringer equivalence for the registered terminal equipment (label on bottom of unit)
- The type of jack needed (if not already installed)

Make: DECtalk
Model: DTC01-AA
FCC Registration: AO994Q-12463-0T-E
Ringer equivalence: 0.3B
Type of jack: USOC RJ11C or USOC RJXA1 for telephone line interference (See the *DECtalk DTC01 Installation Manual.*)

You must also notify the telephone company when you permanently disconnect terminal equipment from telephone line(s).

You may not connect terminal equipment to a party line or coin-operated telephone equipment.

If the telephone or telephone line is already equipped with a jack you should be able to plug in DECtalk without any additional telephone company charge. Otherwise, the telephone company will install a jack, which usually results in a one-time installation charge.

If terminal equipment damages the telephone network, the telephone company can, after notifying the customer, temporarily discontinue service. However, when prior notice is not practical, the telephone company can temporarily discontinue service immediately. In such cases, the telephone company shall:

- Promptly notify customers that service has been discontinued
- Give customers the opportunity to correct the situation
- Inform customers of their right to bring a complaint to the FCC according to Subpart E of Part 68 of FCC Telephone Equipment Rules.

The DECtalk DTC01 unit is classified as terminal equipment.

CANADIAN APPLICATION NOTICE

The Canadian Department of Communications label identifies certified equipment. This certification means that the equipment meets certain telecommunications network protective, operational, and safety requirements. The department does not guarantee the equipment will operate to the user's satisfaction.

Before you install this equipment, make sure it is permissible to be connected to the local telecommunications company's facilities. You must also install the equipment by using an approved connection method. In some cases, the company's inside wiring associated with single line individual service can be extended by a certified jack/plug/cord ensemble (telephone extension cord). Be aware that complying with the above conditions may not prevent degradation of service in some situations. Telecommunications company requirements do not allow you to connect their equipment to customer-provided jacks, except where specified by individual telecommunications company tariffs.

Only authorized Canadian maintenance facilities, designated by the supplier, should repair certified equipment. If you repair or alter certified equipment yourself, or if the equipment malfunctions, the telecommunications company has cause to ask you to disconnect the equipment.

You should ensure (for your own protection) that the electrical ground connections for the power utility, telephone lines, and internal metallic water pipe system, if present, are connected together. This precaution may be particularly important in rural areas.

CAUTION: Do not try to make such connections yourself, but contact the appropriate electric inspection authority or electrician.

CONTENTS

CHAPTER 2 SETUP ESCAPE SEQUENCES

CHAPTER 3 VOICE COMMANDS, PHONEMIC TEXT, AND THE USER DICTIONARY

CHAPTER 4 TELEPHONE COMMUNICATIONS

CHAPTER 5 MAINTENANCE AND DEBUGGING COMMANDS

CHAPTER 6 C PROGRAM EXAMPLE

FIGURES

TABLES

INTRODUCTION

This manual describes how to use DECtalk with a host computer. The text explains the escape sequences you can use with DECtalk.

Terminals display information from a computer on a screen or paper; they provide communication with computers through the sense of sight. DECtalk speaks information from a computer in an English-language voice; it provides communication with computers through the sense of hearing.

The *DECtalk DTC01 Owner's Manual* (EK-DTC01-OM) describes how to use DECtalk connected to a terminal. This manual describes how to use DECtalk connected to a host computer.

Chapter 1 describes how DECtalk can communicate with a host computer through computer application programs. The chapter also describes some guidelines for writing applications.

Chapter 2 describes the setup escape sequences that initialize and control the DECtalk environment.

Chapter 3 describes how to use voice commands, send phonemic text to DECtalk, and load the user dictionary.

Chapter 4 describes how DECtalk works when connected to a telephone network.

Chapter 5 describes the maintenance and debugging commands used to test DECtalk.

Chapter 6 provides a detailed application program written in C programming language. You can copy this program.

Chapter 7 provides a sample program written in BASIC-PLUS programming language. You can copy this program.

The appendices summarize the DECtalk escape sequences, the phonemic alphabet used, and the other available DECtalk documentation.

1 HOW DECtalk PROGRAMMING WORKS

This chapter gives you an overview of DECtalk programming. The chapter has four major sections.

- "Communicating with DECtalk" describes the types of data DECtalk can receive and the operating modes DECtalk uses.
- "Escape Sequences and Control Characters" explains the basic format for entering commands with escape sequences. This section includes a table of the control characters that DECtalk recognizes.
- "DECtalk–Computer Communication" lists some special rules DECtalk follows when processing text. This section describes how escape sequences affect the flow of information between DECtalk, a local terminal, and a computer. The section also describes data synchronization.
- "Developing Your Application" provides some guidelines for writing application dialog and encoding your program.

COMMUNICATING WITH DECtalk

DECtalk is an intelligent peripheral device, so the following guidelines apply.

- You cannot program DECtalk directly. After the initial power-up operations, DECtalk is controlled through a terminal or host computer.
- DECtalk is easy to control, because the internal DECtalk processor is sophisticated enough to process complex operations with simple commands.
- You can select DECtalk's operating characteristics and have DECtalk answer questions (from the host computer) about its status. DECtalk can also inform the host computer of status changes. For example, DECtalk can tell the host computer if a connected telephone has rung.
- DECtalk memory can store some information. For example, DECtalk has an extensive built-in pronunciation dictionary. You can load a user-defined dictionary under computer control.

The following paragraphs describe how DECtalk sends and receives information from the host computer.

Types Of Data

DECtalk can receive two types of data through its communications connector, text and commands.

Text is data that DECtalk will speak. Text consists of English-language sentences, phonemically spelled text, or a combination of both.

Commands are instructions to perform an action. Commands are not spoken by DECtalk.

There are two ways to send commands – with escape sequences or with square brackets []. Some commands you can only send with escape sequences, and some commands you can send both ways.

Escape sequences start with an ESC character, followed by a string of ASCII characters. DECtalk interprets the string as a special command. This manual describes the escape sequence method of sending commands.

Square bracket [] commands let you include speech commands and phonemic text with text information, if MODE SQUARE is on.

The "Mode Selection" section in Chapter 2 describes MODE SQUARE. The *DECtalk DTC01 Owner's Manual* describes square bracket commands and syntax.

Appendix B summarizes both command methods. Figure 1-1 shows the relationship between terminal commands and host commands. This chapter provides more information on escape sequence conventions and format.

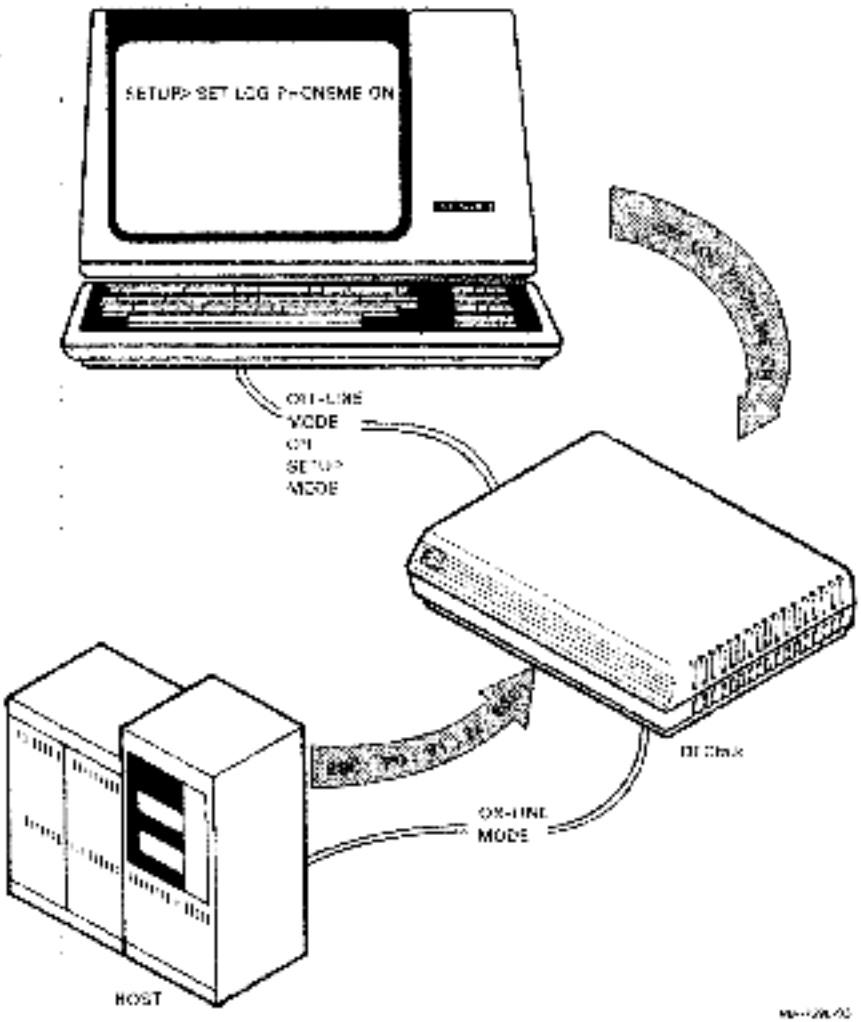

Figure 1-1 DECtalk, Terminal, and Host Communications

Operating Modes

DECtalk has three operating modes: setup, off-line, and on-line.

You use *setup* mode to select the operating parameters of DECtalk, such as communication line characteristics and phonemic representation.

You use *off-line* mode when DECtalk is connected to a terminal. The *DECtalk DTC01 Owner's Manual* describes off-line mode.

You use *on-line* mode when DECtalk is connected to a host computer. Commands must be sent as escape sequences from the host computer.

DECtalk powers up in on-line mode. When you connect DECtalk to a terminal for local use, you must switch DECtalk to off-line mode; press the **BREAK** key to enter setup mode, then select the off-line setting.

USING ESCAPE SEQUENCES AND CONTROL CHARACTERS

This section describes the general syntax of escape sequences, and how to use them with DECtalk.

Escape Sequences

In setup mode, you can enter commands directly to DECtalk through the terminal.

When DECtalk is on-line, you can enter most setup commands plus other on-line commands; however the commands come from the computer instead of the terminal. For example, you can only load the user-defined dictionary while on-line.

On-line and setup commands act the same, but they have different formats. For example, the user command

```
SETUP>SET LOG PHONEME ON
```

is sent from a computer as an escape sequence

ESC P 0 ; 8 1 ; 2 z ESC \

You can omit parameters with a value of 0 (ASCII). For example, you could send the above sequence as ESC P ; 8 1 ; 2 z ESC \. DECtalk does not send parameters with a value of 0 in its reply sequences.

NOTE: Escape sequences in this manual are spaced for clarity only. Spaces are not part of the actual escape sequences.

DECtalk escape sequences have the following characteristics.

1. They begin with an ESC character.

2. The ESC character is followed by ASCII characters that define the command.

3. Every character in the command is important. You must enter the exact characters shown. For example, in the command above, the semicolons are part of the command. The letter z is lowercase; an uppercase Z has no meaning to DECtalk.

4. Programming standards require that you end some commands with a sequence terminator - ESC \. This manual includes ESC \ with all commands that require it.

5. Escape sequences only work on the DECtalk host line. This is different from most terminals, which can interpret typed escape sequences.

The ESC character plus the ASCII characters are like a compressed version of the off-line commands. Figure 1-2 shows the meaning of each part of a typical escape sequence.

DECtalk ignores invalid sequences and commands.

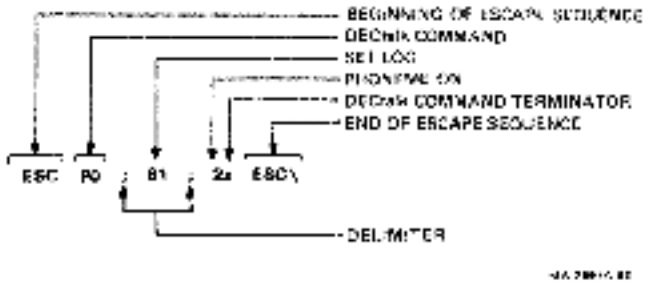

Figure 1-2 Typical Escape Sequence Format

Escape Sequence Format

The following chapters describe the specific escape sequences used with DECtalk. This manual includes the following information with all escape sequences (Figure 1-3).

Mnemonic
ASCII characters
Parameters
Decimal value

NOTE: Since DECtalk suppresses parameters with a value of 0, DECtalk would send ESC P ; 31 ; P3 Z ESC \ *for the sequence in Figure 1-3 (assuming* P3 *is not 0). If* P3 *is 0, DECtalk would send* ESC P ; 31 ; z ESC \.

The *mnemonic* is a unique name (such as DT_INDEX) used to identify the escape sequence. Mnemonics do not have any direct programming significance; that is, DECtalk does not recognize a mnemonic name as a valid escape sequence. However, when you refer to escape sequences by mnemonic in program documentation and program variables, it simplifies editing and debugging.

The program examples in Chapter 6 use mnemonics for the appropriate escape sequences. The header file DECTLK.H in Chapter 6 defines all DECtalk command mnemonics.

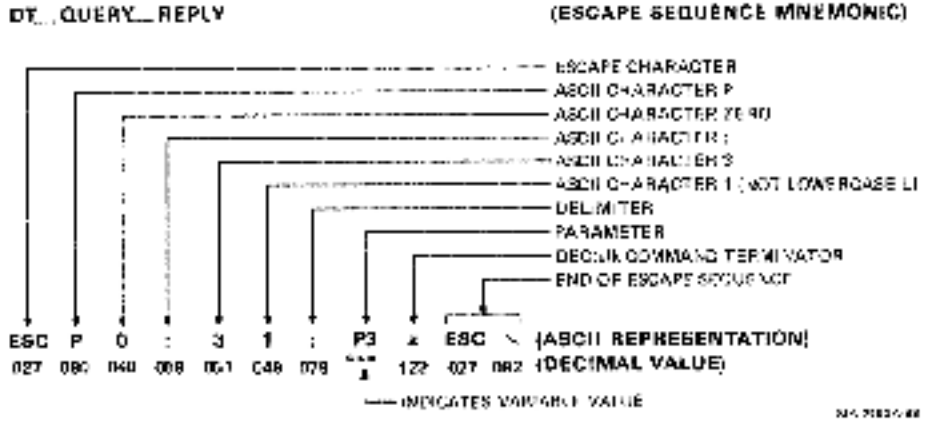

Figure 1-3 Escape Sequence Representations

The *ASCII characters* are the actual characters to use. The "Escape Sequences" section in this chapter gives an example of an escape sequence in ASCII format. The escape character is represented by ESC in all sequences. The numbers that appear are actual ASCII characters, not numeric values.

Parameters appear in escape sequences that can cause several DECtalk actions. These different actions depend on parameter values.

Parameters are represented in this manual by a capital P followed by a number or letter. Parameters are always sent to DECtalk as a decimal number, in ASCII format.

An empty parameter is treated like a parameter with a value of 0. The sequences ESC P ; z and ESC P 0 ; 0 z are identical. *DECtalk always sends a 0 parameter as an empty string*. However, the 0 parameters are always shown as explicit zeros in examples.

This manual lists possible parameter values in tables. There are two methods used to show parameter values.

1. Usually the ASCII character(s) appears (with the decimal value underneath as a check). Use the ASCII character(s) in the escape sequence.

2. Sometimes only a numeric value appears. You must convert the numeric value to a sequence of ASCII characters for the escape sequence.

The *decimal value* of each escape sequence character appears directly under the character, so you can verify the sequence characters. Parameters are marked with asterisks ("*"), indicating that the value is variable.

Chapter 2 provides complete tables of all ASCII characters and their decimal, octal, and hexidecimal equivalents.

Figure 1-3 shows all the parts of an escape sequence.

Control Characters

Some control characters (such as carriage return and backspace) have special meanings. Table 1-1 lists the control characters that DECtalk recognizes. DECtalk ignores any other control characters.

Table 1-1 Control Characters and Host Communications

Character	Mnemonic	Decimal Value	Function
Back space	BS	008	See "Effect of Backspace (BS) Character" (Chapter 1).
Horizontal tab	HT	009	Same as a space.
Line feed	LF	010	Same as a space.
Vertical tab	VT	011	Clause terminator.
Form feed	FF	012	Same as a space.
Carriage return	CR	013	Same as a space.
Shift out	SO	014	Used in character set selection. See "Selecting Alternate Character Sets" (Chapter 2).
Shift in	SI	015	Used in character set selection. See "Selecting Alternate Character Sets" (Chapter 2).
Substitute	SUB	026	If a communication error occurs, DECtalk replaces the bad character with a SUB character. The SUB character acts as a clause terminator. See "DECtalk Communication" (Chapter 1) for information on clause terminators.
Escape character	ESC	027	Introduces escape sequences.
Space	SP	032	Normal word terminator.

Control Character Logging
Version 2.0 of DECtalk firmware improves control character logging. Before version 2.0, some control characters were not correctly logged. In particular, the CTRL-K (clause flush) control character sequence (generated internally by DECtalk), was not logged unless DECtalk received text from the host in 5 seconds. If a heavily loaded system was slow to respond, DECtalk might not log the current event.

For example, suppose the host system stopped sending data in the middle of a phonemic text string or an escape sequence. DECtalk would execute a time-out, exit phonemic text mode (or ignore the escape sequence), and fail to log the event. The result was problems for the application developer in tracking control character logs.

NOTE: You should enable LOG_INHOST *or* LOG_RAWHOST *to ensure the proper logging of all characters. See "Local Log Control (DT_LOG)" in Chapter 5 for more information on control character logging.*

Effect of the Backspace (BS) Character
If DECtalk finds the backspace character in a word, DECtalk modifies the word according to the hierarchy of the characters involved, as follows.

1. letters and digits
2. punctuation
3. underline character

The BS character allows DECtalk to process text containing overstrikes and underlining.

Here are several examples of DECtalk's processing (spaced for clarity).

Input	Pronounced as
a BS _	a
_ BS a	a
a BS b	b
ab BS BS de	de
a BS "	a
a BS " BS _	a

DECtalk-COMPUTER COMMUNICATION

Programming DECtalk is similar to programming a smart terminal (such as a VT220). That is, DECtalk and the host computer must exchange information according to fixed rules.

DECtalk does not process text until reaching a valid clause boundary. Clause boundaries mark the end of phrases or sentences. DECtalk recognizes the following clause boundaries.

- A period, comma, exclamation point, or question mark is a valid boundary. If a period is used, DECtalk checks the characters after the period (because periods do not always mean the end of a sentence).

- A full buffer also acts as a boundary. If DECtalk's temporary buffer begins to approach its fill limit (at about 12 words), DECtalk begins speaking what is in the buffer and treats the last word as a clause boundary.

- A timeout is another boundary. If nothing is sent to DECtalk within 5 seconds and there is text in the buffer, then DECtalk speaks all text in the buffer as though a comma had been sent with the text.

Escape sequences represent (1) commands sent from the host to DECtalk, and (2) status replies sent from DECtalk to the host. All escape sequences begin with the ESC character; a sequence ends when the last character required for that sequence is sent. Do not use a carriage return or any other normal terminating character to terminate an escape sequence.

Figure 1-4 shows the data paths in DECtalk, as follows.

1. The DECtalk unit is in the center of the figure. The speech processor is part of the DECtalk unit, but is shown as a separate module.

2. Arrows show the direction of information flow. Notice that information flows from the terminal (or telephone) to the host, and from the host to the terminal (or telephone). However, information only goes to the speech processor from the host or terminal. Information from the speech processor is sent to the telephone or speaker.

3. Each DECtalk escape sequence affects the flow of information within particular data paths. The switches within the data paths represent the points at which the escape sequences act. For example, the DT_STOP escape sequence affects the data flow from the host to the speech processor.

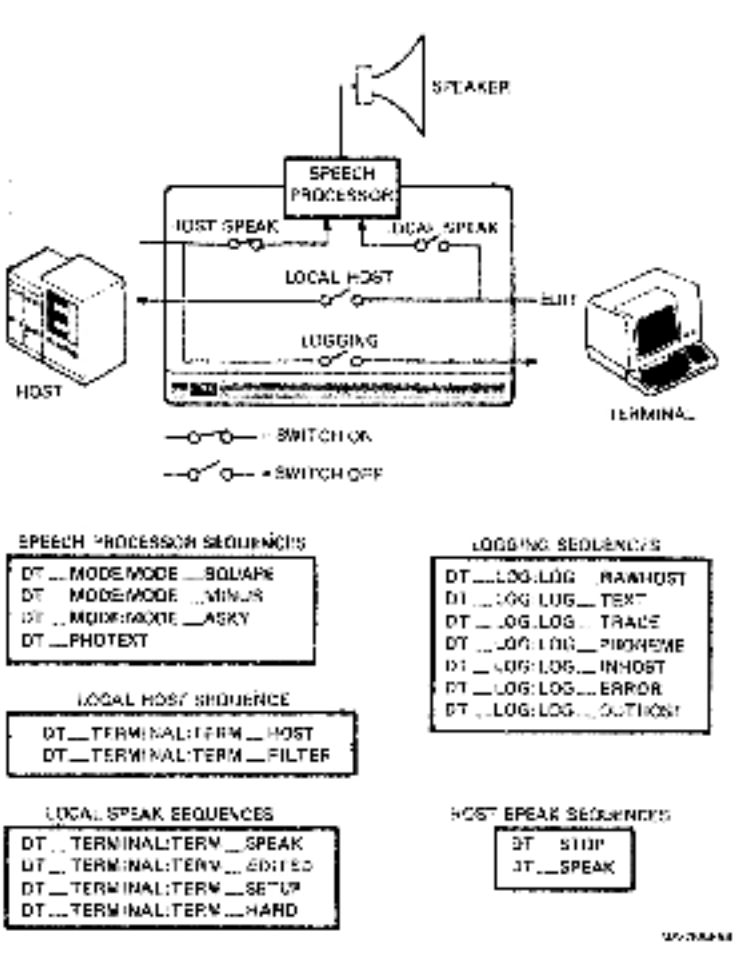

Figure 1-4 DECtalk-Computer Program Interaction

4. Since there are a large number of commands and parameters, they are grouped in boxes under the diagram.

5. Some commands have parameters (sometimes called arguments). Figure 1-4 shows commands and their parameters. The parameters (if any) appear after a colon (:) mark.

There are many ways to control and use DECtalk when connected to a host computer. The rest of this chapter and the sample program in Chapter 6 describe a general programming method for DECtalk. If you are writing a control program for DECtalk, remember that your application and needs may not match the descriptions that follow exactly.

DECtalk Setups

The controlling program first configures DECtalk to ensure that all parameters are set correctly. Use these steps in your program.

1. Programs may wish to send a "What are you?" sequence to make sure DECtalk is available. DECtalk replies with a code correctly identifying DECtalk.

   The "Device Attribute Request" section in Chapter 5 describes "What are you?" sequences.

2. Send setup commands to configure DECtalk for host–DECtalk communication. The required setup commands vary from computer to computer and from application to application; however, here are some commands to consider.

   a. Include any required communication command (such as 7-bit or 8-bit codes, and code interpretation. See "Selecting ASCII Character Sets" in Chapter 2.

   b. Set MODE SQUARE on (if desired). This command ensures that phonemic code values are accepted. See "Mode Selection" in Chapter 2.

   c. If you connect DECtalk to the public telephone network, select the correct telephone handling parameters. See Chapter 4 for these parameters.

3. You may have to set up the host computer (or DECtalk communication line) for DECtalk commands. You must set up the computer for single-character, unsolicited input, and operating system XON/XOFF processing.

   Setting up computers is beyond the scope of this manual; however, Chapter 6 has examples of setting up certain Digital Equipment Corporation computers for DECtalk.

   If your host computer cannot support single-character processing, you can use the DT_MASK escape sequence to permit line-at-a-time processing. See "Keypad Mask Command (DT_MASK)" in Chapter 5.

4. Other commands depend on the DECtalk environment, such as debugging commands or special text-to-speech commands.

Program Control

DECtalk is primarily a speech device; its internal code is directed towards producing artificial speech. DECtalk assumes the host computer will handle most of the necessary control operations (such as waiting for task completion and requests for status).

The host is responsible for control and coordination, but this is not a major task. The rest of this chapter describes areas you should consider when designing the DECtalk program application.

Data Synchronization

DECtalk's speech rate is much lower than the (potential) data transfer rate on the host communication line. DECtalk sends an XOFF character (CTRL-S) to the host when its input buffer is almost full, to signal that any more input will be discarded. When DECtalk's input buffer is almost empty, it sends an XON character (CTRL-Q); XON tells the host to start sending data again.

The DECtalk input buffer is large enough so that the host can continue sending data at the highest speed (9600 baud) for up to 250 milliseconds after it receives the XOFF, without losing data.

Figure 1-5 shows how DECtalk synchronizes data transfer with the host through XON/XOFF signals.

If the host does not stop sending data in time, the input buffer may overflow and characters may be lost. DECtalk does not give an audible warning of this overflow, except for the obvious garbling of partial words. The host can issue a device status request (DSR) command to determine if an input buffer overflow occurred.

Most operating systems have a HOSTSYNC option (or its equivalent) in the terminal setup characteristics. If this characteristic is set on the DECtalk communications line, the host computer handles XON and XOFF signals. If XON/XOFF coordination is not available, the application program may be able to avoid buffer overflow by using the DT_SYNC command and controlling the program's output rate; however, Digital does not recommend this method because it causes errors. The "Data Synchronization" section in Chapter 3 discusses DT_SYNC.

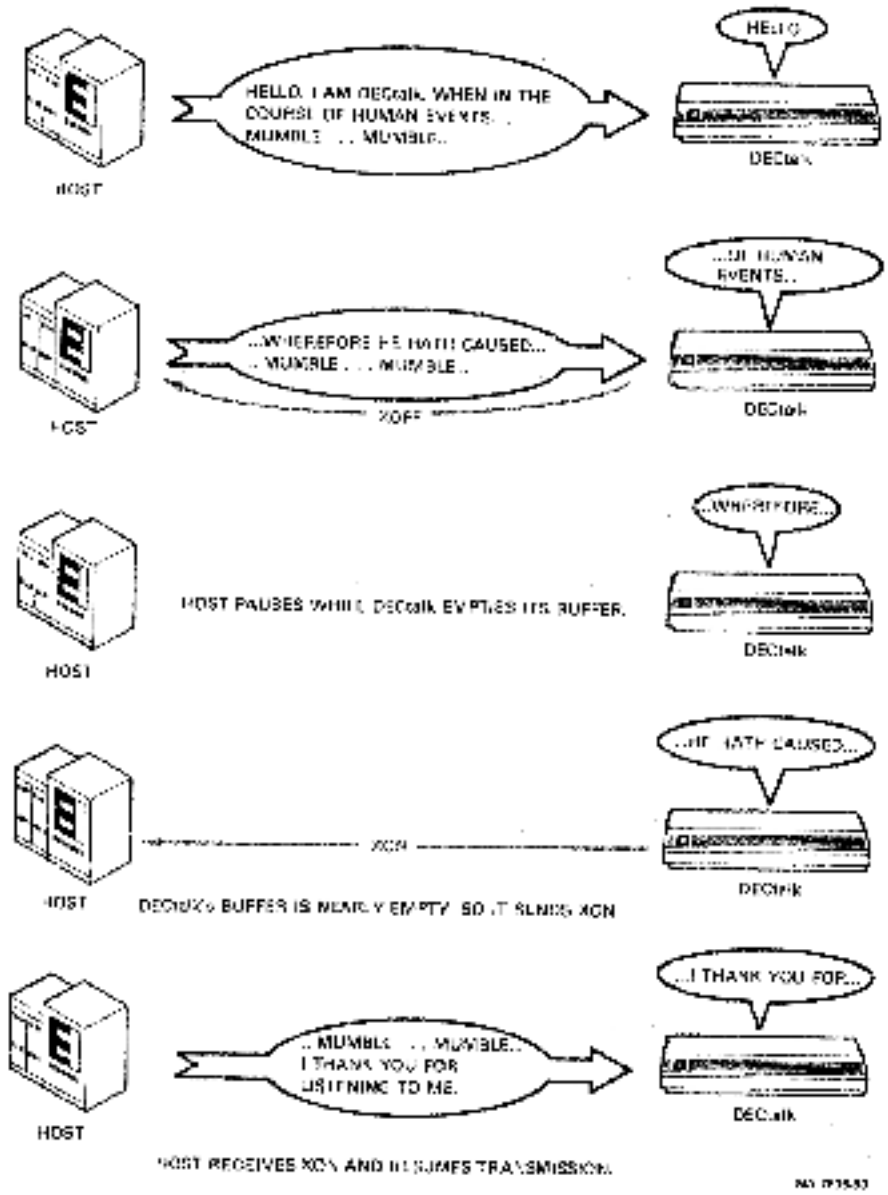

Figure 1-5 Synchronizing DECtalk and Host Communications

DECtalk–Host Program Sequence

After you set up a parsing method so data can pass between the host computer and DECtalk, you should set up the host for the kinds of data to receive. How you set up information handling depends on the needs of the user. The following section provides some guidelines for developing your application's dialog.

DEVELOPING YOUR APPLICATION

DECtalk lets people use your computer-based applications from any keypad telephone. DECtalk speaks your messages in an understandable voice. When the user presses keypad keys, DECtalk sends those characters to your program. The following guidelines should help you adapt your application to your unique needs.

General Guidelines

- Keep the user's point of view, not the programmer's. Use commands that are logically related to the way users see the task.
- Most people will not carry a large user guide around with them.
- Frequent users become experts quickly.

Writing Dialog

- Keep dialog simple, but meaningful.
- Organize each message as follows.

  1. Put the hardest element to remember first.
  2. Put the easiest elements to remember in the middle.
  3. Put information for immediate recall at the end.

- Tell users only what they need to know in order to continue a task.
- Do not use humor or threats. Keep dialog strictly factual and informative.

Help Messages and Replies

- Make help messages optional. Let users decide when they want more information.
- Repeat significant phrases in help messages.
- Let users know that DECtalk is acting on their specific commands. For example, say "Sending reply to Ms. Jones," rather than "Sending reply."

Entering Keypad Commands

- Remember, there are only 12 keys on the telephone keypad.
- Keep the same function on the same key.
- Refer to keypad numbers, not letters. People do not remember which letter is on which key. Use "Press 1 for next, 2 for previous, 3 to exit," rather than "Press N for next, P for previous, E for exit."
- Create a standard method for users to exit from a subtask to the main dialog.

Names, Part Numbers, and Alphanumeric Text

In many DECtalk applications, you use the 12 keypad keys to enter a person's name or an alphanumeric part number. Since the application program only receives a string of digits (and the # and * characters), the program must use the digits as an index to the actual data item.

If you are designing a new system, you could specify numeric part numbers only. However, in the real world, a company is not going to change its existing warehouse methods to match DECtalk. So the user will have to enter something that your application can translate into the current system.

Direct Numeric Encoding

Using this method, the user simply presses the key labeled with the desired letter. For example, to select "DIGITAL" the user would press **3444825.** You could assign the letters Q and Z to the 7 (PQRS) and 9 (WXYZ) keys, respectively.

Numeric encoding is a simple method to describe and implement. Since users can recall more than one item for a given digit string, your application must provide a way to select alternatives. You could have users select alternatives by number. Or you could have them step through a list, using next and previous commands.

Numeric encoding is probably the best method for lists of names and for many part number applications. You can even use this method for ID or password entry.

Two-Character Encoding

Some applications use specific letters in their codes (for example, three-character airport codes). You cannot use direct numeric encoding to select specific letters on the keypad.

One possible solution is two-character encoding. This method matches the three letters on each key to the three columns of keys on the keypad. The user presses two keys to select a letter.

1. The key with the desired letter
2. The **1**, **2** or **3** key (to select the specific letter)

For example, to select "DEC" the user would press **313223**. You could have users enter numbers together with **the 0 (OPER)** key. And you could assign the missing **Q** and **Z** (plus the space character) to the **1** key.

The United States Federal Aviation Administration used the above method to provide a voice response weather system. (The USFAA used stored segments of speech – DECtalk was not available at the time.)

Ending Commands and Data

You can use single-character commands and fixed-length data fields for many applications. But for complex applications or variable-length data you may find it simpler to ask the user to end all commands and data by pressing a special key (such as #). Pressing # lets the program know that the right number of characters have been entered.

You could also use DECtalk's flexible keypad timeout facility. If the user is entering a variable-length numeric field, use a long timeout for the first digit and (possibly) a shorter timeout for successive digits.

Application Development Tips

Here are some tips for encoding the application itself.

- Use timeouts for everything. Assume that the user may hang up the phone at any time. Also assume that data entry will be quite slow. This is important when planning data base entry and record-locking strategies.

- The DECtalk applications support library may return an error code due to transient problems (such as a system overload). The simplest recovery is to hang up the call and reinitialize DECtalk. Log the problem for future action.

- People can recall about 5 seconds of text without difficulty. You can use entries such as "1 for yes, 2 for no, 3 for maybe," but do not ask an untrained user to remember anything more complex.

- DECtalk tends to spell out text that may be ambiguous (for example, part numbers). You can write a small filter subroutine that recognizes certain strings and pronounces them in a form more suitable for your specific application.

- If your application accepts data from the telephone keypad, make sure the operating system can buffer type-ahead characters. Also, make sure the operating system responds to DECtalk's XOFFs.

- DECtalk speaks pending text if the host system stops delivering text for 5 seconds. This feature may be a problem on an overloaded system. You may need help from the system manager to obtain more resources or adjust program priorities.

- When you have DECtalk speak information from a data base, remember that the listener hears the information only once. You should offer a *repeat* function for complex subject matter. If you have DECtalk read mail or other unstructured text, you should offer a *back up one sentence* function, using the Index Test command (Chapter 3) to signal what has been heard.

SETUP ESCAPE SEQUENCES 2

DECtalk has several features you can change to control the operating environment. These parameters include the following.

- line characteristics (such as line speed)
- character sets (to send and receive information)
- modes (to control DECtalk's interpretation of special characters and phonemic text)

There are also several testing and inquiry commands, described in Chapter 5.

SELECTING ASCII CHARACTER SETS

DECtalk is a computer terminal device, and conforms to the standards for computer terminals.

DECtalk speech does not include much of the visual information of character sets. For example, DECtalk uses the following rules for all character sets.

1. Uppercase and lowercase letters are considered the same. For example, DECtalk speaks the letter G as "gee," not "uppercase gee."

2. Foreign letters (as found in the multinational character set) are spoken as English. For example, DECtalk speaks the letter ä as "a," not "a umlaut."

3. You can translate or map 7-bit codes into 8-bit codes, and 8-bit codes into 7-bit codes (Figure 2-1). This mapping has no effect on spoken text. Table 2-1 gives the escape sequences that change DECtalk to 7-bit or 8-bit modes.

The following paragraphs describe how DECtalk interprets certain keyboard (or host computer) generated codes.

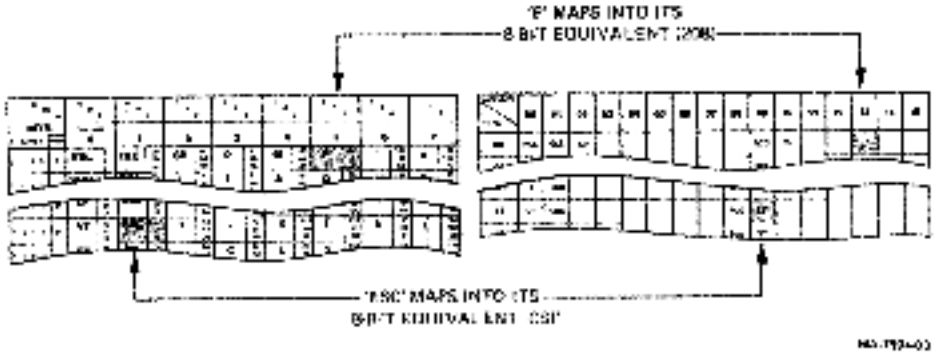

Figure 2-1 Mapping 7-Bit and 8-Bit Tables

Table 2-1 Selecting 7-Bit or 8-Bit Mode

Mnemonic	Escape Sequence Decimal Value			Function
Transmit				
S7C1T	ESC 027	SP 032	F 070	Select 7-bit C1 character transmission.
S8C1T	ESC 027	SP 032	G 071	Select 8-bit C1 control character transmission.
Receive				
DECTC1	ESC 027	SP 032	G 064	Select 7-bit character reception. The high-order (eighth) bit is ignored in all received C1 control characters.
DECAC1	ESC 027	SP 032	? 063	Select 8-bit C1 control character reception. The high-order (eighth) bit is accepted in all received C1 control characters.

CODING STANDARDS

The DTC01 uses an 8-bit character encoding scheme and a 7-bit code extension technique that are compatible with the following ANSI and ISO standards. ANSI (American National Standards Institute) and ISO (International Organization for Standardization) specify the current standards for character encoding used in the communications industry.

Standard	Description
ANSI X3.4 – 1977	American Code for Information Interchange (ASCII)
ISO 646 – 1977	7-Bit Coded Character Set for Information Processing Interchange
ANSI X3.41 – 1974	Code Extension Techniques for Use with the 7-Bit Coded Character Set of American National Code Information Interchange
ISO Draft International Standard 2022.2	7-Bit and 8-Bit Coded Character Sets – Code Extension Techniques
ANSI X3.32 – 1973	Graphic Representation of the Control Characters of American National Code for Information Interchange
ANSI X3.64 – 1979	Additional Controls for Use with American National Standard for Information Interchange
ISO Draft International Standard 6429.2	Additional Control Functions for Character Imaging Devices

CODE TABLE

A code table is a convenient way to represent 7-bit and 8-bit characters, because you can see groupings of characters and their relative codes clearly

7-Bit ASCII Code Table

Figure 2-2 is the 7-bit ASCII code table. There are 128 positions corresponding to 128 character codes arranged in a matrix of 8 columns and 16 rows.

COLUMN	0	1	2	3	4	5	6	7
BITS								
ROW								
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
10	LF	SUB	*	:	J	Z	j	z
11	VT	ESC	+	;	K	[	k	{
12	FF	FS	,	<	L	\	l	|
13	CR	GS	-	=	M	]	m	}
14	SO	RS	.	>	N	^	n	~
15	SI	US	/	?	O	_	o	DEL

KEY

ESC

OCTAL
(DECIMAL)
HEX

MA-0203A-87

Figure 2-2 7-Bit ASCII Code Table

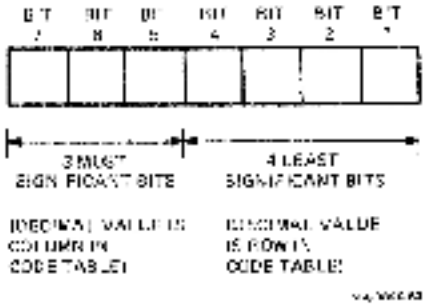

Figure 2-3 7-Bit Code

Each row represents a possible value of the four least significant bits of a 7-bit code (Figure 2-3). Each column represents a possible value of the three most significant bits.

Figure 2-2 shows the octal, decimal, and hexadecimal code for each ASCII character. You can also represent any character by its position in the table. For example, the character H (column 4, row 8) can be represented as 4/8.

DECtalk processes received characters based on two character types defined by ANSI, graphic characters and control characters.

Graphic characters are characters you can display on a video screen. The ASCII graphic characters are in positions 2/1 through 7/14 of Figure 2-2. They include alphanumeric characters plus punctuation marks and various text symbols. Examples are C, n, ", !, –, $.

Control characters are not displayed. They are single-byte codes that perform specific functions in data communications and text processing. The ASCII control characters are in positions 0/0 through 1/15 (columns 0 and 1) of Figure 2-2. The SP character (space, 2/0) can be considered either a graphic character or a control character depending on the context. DEL (7/15) is always used as a control character.

Control character codes and functions are standardized by ANSI. Examples of ASCII control characters with their ANSI-standard mnemonics are CR (carriage return), FF (form feed), and CAN (cancel).

8-Bit Code Table

The above conventions can be generalized to the 8-bit character encoding used on DECtalk. Figure 2-4 shows the 8-bit code table. It has twice as many columns as the 7-bit table, because it contains 256 versus 128 code values.

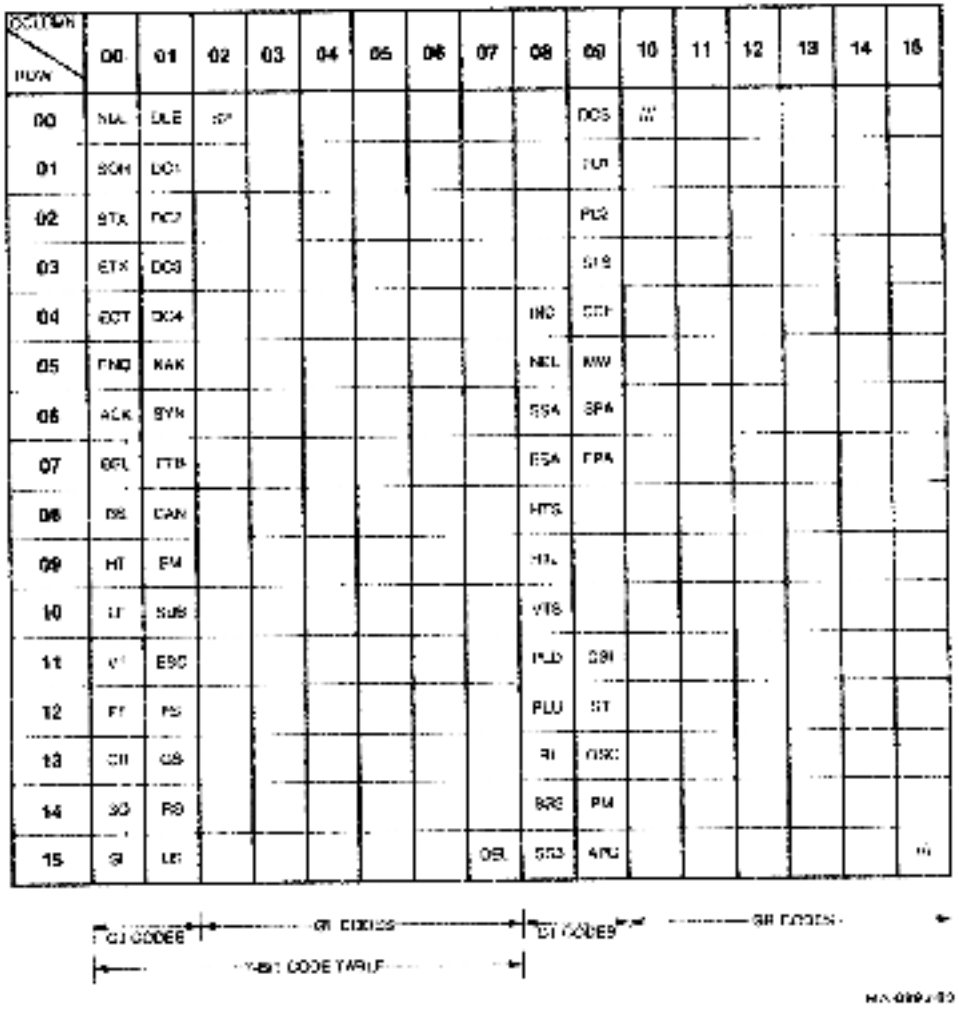

Figure 2-4 8-Bit ASCII Code Table

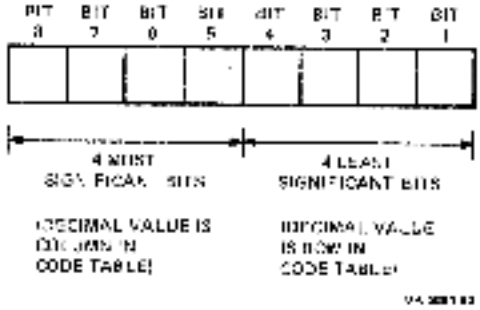

Figure 2-5 8-Bit Code

As with the 7-bit table, each row represents a possible value of the four least significant bits of an 8-bit code (Figure 2-5). Each column represents a possible value of the four most significant bits.

All codes on the left half of the 8-bit table (columns 0 through 7) are 7-bit compatible: their eighth bit is not set and can be ignored or assumed to be 0. You can use these codes in either a 7-bit or an 8-bit environment. All codes on the right half of the table (columns 8 through 15) have their eighth bit set. You can use these codes only in an 8-bit compatible environment.

The 8-bit code table (Figure 2-4) has two sets of control characters, C0 (control zero) and C1 (control one). The table also has two sets of graphic characters, GL (graphic left) and GR (graphic right).

On DECtalk, the basic functions of the C0 and C1 codes are as defined by ANSI. C0 codes represent the ASCII control characters described earlier. The C0 codes are 7-bit compatible. The C1 codes represent 8-bit control characters that let you perform more functions than those possible with the C0 codes. C1 codes can be used directly only in an 8-bit environment. Some C1 code positions are left blank because their functions are not yet standardized.

NOTE: DECtalk only recognizes the SS2, SS3, DCS, CSI, and ST control codes. The others are ignored.

The GL and GR sets of codes are reserved for graphic characters. There are 94 GL codes in positions 2/1 through 7/14 and 94 GR codes in positions 10/1 through 15/14. By ANSI standards, positions 10/0 and 15/15 are not used. You can use GL codes in 7-bit or 8-bit environments. You can use GR codes only in an 8-bit environment.

CHARACTER SETS

You cannot change the functions of the C0 or C1 codes. However, you can map different sets of graphic characters into the GL and/or GR codes. The sets are stored in the terminal. But they are not available for use until mapped into the GL or GR codes.

Selecting Alternate Character Sets (G0 – G3)

DECtalk has four alternate character set areas: G0, G1, G2, and G3. When DECtalk powers up, it loads the ASCII_G (7-bit) character set in alternate buffers G0 and G1. The DEC multinational (8-bit) character set is loaded in alternate buffers G2 and G3.

DECtalk does not call the alternate character sets directly from G0, G1, G2, or G3. The selected set is first mapped into the GL or GR areas, then used to interpret the next received (or transmitted) character. So, three factors determine the active character set.

- Which character area is active: GL or GR
- Which alternate set is mapped into the active area: G0, G1, G2, or G3
- Which character set is loaded in the alternate (G0 or G1) set: ASCII_G or multinational

Figure 2-6 shows how you can map the ASCII_G and multinational sets into the alternate character set buffers. Figure 2-7 shows how you can select active character sets.

Table 2-2 gives the escape sequences and control characters that load and select alternate character sets. Table 2-3 gives the escape sequences to load active character sets into G0 through G3.

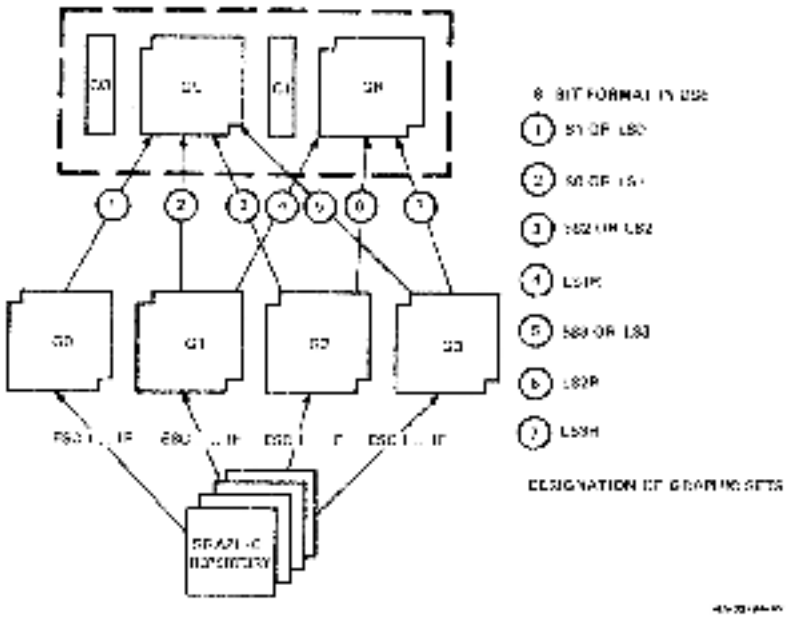

Figure 2-6 Loading 8-Bit Characters

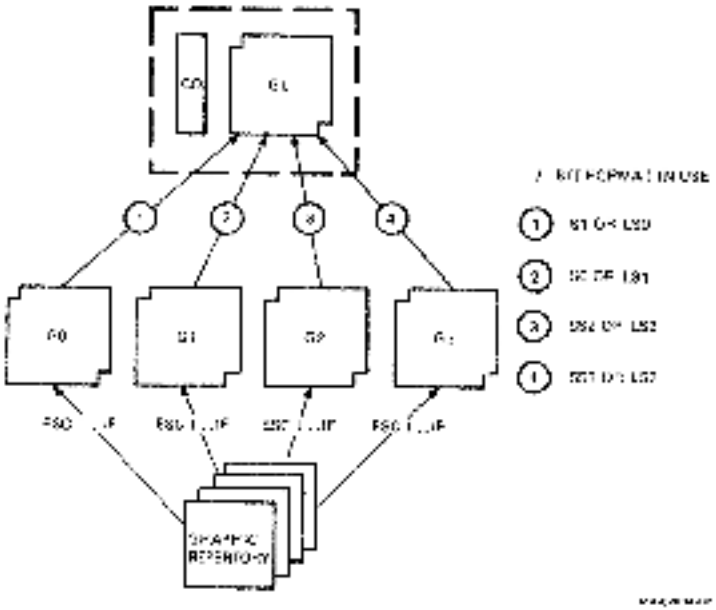

Figure 2-7 Selecting Active Character Sets

Table 2-2. Selecting the Active Character Set

Command	Mnemonic	Escape Sequence	Graphics Set		Table Position
Locking shift 0	LS0	SI 015	G0	→	GL
Locking shift 1	LS1	SO 014	G1	→	GL
Single shift 2	SS2	ESC N 027 078	G2*	→	GL
Single shift 3	SS3	ESC O 027 079	G3*	→	GL
Locking shift 2	LS2	ESC n 027 110	G2	→	GL
Locking shift 3	LS3	ESC o 027 111	G3	→	GL
Locking shift 1 right	LS1R	ESC ~ 027 126	G1	→	GR
Locking shift 2 right	LS2R	ESC } 027 125	G2	→	GR
Locking shift 3 right	LS3R	ESC | 027 124	G3	→	GR

* SS2 (single shift 2) and SS3 (single shift 3) are special cases. These commands select the next character value from the G2 or G3 set, respectively, regardless of the setting of the eighth bit.

Table 2-3. Selecting the Character Set

Character Set	G0	G1	G2	G3
ASCII	ESC (B 027 040 066	ESC) B 027 041 066	ESC * B 027 042 066	ESC + B 027 043 066
Multinational	ESC (< 027 040 060	ESC) < 027 041 060	ESC * < 027 042 060	ESC + < 027 043 060

DEC Multinational Character Set

By factory default, when you power up or reset DECtalk, the DEC multinational character set is mapped into the 8-bit code matrix (columns 0 through 15). Figure 2-8 shows the DEC multinational character set.

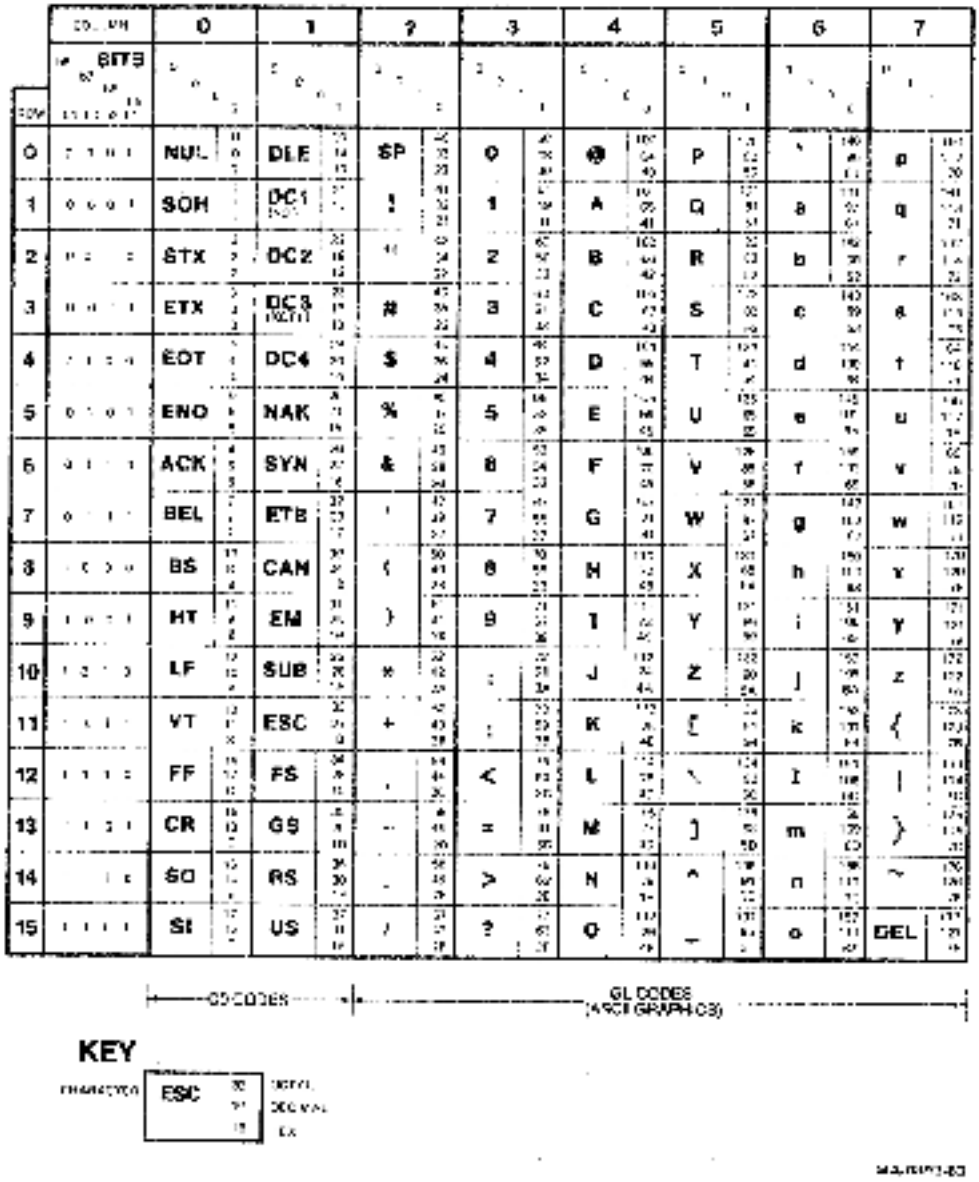

Figure 2-8 DEC Multinational Character Set (Left Half)

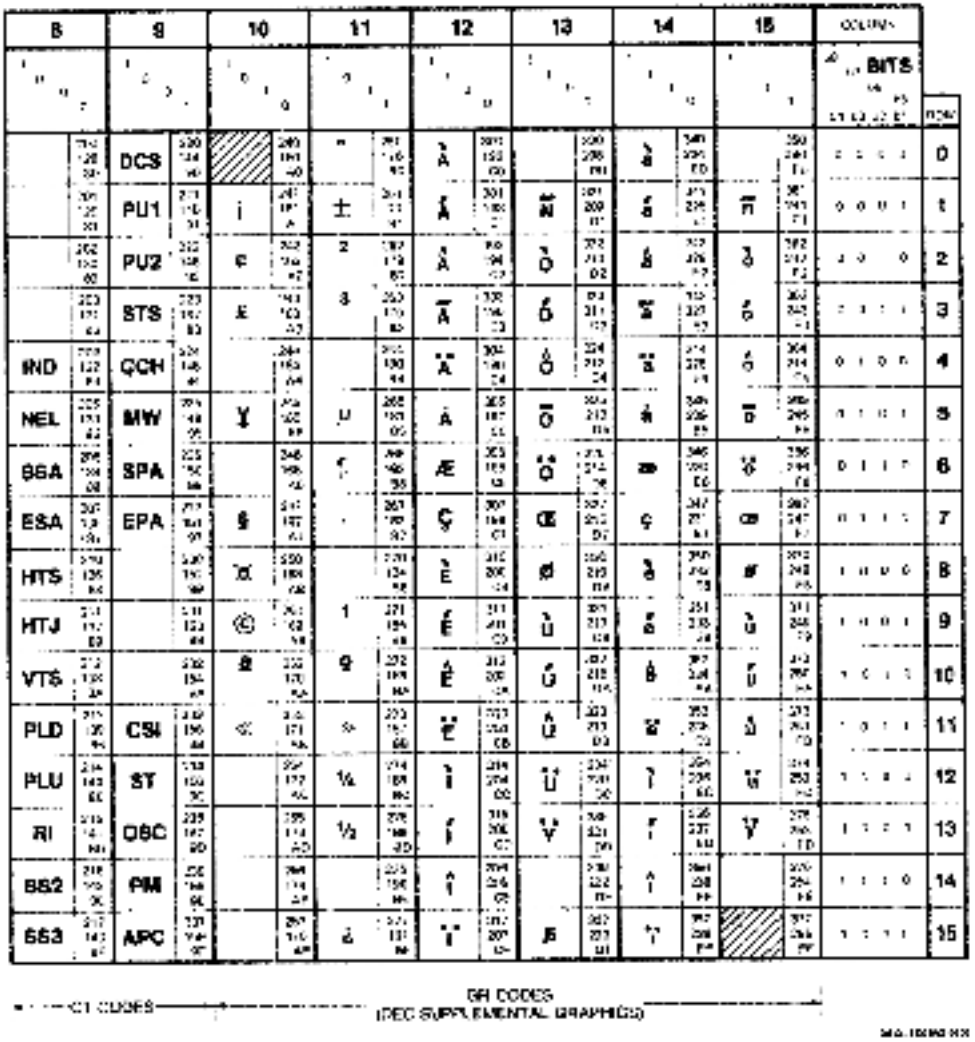

Figure 2-8 DEC Multinational Character Set (Right Half)

The 7-bit compatible left half of the DEC multinational character set is the ASCII graphics set: the C0 codes are the ASCII control characters and the GL codes are the ASCII graphics set.

The 8-bit compatible right half of the DEC multinational character set includes the C1 8-bit control characters in columns 8 and 9. The GR codes are the DEC supplemental graphics character set. The DEC supplemental graphics character set has alphabetic characters with accents and diacritical marks that appear in the major Western European alphabets. It also has other symbols not included in the ASCII graphics character set.

DECtalk removes the accent from characters in the supplemental graphics character set, which are accented versions of characters in the ASCII graphics set. (Naïve is the same as naive.) Other supplemental graphic characters are ignored.

WORKING WITH 7-BIT AND 8-BIT ENVIRONMENTS

To take advantage of DECtalk's 8-bit character set, your program and communication environment must be 8-bit compatible.

Conventions for Codes Transmitted to the Terminal

DECtalk expects to receive character codes in a form consistent with 8-bit coding. Your application can freely use the 8-bit codes as well as the 7-bit code extensions if it has enabled 8-bit controls.

When your program sends GL or GR codes, DECtalk interprets these according to the graphic character mapping currently being used. The factory default mapping, which is set when you power up or reset DECtalk, is the DEC multinational character set.

Mode Selection (DT_MODE)

This sequence acts like the SET MODE command in setup mode. DT_MODE controls how DECtalk handles particular characters in spoken text. The general DT_MODE escape sequence is as follows.

ESC	**P**	**0**	**;**	**8**	**0**	**;**	**P3**	**z**	**ESC**	****
027	080	048	059	056	048	059	***	122	027	092

Use the following method to obtain the P3 value.

1. Add up the values of the MODE flags in Table 2-4 that you want to use.

2. Convert the sum to ASCII digits. Use these digits in place of P3 in the escape sequence.

For example, assume you want to set MODE_SQUARE and MODE_MINUS, and clear MODE_ASKY.

```
MODE_SQUARE      = 1
MODE_MINUS       = 4
                 ----

Desired P3 value = 5
```

ESC	P	0	;	8	0	;	5	z	ESC	\
027	080	048	059	056	048	059	053	122	027	092

Table 2-4 DT_MODE Parameters

Mnemonic	Value	Function	
MODE_SQUARE	1	DECtalk interprets the following graphic characters as commands.	
		[ ]	Delimit phonemic text strings.
		)word	You can use a right parenthesis immediately before a word to select an alternate pronunciation for certain words, such as read, whose pronunciation depends on usage.
MODE_ASKY	2	DECtalk interprets phonemic text in single-character "asky" (ASCII) instead of multicharacter "arpabet" phonemic mode. This flag also determines which phonemic representation is used by DECtalk phonemic logging. See the *DECtalk DTC01 Owner's Manual* for more information.	
MODE_MINUS	4	DECtalk pronounces the hyphen (-) character as "minus." Usually, it is pronounced "dash."	

VOICE COMMANDS, PHONEMIC TEXT, AND THE USER DICTIONARY 3

The *DECtalk DTC01 Owner's Manual* describes how to modify the DECtalk voice (using phonemic commands and the phonemic alphabet) from a terminal. This chapter describes a special series of escape sequences that gives a host computer slightly greater control over DECtalk. For example, escape sequences can turn the DECtalk voice on or off and load the user dictionary.

SPEECH CONTROL

There are three ways to control DECtalk speech.

1. *Through English text* (sentences in standard English format and spelling). DECtalk speaks this text as written.

2. *Through phonemic spelling* (sentences or phrases written in phonemic symbols). Phonemic spelling is closer to the actual pronunciation of the text.

3. *Through phonemic commands.* Phonemic commands control features of speech that are not obvious from the visible text, such as rate of speech, sex of the speaker, and excitement level.

SPEECH TIMEOUT

Usually, DECtalk does not begin speaking until the host computer sends a clause terminator (period, comma, exclamation point, or question mark); however, there is a 5-second timeout limit. If the host does not send data within 5 seconds, DECtalk speaks the pending text in its input buffer, as if a comma had been sent.

Programs with long interruptions (such as pauses to search a database) should collect complete sentences before sending anything to DECtalk. Otherwise, this timeout may cause unnatural breaks in sentences and jerky-sounding speech.

ENGLISH TEXT

DECtalk speaks sentences written in standard English, if the text follows three rules.

1. Sentences end with a period, exclamation point, or question mark.

2. All commas, periods, exclamation points, and question marks are followed by a space (or an equivalent character from Table 1-1).

3. A period must be followed by enough text to distinguish between abbreviations and the end of a sentence.

The host computer can send English text in paragraph format; that is, sentences can be broken in the middle by carriage returns.

If a sentence is too long to store in DECtalk's buffers, the sentence is spoken in sections. DECtalk breaks up the sentance and speaks it as if clause boundaries were present; the effect is similar to a person trying to speak a long sentence and running out of breath. Keep sentances down to a reasonable length to avoid this effect.

See the *DECtalk DTC01 Owner's Manual* for more information on speech phrasing and emphasis. The "Data Synchronization" section in this chapter also describes how to coordinate speech and interaction commands to prevent loss of information.

SPEAK PHONEMIC TEXT (DT_PHOTEXT)

When MODE SQUARE is on, you can embed phonemic text in normal text with square brackets. When sending data from the host computer, you can use the DT_PHOTEXT escape sequence as well as the square brackets; MODE SQUARE does not have to be on. The DT_PHOTEXT escape sequence is as follows.

ESC	P	0	;	0	z	text	ESC	\
027	080	048	059	048	122	.....	027	092

ESC P 0 ; 0 z is the same as a left bracket ([), and ESC \ is the same as a right bracket (]). DECtalk uses phonetic speech for all text between the command terminator z and sequence terminator ESC \.

Appendix C lists the phonemic alphabet used by DECtalk. The *DECtalk DTC01 Owner's Manual* describes the alphabet in detail.

Within the phonemic text string, the host computer can transmit comments (for program maintenance) enclosed in /* and */ sequences. (An ESC \ can also terminate any comment.)

For example, in the following sequence the word Hello is a comment.

ESC P 0 ; 0 z hx'ehlow /* Hello */ ESC \

DECtalk processes a phonemic text escape sequence as though the introducer and terminator were spaces. This means phonemic text cannot replace part of a word.

In addition to transmitting the proper pronunciation, the phonemic text escape sequence can send control phonemes. This example changes the speech rate to 250 words per minute.

ESC P 0 ; 0 z :ra250 /* Rate – 250 wpm */ ESC \

NOTE: You cannot use STX (CTRL-B) and ETX (CTRL-C) to delimit phonemic text. Use the DT_PHOTEXT *escape sequence instead.*

STOP SPEAKING (DT_STOP)

This escape sequence immediately stops speech, even if DECtalk is in the middle of a sentence. DT_STOP is useful for stopping speech to perform other actions. For example, the user may press a key to get more instructions, warnings, or shortened versions of explanations (such as lengthy HELP information).

The DT_STOP escape sequence is as follows.

ESC	P	0	;	1	0	z	ESC	\
027	080	048	059	049	048	122	027	092

Speech stops immediately and all internal buffers are reinitialized.

DATA SYNCHRONIZATION (DT_SYNC)

The application program can send data to DECtalk faster than DECtalk can speak it. If the user must carry on a dialogue with the application program (through the telephone keypad), the application program should know whether or not DECtalk has finished speaking the text sent to it. DT_SYNC provides this coordination between the application program and DECtalk speech.

When the host sends DT_SYNC, DECtalk finishes speaking any pending text before processing the next command from the host. This ensures that the user hears a message before any other action starts, such as hanging up the phone or starting the phone timeout clock. Note that DT_SYNC acts as a clause boundary, the same as a comma, period, exclamation point, or question mark.

DECtalk considers a section of text to be spoken as soon as the parameters for that section are successfully sent to its signal processing section. Audio output runs approximately 6 milliseconds behind the transmission of the parameters. Applications that switch the audio output of a single DECtalk to a number of sites may need to take this delay into account.

The DT_SYNC escape sequence is as follows.

ESC	P	0	;	1	1	z	ESC	\
027	080	048	059	049	049	122	027	092

DT_SYNC does not reply to the host when processing is complete. However, you can do this by following the DT_SYNC command with a DT_INDEX_QUERY command.

ENABLE OR DISABLE SPEAKING (DT_SPEAK)

The DT_STOP sequence stops speech in progress. The DT_SPEAK sequence turns speech processing off or on, so received text is either spoken or discarded. DT_SPEAK is useful if the host computer can recognize such things as electronic mail letterheads and discard them as unnecessary. The host can act as a filter, removing extraneous speech.

The DT_SPEAK escape sequence is as follows.

ESC	P	0	;	1	2	;	P3	z	ESC	\
027	080	048	059	049	050	059	***	122	027	092

If P3 is 0, DECtalk stops speaking text; that is, it stops passing characters received from the host to the text-to-speech processing section. If P3 is not 0, DECtalk resumes speaking.

DECtalk also resumes speaking if the host sends DT_SYNC, DT_STOP, RIS, DECSTR, or DT_PHONE:ph_answer.

INDEXING

Text sent to DECtalk can contain index marks. DECtalk remembers these marks when they are spoken. The host application can listen to the spoken text (by reading the value of the last index) to determine how much transmitted text was actually spoken.

Index markers affect the way numbers and abbreviations are spoken. For example, DECtalk says $ 12.45 as "twelve dollars and forty-five cents." (The space after the $ is optional.) If an index marker separates the $ and 1, then DECtalk says "dollar twelve point four five."

The following paragraphs describe how to mark text and return their values to the application program.

Index Text (DT_INDEX)

This sequence inserts an index marker (flag) in the text stream sent to DECtalk.

The DT_INDEX escape sequence is as follows.

ESC	P	0	;	2	0	;	P3	z	ESC	\
027	080	048	059	050	048	059	***	122	027	092

The P3 parameter may range from 0 to 32767, sent as the ASCII characters for the number. Numbers outside the range are brought into range by masking off the overflow bits.

For example, the host computer sends the following data stream to DECtalk and marks the second word with the index 15.

Hello ESC P 0 ; 2 0 ; 1 5 z ESC \ there.

After speaking the text before DT_INDEX, DECtalk remembers the value 15. The host may use DT_INDEX_QUERY (described later in this chapter) to get this stored value.

Index Reply (DT_INDEX_REPLY)

DT_INDEX simply marks a position in the text. DT_INDEX_REPLY marks a position, but also has DECtalk inform the host when the index is spoken.

The DT_INDEX_REPLY escape sequence is as follows.

ESC	P	0	;	2	1	;	P3	z	ESC	\
027	080	048	059	050	049	059	***	122	027	092

The P3 parameter is in the range 0 to 32767, using ASCII characters for the selected number.

When DECtalk speaks the DT_INDEX_REPLY sequence, it sends a reply (containing the P3 parameter of the index) to the host. The escape sequence reply format is as follows.

ESC	P	0	;	3	1	;	P3	z	ESC	\
027	080	048	059	051	049	059	***	122	027	092

P3 has the original value specified in DT_INDEX_REPLY.

Index Query (DT_INDEX_QUERY)

DT_INDEX_QUERY requests DECtalk to reply to the host with the last index marker spoken (that is, the last portion of spoken text that had an index marker). The DT_INDEX_QUERY escape sequence is as follows.

ESC	P	0	;	2	2	z	ESC	\
027	080	048	059	050	050	122	027	092

DECtalk immediately returns a DECtalk reply escape sequence to the host in the following format.

ESC	P	0	;	3	2	;	P3	z	ESC	\
027	080	048	059	051	050	059	***	122	027	092

P3 contains the last index spoken. The P3 value is ASCII 0 under any of the following conditions.

- The last index passed was ASCII 0.
- No index has been passed yet.
- No index has been marked in the text; that is, the host has not sent a DT_INDEX or DT_INDEX_REPLY sequence.

Figure 3-1 shows how DT_SYNC and DT_INDEX_QUERY can coordinate host–DECtalk communications.

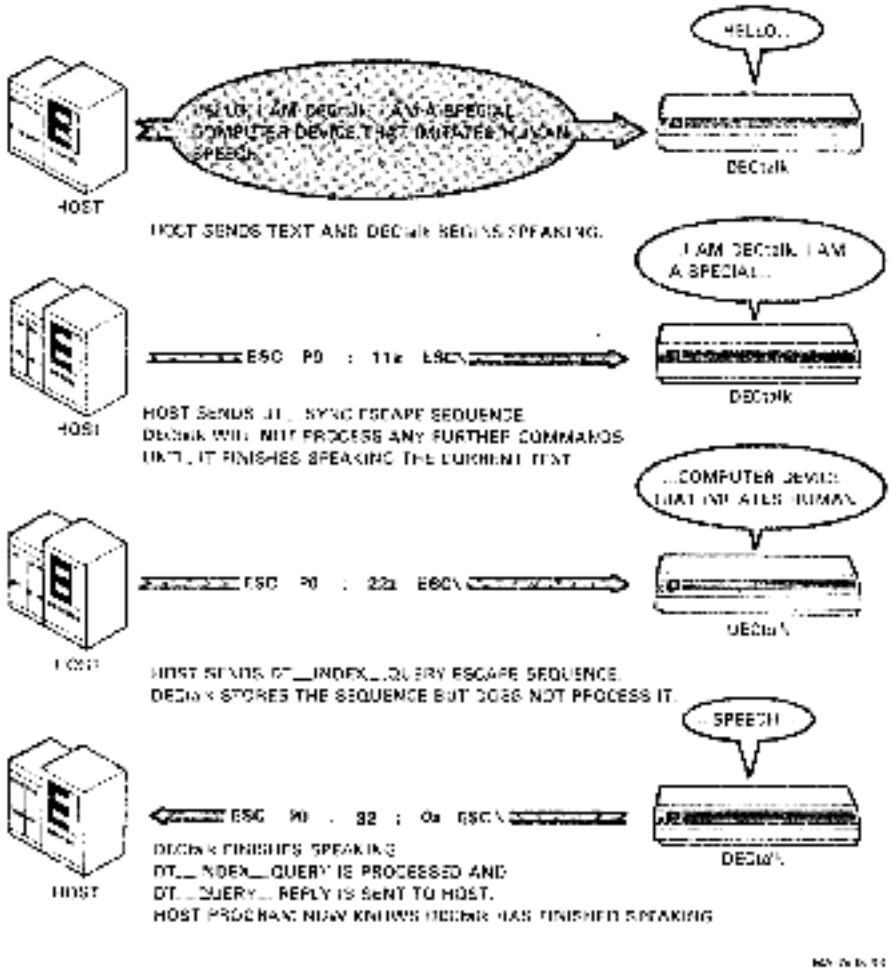

Figure 3-1 Using DT_SYNC and DT_INDEX_QUERY to Coordinate Communications

LOAD DICTIONARY (DT_DICT)

The user dictionary is used for processing abbreviations, and for providing phonemic equivalents of unusual words. The DT_DICT escape sequence is as follows.

ESC	**P**	**0**	**;**	**4**	**0**	**z**	**name substitution**	**ESC**	****
027	080	048	059	052	048	122	....	027	092

Whenever the word represented by "name" appears in the input text, the phonemic pronunciation given by "substitution" is used.

Any uppercase characters in the name only match uppercase characters in the input text. Lowercase characters in the name match both uppercase and lowercase characters in the input text. DECtalk always searches dictionary entries in the order entered.

If a name ends with a period (.), a period must follow the word in running input text. This period is included as part of the word, and is not recognized as a sentence terminator.

Here are some examples of dictionary entries.

```
ESC P 0 ; 4 0 ; z ms m'ihz ESC \
ESC P 0 ; 4 0 ; z ms. m'ihz ESC \
ESC P 0 ; 4 0 ; z DEC d'ehk ESC \
ESC P 0 ; 4 0 ; z dec d'iysahmber ESC \
ESC P 0 ; 4 0 ; z Goethe g'owth`iy ESC \
ESC P 0 ; 4 0 ; z GOSLOW :ra 120 ESC \
```

DECtalk does not recognize an error in phonemic spelling until the word is used. You can use comments in the substitution, but they are not recommended. Note the use of capitalization in the previous examples to distinguish between abbreviations with the same spelling.

If you do not enter a substitution, DECtalk removes the word from the user text dictionary. You cannot remove words from the built-in dictionary.

After loading the word and its definition, DECtalk replies with a dictionary status report.

ESC	P	0	;	5	0	;	P3	z	ESC	\
027	080	048	059	053	048	059	***	122	027	092

P3 may have one of the following values.

0 (048) Word entered correctly.

1 (049) No room in dictionary.

2 (050) Entry too long (256 characters maximum).

4 TELEPHONE COMMUNICATIONS

You can connect DECtalk to the public telephone system to provide a dial-up link between remote users on telephones and a computer application program. DECtalk sends and receives information as a link between a remote user and the host computer.

DECtalk communicates with the phone through the voice circuits, passing on spoken data to the listener. DECtalk passes information back to the host both as ordinary ASCII characters, and as escape sequences. The user can communicate with the host (through DECtalk) by using the Touch-Tone keypad, if available.

The DT_PHONE escape sequence is the controlling sequence for all telephone operations.

TELEPHONE MANAGEMENT (DT_PHONE)

This escape sequence takes one or more parameters and controls the attached telephone and Touch-Tone keypad interface. The DT_PHONE escape sequence is as follows.

ESC	**P**	**0**	**;**	**6**	**0**	**;**	**Pn**	**;**	**Pn**	**z**	**text**	**ESC**	****
027	080	048	059	054	048	059	***	059	***	122	.....	027	092

The Pn parameters act as a list of telephone management commands and execute in sequence. Table 4-1 lists the valid Pn parameters.

A single DT_PHONE sequence can perform several commands. Some commands can take additional parameters.

All DT_PHONE commands return a status report to the host in the following escape sequence. Table 4-2 lists the valid P3 values.

ESC	**P**	**0**	**;**	**7**	**0**	**;**	**P3**	**z**	**ESC**	****
027	080	048	059	055	048	059	***	122	027	092

All telephone management commands return a reply sequence back to the host upon command execution. PH_STATUS is only needed to check the telephone status when a DT_PHONE command is not pending. Note that PH_ANSWER generates an additional status report when the phone is answered.

Telephone keypad characters are sent as text, not escape sequences.

Note that the R3_PH_TIMEOUT reply sequence is sent when a timeout occurs; that is, the reply sequence may arrive as unrequested input, and the application program must be ready to receive it.

Table 4-1. BT_PHONE Parameters

Mnemonic	ASCII Code Decimal Value		Function
PH_STATUS	0 048		Send a telephone status report.
PH_ANSWER	1 049	0 048	Enable pulse answer of the telephone.
PH_HANGUP	1 049	1 049	Hang up the telephone and disable the keypad.
PH_KEYPAD	2 050	0 048	Enable the keypad and select direct keypad decoding.
PH_NOKEYPAD	2 050	1 049	Disable the keypad (without hanging up the telephone).
PH_TIMEOUT	3 051	0 048	Enable timeouts on telephone keypad input. Timeout equals P1 parameter value.
PH_TONE_DIAL	4 052	0 048	Dial an outgoing phone call using Touch-Tones.
PH_PULSE_DIAL	4 052	1 049	Dial an outgoing phone call using pulse dialing.

Table 4-2. Phone Status Reply Codes.

Mnemonic	ASCII Code Decimal Value	Function
R3_PH_ONHOOK	0 048	Telephone is on-hook (hung up).
R3_PH_OFFHOOK	1 049	Telephone is off-hook (active).
R3_PH_TIMEOUT	2 050	No response after TIMEOUT command.
R3_PH_TOOLONG	3 051	Number dialed is longer than 256 characters.

PH_ANSWER
DECtalk is set up to answer incoming phone calls. The parameter that follows the PH_ANSWER parameters indicates the number of rings to wait before answering the telephone. A parameter of 0 or 1 means answer the telephone after the first ring; 2 means answer after 2 rings, and so on.

If the telephone is off-hook when the host sends a PH_ANSWER parameter, DECtalk hangs up the telephone (disconnects any active call) before executing the PH_ANSWER command.

DECtalk sends two status replies to a PH_ANSWER request. The first status reply informs the host that the DT_PHONE command was correctly received. The second reply informs the host that the telephone has actually been answered.

DECtalk stops waiting for incoming calls whenever the host sends PH_HANGUP, PH_TONE_DIAL, PH_PULSE_DIAL, RIS, or DECSTR.

PH_HANGUP
This command hangs up the telephone. The status reply is delayed until the telephone is back on-hook (disconnected). The host should wait for the R3_PH_ONHOOK reply before sending other commands to DECtalk.

PH_KEYPAD
This command enables the telephone keypad. The request is ignored if the phone is inactive (on-hook); however, DECtalk returns an R3_PH_ONHOOK status reply.

PH_NOKEYPAD
This command disables the telephone keypad, but maintains the phone connection. This request is ignored if the phone is inactive (on-hook); however, DECtalk returns an R3_PH_ONHOOK status reply.

PH_TIMEOUT
This command starts (or restarts) an internal DECtalk timer. If the user does not press a telephone keypad button within the timeout interval, an R3_PH_TIMEOUT status is returned (Table 4-2).

The application program should set PH_KEYPAD on before sending a PH_TIMEOUT command; otherwise, the user cannot respond to DECtalk requests for input.

The parameter following PH_TIMEOUT is the number of seconds to wait for a response from the caller. A parameter of 0 cancels any active timeouts. After a timeout, the timer is stopped. The application program must send a new PH_TIMEOUT command to restart the timer.

Timeouts are the only way to detect that the caller has hung up the telephone.

The public telephone system in your country may have another timeout requirement, independent of DECtalk. If this is true, a phone call may be automatically terminated (hung up) if a response is not given in a certain length of time. Your application program should accept unsolicited R3_PH_ONHOOK replies.

PH_TONE_DIAL and PH_PULSE_DIAL

DECtalk can dial an outgoing call by using these two commands. If DECtalk is connected to a Touch-Tone public telephone network, then use the PH_TONE_DIAL parameter; otherwise, use PH_PULSE_DIAL. PH_PULSE_DIAL works like an old rotary phone dial.

If the telephone is on-hook when the host sends a dialing command, DECtalk picks up the telephone and inserts a 2-second delay.

The text between the command terminator z and the ESC \ sequence is the number to dial. For the Touch-Tone dialing system, the characters 0123456789*#ABCD!^ are recognized. For the pulse dialing system, the characters 0123456789!^ are recognized.

The ! character inserts a 1-second delay into the dialing stream. DECtalk pauses during the dialing sequence every time it finds a ! character.

On some telephone systems, a user can press the switch hook to transfer calls or otherwise interrupt a phone call. This signal is called a switch-hook flash. The ^ character inserts a 250-millisecond switch-hook flash signal into the dialing stream. You can use successive ^ characters to generate longer flashes.

With Touch-Tone dialing, the characters ABCD generate the extra four tones of the military handset. A is the character to the right of the 3, B is the character below it, and so on.

Figure 4-1 shows a complete phone call session, including a timeout sequence initiated because a user disconnected.

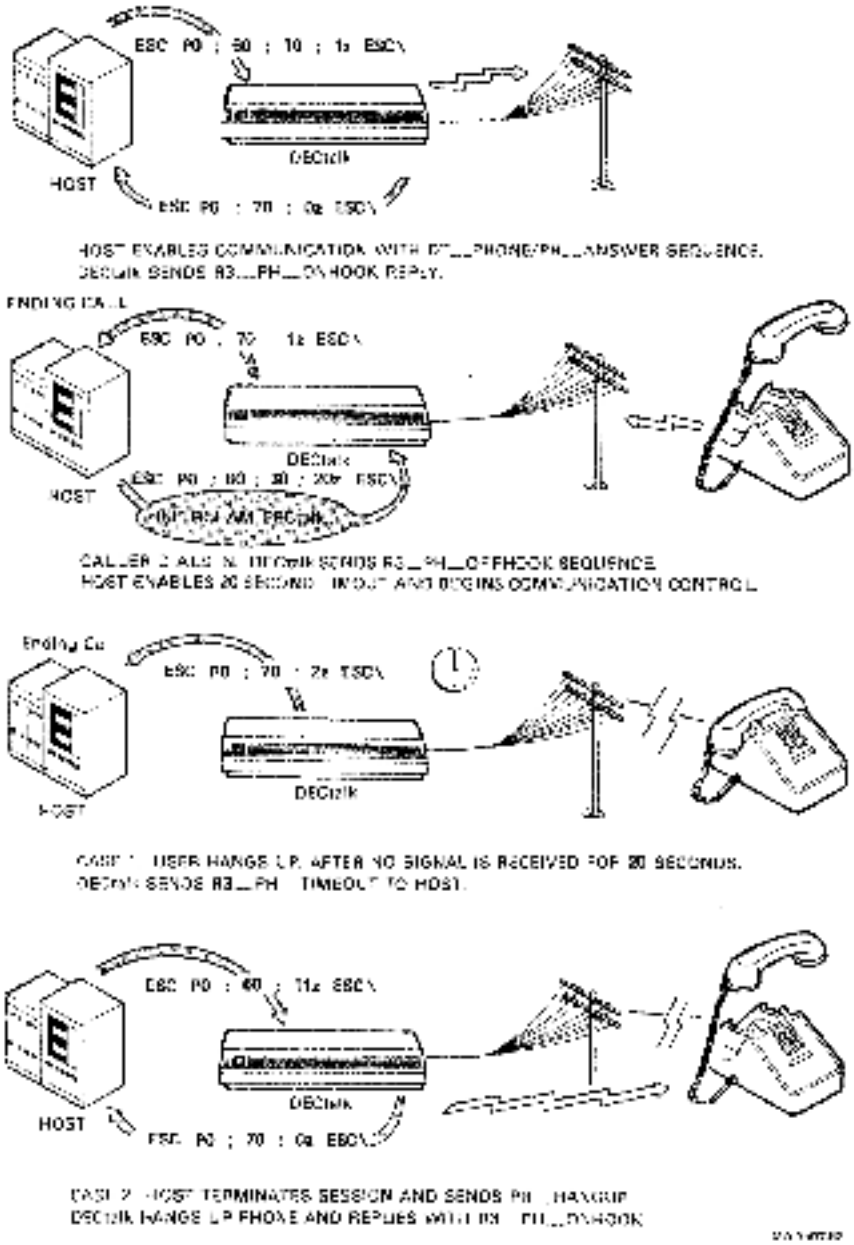

Figure 4-1 Telephone Communications

MAINTENANCE AND DEBUGGING COMMANDS 5

DECtalk has a set of commands that set DECtalk operating features, test DECtalk, and help debug application programs. Most of these commands have an inquiry-response format. DECtalk returns an answer to the host computer after the action is complete (or in response to a pure inquiry).

DEVICE ATTRIBUTE REQUEST

DECtalk responds to device identity requests from the host computer. For compatibility with an older escape sequence, DECtalk recognizes two different request sequences, described in the following paragraphs.

Device Attribute Request (DA Primary)

The preferred device attribute request escape sequence is as follows.

ESC	[	0	c
027	091	048	099

DECtalk identifies itself by sending the following sequence.

ESC	[	?	1	9	c
027	091	063	049	057	099

DECtalk does not respond to secondary device attribute requests, since its product identification code is less than 50.

Identify Terminal (DECID)

DECtalk responds to the old identify terminal sequence (DECID) exactly as it responds to the DA Primary request.

The old identify sequence is as follows.

ESC	Z
027	090

DECtalk identifies itself by sending the following sequence.

ESC	[	?	1	9	c
027	091	063	049	057	099

This is the same sequence as the DA Primary answer.

DEVICE TEST AND STATUS

A special set of escape sequences run DECtalk hardware self-tests. Another set of escape sequences forces DECtalk to return status reports. The following paragraphs describe these sequences.

DECtalk Power-Up Status

You can reset DECtalk to its power-up state. The method you use to reset DECtalk may affect the operating features (such as baud rate). You can reset DECtalk with any of the following methods. This chapter describes methods 2 through 4.

1. Power-up (PUP) is the state that DECtalk is in when first turned on.

2. Return to initial state (RIS) is a hard reset you can set with an escape sequence.

3. Soft reset (DECSTR) partially restores DECtalk to its power-up state.

4. Nonvolatile memory reset (DECNVR) lets you reset the operating features in permanent memory. At power-up, DECtalk restores the feature settings that you reset in this memory.

Table 5-1 lists the DECtalk operating features and their factory default settings; the reset methods in column three restore the feature to its power-up setting.

The power-up setting is the factory default, unless you changed the setting and stored it with a DECNVR sequence. See the section on DECNVR in this chapter.

Table 5-2 lists some other DECtalk actions performed by certain reset methods.

Table 5-1 Restoring DECtalk Operating Features

Feature	Factory Default	Restored from NVR by
Local line speed	9600 baud	PUP, DECNVR
Local line format	No parity	PUP, DECNVR
Host line speed	1200 baud	PUP, DECNVR
Host line format	No parity	PUP, DECNVR
Host DT transmit mode	7 bit	PUP, DECNVR
Host DT receive mode	7 bit	PUP, DECNVR
Local log tags	0	PUP, DECNVR, RIS
Local terminal tags	6	PUP, DECNVR, RIS

Table 5-2 DECtalk Actions Performed at Resets

DECtalk Action	Performed By
Implied DT, SPEAK	PUP, RIS, DECSTR
Hang up telephone	PUP, RIS, DECSTR
Delete user dictionary	PUP, RIS
Flush all pending text	PUP, RIS
Turn fast speech on	RIS, DECSTR

Device Self-Test (DECTST)

This sequence initiates local self-tests. The escape sequence is as follows.

ESC	[	5	;	Pn	y
027	091	053	059	***	121

The Pn parameter specifies the test to perform (Table 5-3).

The TEST_POWER parameter (Pn = 1) causes DECtalk to rerun its power-up initialization and test sequences. ALL DECtalk operating features return to the power-up state; the telephone is hung up, the user dictionary is deleted, and all features are reset to their power-up values.

The loopback tests require the appropriate loopback connectors.

The built-in message provides a quick check of the DECtalk system. The message includes the version number of the DECtalk firmware.

Table 5-3 Self-Test Parameters

Mnemonic	ASCII Code Decimal Value	Function
TEST_POWER	1 049	Rerun power-up tests
TEST_HDATA	2 050	Run host port data loopback test
TEST_HCONTROL	3 051	Run host port control loopback test
TEST_DATA	4 052	Run local port data loopback test
TEST_SPEAK	5 053	Speak a built-in message

Device Status Request (DSR) (Brief Report)
The brief DSR escape sequence is as follows.

ESC	[	5	n
027	091	053	110

If no malfunctions are detected, DECtalk replies with the following sequence.

ESC	[	0	n
027	091	048	110

If a malfunction is detected, DECtalk replies with the following sequence.

ESC	[	3	n
027	091	051	110

Applications can use this brief DSR format in most cases, because a brief request does not reset any of DECtalk's internal error flags. The following extended DSR format is useful when a malfunction is detected.

Device Status Request (DSR) (Extended Report)
The extended DSR escape sequence lets an application program determine when DECtalk was first powered on. The application sends the extended DSR escape sequence as follows.

ESC	[	n
027	091	110

If no malfunctions are detected, DECtalk replies with one of two sequences. If this is the first extended DSR since DECtalk was powered on, DECtalk replies with the following sequence.

ESC	[	0	n	ESC	[	?	2	1	n
027	091	048	110	027	091	063	050	049	110

For later requests, DECtalk replies with the following sequence.

ESC	[	0	n	ESC	[	?	2	0	n
027	091	048	110	027	091	063	050	049	110

If a malfunction is detected, DECtalk sends the following sequence.

ESC	**[**	**3**	**n**	**ESC**	**[**	**?**	**Pn**	**;**	...	**Pn**	**n**
027	091	051	110	027	091	063	***	059	...	***	110

Each Pn parameter specifies an error as follows. The extended status request sequence resets the error flags.

2	**2**	Communication failure.
050	050	

2	**3**	Input buffer overflow.
050	051	

2	**4**	Last NVR operation failed.
050	052	

2	**5**	Error in phonemic transcription.
050	053	

2	**6**	Error in DECtalk private control sequence.
050	054	

2	**7**	Last DECTST failed.
050	055	

Reset to Initial State (RIS)

Table 5-1 shows how the reset to initial state affects DECtalk. The RIS escape sequence is as follows.

ESC	**c**
027	099

This sequence resets DECtalk to its power-up state, without changing the speeds or data formats used on the host and local communication lines. All pending, unspoken text is lost. All user-defined dictionary entries are deleted. The telephone is returned to the on-hook state. Some operating features are restored from nonvolatile memory (NVR).

The RIS sequence always turns host speech on, even if host speech is turned off by the setup commands.

This NVR recall is almost identical to a DECNVR recall from user memory. (See "NVR Parameters" in this chapter.) RIS does not change the line characteristics, and RIS updates the "status of the last NVR operation" flag reported by device status reply sequences.

Digital recommends always using the DT_PHONE:ph_hangup sequence to hang up the telephone. If DECtalk receives an RIS sequence when the telephone is off-hook, and reads the telephone status during the hangup, DECtalk may report an off-hook status (instead of the expected on-hook status).

Soft Terminal Reset (DECSTR)

Table 5-2 shows how the soft terminal reset affects DECtalk. The DECSTR escape sequence is as follows.

ESC	[	1	p
027	091	033	112

This sequence resets DECtalk to its power-up state, without changing the speeds or data formats used on the host and local communication lines, or resetting user convenience features on the local terminal. Pending, unspoken text is not lost. The telephone returns to the on-hook state.

Digital recommends always using the DT_PHONE:ph_hangup sequence to hang up the telephone. If DECtalk receives a DECSTR sequence when the telephone is off-hook, and reads the telephone status during the hangup, DECtalk may report an off-hook status (instead of the expected on-hook status).

The DECSTR sequence always turns host speech on, even if host speech is turned off by the setup commands.

NVR Feature Settings (DECNVR)

You can store operating feature settings permanently in nonvolatile memory (NVR). DECtalk restores these settings at the next power-up. To save or restore the current settings in NVR, use the following DECNVR escape sequence.

ESC	[	Pn	:	Pm	t	r
027	091	***	059	***	033	114

If Pn is 0, this sequence restores all feature settings from NVR. This action may change the speeds or data format of the serial lines, so communication with the host or local terminal may be lost. The user dictionary is not deleted. The telephone is not hung up.

If Pn is 1, this sequence stores all current feature settings in NVR. DECtalk stops processing host line commands until the feature settings are safely stored.

The Pm parameter specifies which NVR memory to use. Memory 0 is a read/write memory you can use to store feature settings. DECtalk normally uses memory 0 at power-up. Memory 1 is a read-only memory, and always contains the factory-default DECtalk feature settings. DECtalk uses memory 1 at power-up if memory 0 cannot be used. Diagnostics may use memory 1 to force DECtalk back to its factory settings.

DECtalk remembers the success or failure status of the last NVR operation command. A device status request (DSR) sequence can check this status.

TRACING AND DEBUGGING COMMANDS

You can set DECtalk to log its actions and reactions to various commands on the local terminal. These commands are useful for testing and debugging during application program development.

Local Log Control (DT_LOG)

This sequence controls the logging of trace and debugging information on the local terminal. DT_LOG works like the SET LOG command in setup mode. (See the *DECtalk DTC01 Owner's Manual*.) The DT_LOG escape sequence is as follows.

ESC	P	0	;	8	1	;	P3	z	ESC	\
027	080	048	059	056	049	059	***	122	027	092

Use the following method to obtain the P3 value.

1. Add up the values of the DT_LOG parameters in Table 5-4 that you want to use.

2. Convert the sum to ASCII digits. Use these digits in place of P3 in the escape sequence.

Table 5-4 DT_LOG Parameters

Mnemonic	Value	Function
LOG_TEXT	1	Enable ASCII text logging.
LOG_PHONEME	2	Enable phonemic text logging.
LOG_RAWHOST	4	Send all characters from the host to the terminal without inspection.
LOG_INHOST	8	Enable logging of text read from the host.
LOG_OUTHOST	16	Enable logging of text sent to the host.
LOG_ERROR	32	Enable the display of DECtalk error messages.
LOG_TRACE	64	Symbolically display all escape sequences that affect DECtalk operations.
LOG_DEBUG	128	Reserved for Digital internal use.

For example, assume you want to set LOG_TEXT and LOG_RAWHOST.

LOG_TEXT	= 1
LOG_RAWHOST	= 4
	———
Desired P3 value	= 5

ESC	P	0	;	8	1	;	5	z	ESC	\
027	080	048	059	056	049	059	053	122	027	092

Table 5-4 lists the P3 parameters. Figure 5-1 shows the data paths for logging and debugging.

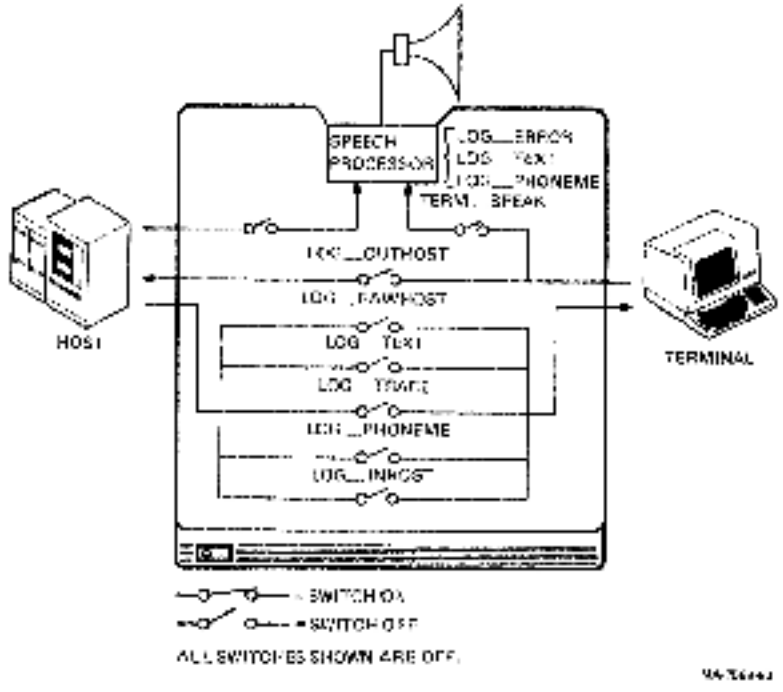

Figure 5-1 Data Paths for Logging and Debugging

LOG_TEXT
This command logs all spoken text. The text source does not matter; text is logged from both the host and the terminal.

LOG_PHONEME
This command logs all spoken text in its phonemic transcription. LOG_PHONEME is useful for testing the phonemic form of words and phrases.

LOG_RAWHOST
This command logs all control and text characters as received, except NUL characters (which are always deleted) and XON/XOFF characters (which still perform flow control functions).

LOG_INHOST
This command logs all characters received from the host. Control characters also print.

LOG_OUTHOST
This command logs all characters sent to the host. Control characters also print.

LOG_ERROR
This command logs all error messages. Usually DECtalk error messages are returned as escape sequences. Setting the LOG_ERROR flag causes error messages to be logged also.

LOG_ERROR is useful during the early stages of application program development.

LOG_TRACE
This command displays all escape sequences symbolically rather than as escape sequences. If you use LOG_TRACE in debugging, you do not have to look up the meaning of escape sequences.

LOCAL TERMINAL COMMAND (DT_TERMINAL)

This escape sequence controls the destination of characters typed on the local terminal when the terminal is not in setup mode. (The TERM FILTER parameter affects characters sent to the local terminal when the terminal is not in setup mode.) Figure 5-2 shows the data paths in local terminal operations.

The format of the DT TERMINAL escape sequence is as follows.

ESC	P	0	;	8	2	;	P3	z	ESC	\
027	080	048	059	056	050	059	***	122	027	092

Use the following method to obtain the P3 value.

1. Add up the values of the DT_TERMINAL parameters in Table 5-5 that you want to use.

2. Convert the sum to ASCII digits. Use those digits in place of P3 in the escape sequence.

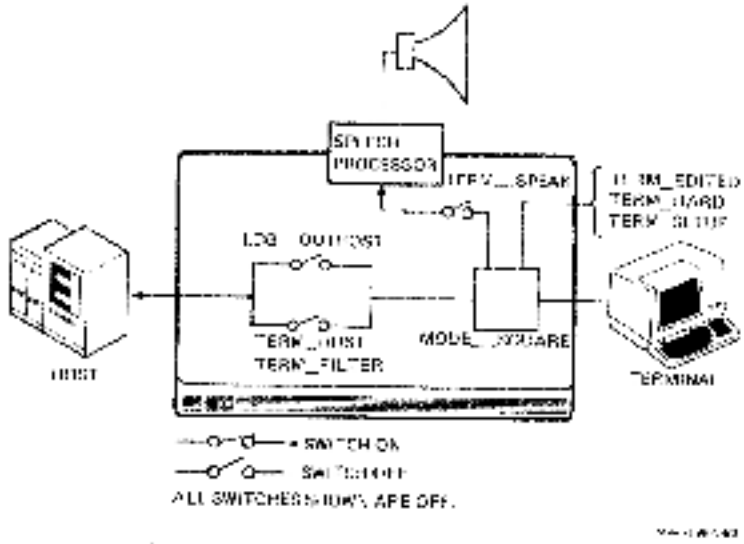

Figure 5-2 Data Paths for Local Terminal Operations

For example, assume you want to set TERM_HOST and TERM_EDITED.

```
TERM_HOST          = 1
TERM_EDITED        = 4
                   -----

Desired P3 value   = 5
```

ESC	P	0	;	8	2	;	5	z	ESC	\
027	080	048	059	056	050	059	053	122	027	092

Table 5-5 lists the possible P3 values.

NOTE: If LOG_RAW *and* TERM_HOST *are in effect and the host sends a device attribute request, both DECtalk and the terminal will respond. The application program sample in Chapter 6 turns off* TERM_HOST *for this reason.*

Table 5-5 DT_TERMINAL Parameters

Mnemonic	Value	Function
TERM_HOST	1	Send all characters typed on terminal to host line.
TERM_SPEAK	2	Speak all characters typed on terminal.
TERM_EDITED	4	Line edit all characters typed on terminal. (See the *DECtalk DTC01 Owner's Manual*.)
TERM_HARD	8	Do local terminal echo operations in hardcopy terminal format.
TERM_SETUP	16	Speak all characters displayed on the terminal when in setup mode.
TERM_FILTER	32	Do not send DECtalk-specific escapes sequences to the terminal.

NOTE: When you set TERM_FILTER, *DECtalk also ignores non-DECtalk escape sequences (except those needed for character set and communications setup). You still need to set* LOG_RAWHOST *or to have DECtalk send text to the local terminal.*

TERM_FILTER is useful when you use DECtalk as a link between a general-purpose operating system and an applications terminal. TERM_FILTER modifies the operation of LOG_RAWHOST to prevent sending DECtalk-specific escape sequences to the local terminal.

When you set TERM_FILTER, the following escape sequences usually processed by DECtalk are now only processed by the local terminal.

- Device self-test (ESC [5 ; Ps y)
- Brief device status request (ESC] 5 n)
- Extended device status request (ESC [n)
- Reset to initial state (ESC c)
- Soft terminal reset (ESC [! p)
- NVR parameters (ESC [Pn ; Pn ! r)
- Device attributes inquiry (ESC [0 c)
- Identify terminal (ESC z)

The following escape sequences are acted on by both DECtalk and the local terminal.

- Select active character set (several sequences)
- Select graphics repertory (ESC i B and ESC i <)
- Select 7-bit C1 transmission (ESC SP F)
- Select 8-bit C1 transmission (ESC SP G)
- Truncate high-order bit in C1 (ESC SP 6)
- Accept high-order bit in C1 (ESC SP 7)

Because TERM_FILTER must parse and understand escape sequences, you can only use TERM_FILTER when the local terminal supports ANSI escape sequences. Digital's VT100 and VT200 series terminals and the terminals communications programs available for Digital's personal computers support ANSI escape sequences.

KEYPAD MASK COMMAND (DT_MASK)

This command controls how DECtalk sends escape sequences and keypad characters to the host. DT_MASK simplifies application development when DECtalk is connected to a host via a packet-switched network or a network using the SNA (systems network architecture) protocol. These networks have a significant overhead associated with each message, so sending a line of text (several characters) is more economical than sending a single character.

DT_MASK is also useful when DECtalk is connected to an operating system that prefers to communicate line-by-line, rather than character-by-character. For example, when DT_MASK is on, you can use BASIC's INPUT LINE command to read text from DECtalk.

The command takes one parameter, which is interpreted as a 16-bit value. If a bit is set, DECtalk sends a carriage return after sending the associated keypad character. If any bit is set, DECtalk sends a carriage return after its escape sequence replies. (The carriage return follows the ESC \ string terminator.) The DT MASK escape sequence is as follows.

ESC	P	0	;	8	3	;	P3	z	ESC	\
027	080	048	059	056	051	059	***	122	027	092

The P3 parameter is bit-encoded. Specified values have associated characters (Table 5-6). If you specify a value, DECtalk sends a carriage return after the associated character (when a user presses that key).

Table 5-6 DT_MASK Parameters

Value	Bit	Character
1	0	0
2	1	1
4	2	2
8	3	3
16	4	4
32	5	5
64	6	6
128	7	7
256	8	8
512	9	9
1024	10	*
2048	11	#
4096	12	A
8192	13	B
16384	14	C
32768	15	D

For example, to have DECtalk treat the # and * characters as response terminators (but not the digits), a program would send the following sequence.

**ESC P ; 8 3 ; 3 0 7 2 z ESC **
027 080 059 056 057 059 051 048 055 050 122 027 092

(3072 = 1024 + 2048)

If the person calling the application presses **123#** followed by a keypad timeout, DECtalk would send the following.

1 2 3 # <carriage return>

**ESC P ; 7 0 ; 2 z ESC ** <carriage return>
027 080 059 055 048 059 050 122 027 092

This allows the application program to use standard line-oriented input routines, rather than character-oriented routines. If you specify a P3 parameter of 0 with DT_MASK, DECtalk will not send a carriage return after keypad characters or escape sequences.

NOTE: DECtalk will a send carriage return after all sequences, including responses to non-DECtalk-specific sequences such as device status request. Only responses generated within DECtalk are affected. Characters and escape sequences generated by a local terminal are sent without interpretation.

The DECtalk support library does not interpret carriage return characters. You have to process carriage returns with an application program. (Usually, an application will ignore them.)

DETERMINING FIRMWARE REVISION LEVEL

If your application environment has DECtalk units with different versions of firmware (1.8 and 2.0) you may need to determine the revision level of a particular unit. You can use the following steps to determine the firmware revision level.

1. Use the extended DSR escape sequence in this chapter to clear all DECtalk errors. (Remember to note the DECtalk reply.)

2. Use the following escape sequence to send a [+] phoneme.

   **ESC P ; z + ESC **
   027 080 059 122 043 027 092

   This is silent and new to revision level 2.0 only.

3. Send another extended DSR escape sequence. If DECtalk is a firmware revision level 1.8, it will report an error in the phonemic transcription. If DECtalk is revision level 2.0, it will not report any errors.

   ESC [0 n ESC [? 2 0 n
   027 091 048 110 027 091 063 050 048 110
   (firmware 2.0 report)

   ESC [3 n ESC [? 2 5 n
   027 091 051 110 027 091 063 050 053 110
   (firmware 1.8 report)

Phonemic Alphabet

Appendix A lists all phonemes you can use in the DECtalk phonemic alphabet.

C PROGRAM EXAMPLE 6

This chapter provides the source listings of a sample DECtalk application written in C programming language. The program uses DECtalk, a host computer, and a telephone connection to the United States public telephone network.

You can copy and use this application program; however, the program is only a model, and cannot cover all possible DECtalk applications. You will find many algorithms and sections within the application that you can use in your own program; however, you will probably have to modify large sections of this program for your own needs. Also, there is no guarantee that this application program will run in the same way on your computer or on your public telephone system (especially if you do not live in the United States).

The source programs are available from the DECUS Library (Digital Equipment Corporation User's Society) as 11-SP-58 (for PDP-11s) or V-SP-20 (for VAX/VMS). RSX and RSTS operating systems need a system services library distributed with DECUS C (DECUS 11-SP-18). All operating systems require a C compiler to compile the programs. The DECUS library also has versions of the library written in BASIC-PLUS and COBOL.

To order the latest version of source programs from the DECUS Library, mail your request to:

DECUS Order Processing
MR02-1/C11
One Iron Way
Marlboro, MA 01752

For general information before placing an order, call (617) 480-3422.

PROGRAM LANGUAGE AND STRUCTURE

This application program is written in C, a language originally written for the UNIX operating system. C is a highly structured language, similar to Pascal, ALGOL, and COBOL in form and syntax. C is also reasonably transportable: the application program shown here can run on RSTS/E, RSX, UNIX, or VAX/VMS operating systems (if the correct compilers are on those systems).

The application is written in many small modules, which are called according to a tree structure (Figure 6-1). There are many modules, because each module has only one or two functions within the program. The small, tight structure of each module means that their function is easy to read and grasp.

All variables, constants, and other special values are listed in one module: DECTLK.H. You must include DECTLK.H with the compilation of all other modules.

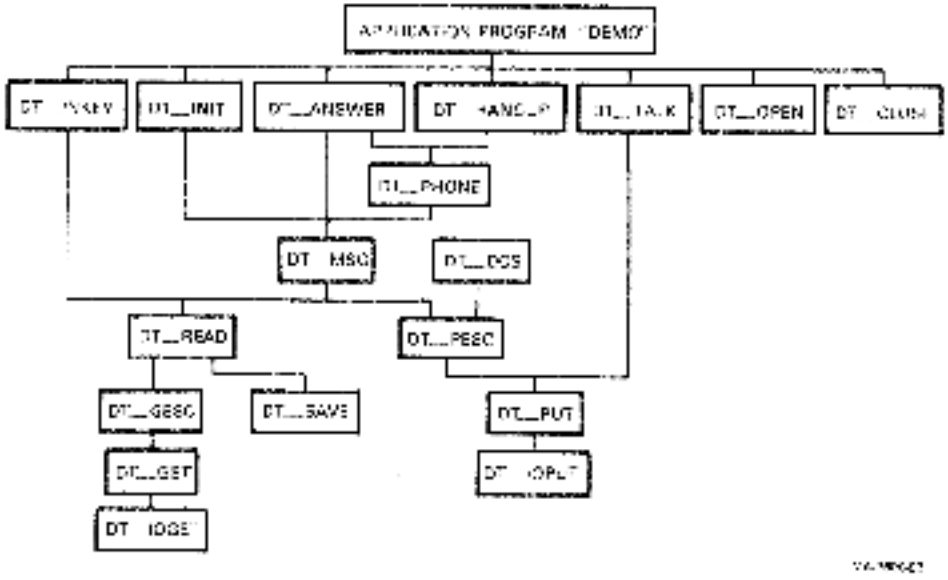

Figure 6-1 Calling Tree of DECtalk Application Program

HOW THE PROGRAM WORKS

The application program waits for a caller to dial the DECtalk phone number. DECtalk then acts as a link between the host computer and the caller, passing a canned message to the caller and informing the host when the caller presses any keypad buttons. DECtalk releases the phone line (1) when the caller hangs up, or (2) if no response is received after a certain length of time.

The program works as follows.

1. When started, the program establishes the DECtalk- telephone-host operating environment. DECtalk is set to wait for an incoming call.

2. When a phone call is received, DECtalk answers the call and informs the host that a call is active.

3. The host then sends a message for DECtalk to speak to the caller. This message informs the caller that the keypad can be used.

4. At the end of the host message, the telephone keypad is enabled and the caller can send responses back to the host.

5. The host responds with a "you pressed button ..." message when the caller presses a keypad button. If the caller doesn't press a button for 15 seconds, the host tells DECtalk to hang up the phone.

VARIABLE NAMES AND DEFINITIONS

All global variables and constants are defined in the module DECTLK.H. The function of an escape sequence or coded reply from DECtalk is not clear when embedded within a program; therefore, all escape sequences and status codes are given their mnemonic names in DECTLK.H. The program then refers to these names rather than the escape codes themselves.

What follows is a list of the global variables, mnemonics, and codes used in the application program.

Flags

Two flags are used throughout the application program. The flags control certain critical actions, as follows:

dt_abort	This flag is normally FALSE. If dt_trap() is called, the library will trap a CTRL-C (or INTERRUPT on UNIX). If the user types **CTRL-C**, the flag is set to TRUE and all library modules exit as quickly as possible.
dt_debug	This flag can be set nonzero by an application program to enable debug printouts. Note that the library must have been compiled with dt_debug defined in order to compile in the necessary print calls.

Error Codes

The library may return the following error codes. The error codes are all less than zero, so they cannot be defined as part of the ASCII character set.

DT_ERROR	An operating system error occurred.
DT_TIMEOUT	An input operation did not complete in the required (operating system) time.
IO_ERROR	This is an error exit code for the exit() library routine. The value selected depends on the particular operating system.

DECtalk-Specific Parameters

Certain codes apply only to DECtalk (and not other devices, such as terminals). These codes are as follows.

CSI_DA_PRODUCT	The DECtalk product identification code.
DCS_F_DECTALK	The DECtalk specific device control sequence (DCS) final character.
P1_DECTALK	All DTC01-AA DCS sequences send this for their first (P1) parameter.
R1_DECTALK	All DTC01-AA DCS replies send this for the first (R1) reply parameter.

DECtalk Commands

The DECtalk commands that do not require specific parameters are coded as follows.

P2_PHOTEXT	Speak phonemic text.
P2_STOP	Stop speaking.
P2_SYNC	Synchronize.
P2_SPEAK	Enable/disable speaking.
P2_INDEX	Index text.
P2_IX_REPLY	Index with reply.
P2_IX_QUERY	Return last spoken index.
P2_DICT	Load user dictionary.
P2_PHONE	Telephone control (See "Telephone Control Parameters.")

P2_MODE	Synthesis mode control.
P2_LOG	Local terminal log control.
P2_TERMINAL	Local terminal control.

Telephone Control Parameters

The telephone control command P2_PHONE takes an additional parameter to specify the specific telephone action.

P3_PH_STATUS	Send a status report.
P3_PH_ANSWER	Answer on P4 rings.
P3_PH_HANGUP	Hang up the phone.
P3_PH_KEYPAD	Enable keypad data entry.
P3_PH_NOKEYPAD	Disable keypad data entry.
P3_PH_TIMEOUT	Send a timeout report if no data entered in P4 seconds if P4 is greater than zero; disable time-outs if P4 is zero.
P3_PH_TONE	Dial out using Touch-Tones.
P3_PH_PULSE	Dial out using pulses.

DECtalk Replies

Several P2_ commands return messages to the host.

R2_IX_REPLY	Reply to P2_IX_REPLY. R3 contains the last index processed.
R2_IX_QUERY	Reply to P2_IX_QUERY. R3 contains the last index processed.
R2_DICT	Reply to P2_DICT. R3 contains the dictionary entry status code.
R2_PHONE	Reply to P2_PHONE. R3 contains the telephone status.

DECtalk returns the following R3 parameters after a P2_PHONE command.

R3_PH_ONHOOK	Telephone is hung up (inactive).
R3_PH_OFFHOOK	Telephone is answered (active).
R3_PH_TIMEOUT	No data was entered by the telephone user within the required number of seconds.
R3_PH_TOOLONG	A telephone number to dial is too long.

DECtalk returns the following R3 parameters after a P2_DICT command.

R3_DI_LOADED	Dictionary entry was loaded.
R3_DI_NOROOM	The user dictionary is full.
R3_DI_TOOLONG	The dictionary entry is too long.

Self-Test Parameters

The following parameters control the DECtalk self-test (DECTST).

TEST_POWER	Rerun power-up test.
TEST_HDATA	Run host data link loopback test.
TEST_HCONTROL	Run host line control test.
TEST_LDATA	Run local line data test.
TEST_SPEAK	Speak a canned message.

The following status codes are returned by the extended DSR sequence.

DSR_OK	No errors detected.
DSR_COMFAIL	Communication failure.
DSR_INBUFOVER	Input buffer overflow.
DSR_DECNVRFAIL	Last restore from nonvolatile memory failed.
DSR_PHONEME	Incorrect phoneme entered.
DSR_PRIVATE	DECtalk DCS parameter error.
DSR_DECTSTFAIL	Last DECTST self-test failed.

Logging Command Parameters
The following parameters configure the P2_LOG command.

LOG_TEXT	Log spoken text.
LOG_PHONEME	Log generated phonemes.
LOG_RAWHOST	Log all characters received from host without change.
LOG_INHOST	Log all characters received from host in visible format.
LOG_OUTHOST	Log all output to host in visible format.
LOG_ERROR	Log error messages.
LOG_TRACE	Log commands in mnemonic form.

The following parameters are for the P2_TERMINAL command.

TERM_HOST	Send text entered from the local terminal to the host.
TERM_SPEAK	Speak text entered from the local terminal.
TERM_EDITED	Line-edit text entered from the local terminal.
TERM_HARD	Use hardcopy edit conventions.
TERM_SETUP	Speak setup dialog.
TERM_FILTER	Filter sequences sent to the local terminal.

The following parameters are for the P2_MODE command.

MODE_SQUARE	Accept [ ] bracket phonemic text.
MODE_ASKY	Use single-letter phonemic alphabet.
MODE_MINUS	Pronounce a hyphen (-) as "minus."

THE SEQUENCE DATA STRUCTURE

The C language uses a powerful form of information control called a data structure. Data structures closely resemble Pascal records and can pass and hold multiple pieces of information.

All information needed to generate and parse escape sequences is in the SEQUENCE data structure. SEQUENCE is configured by the following size constants.

SEQ_INTMAX	Maximum number of intermediate characters.
SEQ_PARMAX	Maximum number of parameters.

The SEQUENCE data structure contains the following components.

short state	Processing state or introducer character to send.
char final	Final character in sequence.
char private	Private introducer character (or X to indicate an error).
short param[]	Private parameters (unsigned); param[0] contains the number of parameters.
char inter[]	Intermediate characters; inter[0] contains the number of intermediates.

All information needed by the application program is in the DECTALK data structure which is created by dt_open() and freed by dt_close(). The DECtalk data structure is configured by the following parameters.

PEND_SIZE	Maximum number of keypad characters that may be typed ahead. Additional characters are discarded.
IN_BUFLEN	Size of the operating system input buffer.
OUT_BUFLEN	Size of the operating system output buffer.

The data buffer contains the following information.

DECTALK *link	Chains together all active units.
int unit	Operating system I/O channel.
short timeout	TRUE if timeouts enabled.
short pend_fc	Bytes in pending buffer.
short pend_fo	Index to free byte in pending buffer.
short pend_ep	Index to next byte to return from pending buffer.
char *in_ptr	Input buffer pointer.
char *in_end	Input buffer end.
char *out_ptr	Output buffer free pointer.
SEQUENCE send	Last DCS sequence sent.
SEQUENCE reply	Last DECtalk reply received.
SEQUENCE seq	Look-ahead for string terminator processing.
char *device	Remember dt_open() device name for debug printouts.
char pend[]	Type-ahead buffer.
char in_buff[]	Input buffer.
char out_buff[]	Output buffer.
struct sgtty stty_save	Terminal characteristics block (UNIX only).
FILE *fildes	File descriptor (RSX only).
struct iosb iosb	I/O status block (RSX only).
struct qioparm parm	QIO parameter block (RSX only).

APPLICATION PROGRAMS

The rest of this chapter lists the modules used to build the complete application program. All modules with the indicator comment

```
/*)LIBRARY
```

should be compiled and loaded into an object library. The main program, DEMO.C, is compiled and linked with the DECtalk library and the C standard library.

The modules appear in the following order (Table 6-1).

Table 6-1 Application Program Modules

Module	Brief Description
DECTLK.H	This file must be included in all modules that use the DECtalk application library. Contains common definitions (for example, escape sequence parameters).
DEMO.C	This is the main module of the program.
DTANSW.C	Hangs up the phone and answers on N rings.
DTCLOS.C	Closes the DECtalk channel and frees all buffers.
DTCMD.C	Sends a DCS command to the DECtalk terminal.
DTDCHA.C	Formats characters into a visible ASCII datastream format and writes the text to the indicated file.
DTDCS.C	Sends a DECtalk DCS control sequence.
DTDIAL.C	Dials the DECtalk telephone.
DTDRAIN.C	Absorbs any type-ahead characters.
DTDUMP.C	Writes an escape sequence buffer to the standard output file (for debugging).
DTEOL.C	Writes an end-of-line to DECtalk.
DTGESC.C	Reads an escape sequence or keypad character.

Table 8-1 Application Program Modules (Cont.)

Module	Brief Description
DTGET.C	Reads a character from the DECtalk terminal line.
DTHANG.C	Hangs up the telephone line connected to DECtalk.
DTINIT.C	Initializes the DECtalk terminal on the channel opened on it.
DTKEYB.C	Reads a telephone keypad button.
DTLOOK.C	Reads one character from the DECtalk terminal line.
DTOPUT.C	Either writes the output buffer contents to the DECtalk device or stores the character in a local buffer.
DTSAK.C	Indicates if the keyboard user typed any characters.
DTSELC	Tests the result of a dt_phone() message for keypad buttons.
DTSVA.C	Indicates if the argument character is one of 0123456789*#ABCD.
DTTEYP.C	Enables or disables the telephone keypad.
DTWRD.C	Sends a DECtalk DCR control sequence and reads a DCS reply.
DTOFFH.C	Tests the result of a dt_phone() message for OFFHOOK.
DTONHO.C	Tests the result of a dt_phone() message for ONHOOK.
DTOPEN.C	Initiates communications by performing operating system specific initializations.
DTPEEK.C	Tests if a character is pending from DECtalk.
DTPESC.C	Composes an appropriate escape sequence from the parameter buffer.
DTPHON.C	Sends a DECtalk phone message.
DTPTES.C	Tests a phone reply.

Table 6-1 Application Program Modules (Cont)

Module	Brief Description
DTPUT.C	Sends one character to the DECtalk terminal line. No value is returned.
DTREAD.C	Reads a sequence or character.
DTRESE.C	Sends a soft reset escape sequence.
DTSAVE.C	Saves user type ahead characters.
DTSPLICE.C	Lets you control a terminal connected to DECtalk's local port.
DTST.C	Sends a string terminator (for phoneme text and telephone dial commands) to DECtalk.
DTSYNC.C	Synchronizes DECtalk and the application.
DTTALK.C	Speaks one line of text.
DTTEST.C	Tests a DECtalk reply.
DTTIME.C	Enables or disables a telephone keypad timeout.
DTTONE.C	Sends the msg text string as a tone dialing sequence.
DTTRAP.C	Traps CTRL/C interrupts.
DTVISI.C	Generates visible ASCII character representations.
HELLO.C	Tests that DECtalk is operating correctly.

DECTLK.H

DECTLK.H must be included in all modules that use the DECtalk applications library. This file also defines common ASCII characters, DECtalk escape sequence parameters, library globals, and the DECtalk buffer structure. You can edit this file to enable debugging code defined by the DTDEBUG flag.

```
/*
 *      D e f i n i t i o n s   a n d   G l o b a l s
 *
 * This file contains symbolic definitions of the structures
 * and characters used by DECtalk application programs,
 * including all DECtalk escape sequence parameters.
 *
 * Note: on RSX-11M, your program must first #include <stdio.h>
 */

/*
 * Select a UNIX "flavor" (bizarre code as DECUS C lacks "defined()")
 */
#ifdef  unix
#ifndef BSD_42
#ifndef UNIX_V
#define UNIX_V
#endif
#endif
#endif

#:ifdef  DOCUMENTATION

title   dectlk.h        DECtalk Library Header File
index                   DECtalk library header file

synopsis

        #include "dectlk.h"

description

        This file is included in the compilation of all
        modules that use the DECtalk applications library.
        It defines common ASCII characters, DECtalk
        escape sequence parameters, library globals,
        and the DECTALK buffer structure.

configuration

        You can edit dectlk.h to enable debugging code
        by defining the DT_DEBUG flag as follow.

            #define DT_DEBUG 1

        This changes the primary input and output routines
        so that they become capable of logging all characters
        transmitted to and from the DECtalk device.
```

globals

The library provides two global flags which are used as follows.

dt_abort	This is set non-zero by an interrupted CTRL-C trap (if you have called dt_trap()). When set, no I/O will be performed, and library subroutines will exit as quickly as possible.
dt_debug	This may be set nonzero by an applications program to enable debug printouts. Note that the library must have been compiled with DT_DEBUG defined in order to compile in the necessary print calls.

error codes

The library may return the following error codes. These are all less than zero, and consequently cannot be part of the ASCII character set.

DT_ERROR	An operating-system error.
DT_TIMEOUT	An input operation did not complete in the required (operating-system) time.
IO_ERROR	An error exit code for the exit() library routine. The value is selected as appropriate for the particular operating system.

Routines implemented as macros

Certain frequently routines may be implemented as macros (if macro expansion is supported by the particular C compiler). These are as follows.

dt_iskey(dt)	TRUE if data is currently stored in the keypad type-ahead buffer.
dt_isvalid(c)	TRUE if the character is a valid keypad character. Note: evaluation of the argument must not have side-effects, i.e., you must not write dt_isvalid(*p++).
dt_ptext(dt,r3)	Phone test, TRUE if the current reply is R2_PHONE, R3.

```
        dt_offhook(dt)      Phone test, TRUE if the current
                            reply is R2_PHONE, R3_PH_OFFHOOK.

        dt_onhook(dt)       Phone test, TRUE if the current
                            reply is R2_PHONE, R3_PH_ONHOOK.

        dt_istimeout(dt)    Phone test, TRUE if the current
                            reply is R2_PHONE, R3_PH_TIMEOUT.

        dt_phone(dt,p3,p4) Send a phone message.

        dt_eol(dt)          Send "end of line" and force
                            output to DECtalk.

general definitions
    The following variables are defined.

        EOS                 End of string
        FALSE               For TRUE/FALSE testing
        TRUE                For TRUE/FALSE testing

ascii characters

    The following C0 control characters are defined.

      NUL   STX  ETX    BEL   BS    VT    SI

      LSO   XON  XOFF   CAN   SUB   ESC   DEL

    The following C1 control characters are defined.

      SS2   SS3   DCS  DLDID   CST    ST   OSC

      PM    APC   RDEL

    The following DECtalk-specific parameters are
    also defined.

      DSI_DA_PRODUCT    The DECtalk product
                        identification code.

      DCS_F_DECTALK     The DECtalk specific device
                        control sequence (DCS) final
                        character.

      P1_DECTALK        All DCTD1 DCS sequences
                        transmit this for their first
                        (P1) parameter.

      R1_DECTALK        All DCTD1 DCS replies transmit
                        this for the first R1 reply
                        parameter.
```

The P2 and P3 parameters select the specific DECtalk command.

P2_PHOTEXT	Speak phonemic text.
P2_STOP	Stop speaking.
P2_SYNC	Synchronize.
P2_SPEAK	Enable/disable speech.
P2_INDEX	Index text.
P2_IX_REPLY	Index with reply.
P2_IX_QUERY	Return last spoken index.
P2_DICT	Load user dictionary.
P2_PHONE	Telephone control.
P2_MODE	Synthesis mode control.
P2_LOG	Local terminal log control.
P2_TERMINAL	Local terminal control.
P2_MASK	Keypad mask control.

The telephone control command takes an additional parameter to specify the specific telephone action.

P3_PH_STATUS	Return a status report.
P3_PH_ANSWER	Answer on P4 rings.
P3_PH_HANGUP	Hangup the phone.
P3_PH_KEYPAD	Enable keypad data entry.
P3_PH_NOKEYPAD	Disable keypad data entry.
P3_PH_TIMEOUT	Send a timeout report if no data entered in P4 seconds if P4 is greater than zero; disable timeouts if P4 is zero.
P3_PH_TONE	Dial out using tones.
P3_PH_PULSE	Dial out using pulses.

Several P2 commands return messages to the host.

R2_IX_REPLY	Reply to P2_IX_REPLY. R3 contains the last index processed.
R2_IX_QUERY	Reply to P2_IX_QUERY. R3 contains the last index processed.
R2_DICT	Reply to P2_DICT. R3 contains the dictionary entry status code.
R2_PHONE	Reply to P2_PHONE. R3 contains the telephone status.

The following R3 parameters are returned after a P2_PHONE command.

R3_PH_ONHOOK	Telephone is hung up (inactive).
R3_PH_OFFHOOK	Telephone is answered (active).
R3_PH_TIMEOUT	No data was entered by the telephone user within the required number of seconds.
R3_PH_TOO_ONG	A telephone number to dial is too long.

```
The following R3 parameters are returned after a P2_DICT
command.

  R3_DI_LOADED      Dictionary entry was loaded.
  R3_DI_NOROOM      The user dictionary is full.
  R3_DI_TOOLONG     The dictionary entry is too long.

The following codes are used to control host-requested
self test (DECTST).

  TEST_POWER        Rerun power up test.
  TEST_HDATA        Host data link loopback test.
  TEST_HCONTROL     Host line control test.
  TEST_LDATA        Local line data test.
  TEST_SPEAK        Speak a canned message.

The following status codes are returned by the extended
DSR sequence.

  DSR_OK            No errors detected.
  DSR_COMFAIL       Communication failure.
  DSR_INBUFOVFLW    Input buffer overflow.
  DSR_DECNVRFAIL    Last restore from nonvolatile
                    memory failed.
  DSR_PHONEME       Incorrect phoneme entered.
  DSR_PRIVATE       DECtalk DCS parameter error.
  DSR_DECTSTFAIL    Last DECTST self-test failed.

The following flags configure the P2_LOG command.

  LOG_TEXT          Log spoken text.

  LOG_PHONEME       Log generated phonemes.

  LOG_RAWHOST       Log all characters received
                    from host without change.

  LOG_INHOST        Log all characters received
                    from host in "visible" format.

  LOG_OUTHOST       Log all output to host in
                    visible format.

  LOG_ERROR         Log error messages.

  LOG_TRACE         Log commands in mnemonic form.
```

The following flags are for the PS_TERMINAL command:

TERM_HOST	Send text entered from the local terminal to the host.
TERM_SPEAK	Speak text entered from the local terminal.
TERM_EDITED	Line-edit text entered from the local terminal.
TERM_HARD	Use hard-copy edit conventions.
TERM_SETUPSPEAK	Speak SETUP dialog.
TERM_FILTER	Filter escape sequences sent to the local terminal.

The following flags are for the PS_MODE command.

MODE_SQUARE	[ ] bracket phonemic text.
MODE_ASKY	Use single-letter phonemic alphabet.
MODE_MINUS	Pronounce '-' as "minus."

The following flags are for the dt_splice() function.

SPLICE_SPEAK	DECtalk speaks text if set.
SPLICE_LOG	Text sent to DECtalk is sent to the terminal (PS_LOG, LOG_RAWHOST).
SPLICE_TERM	The terminal may send text to DECtalk (PS_TERM, TERM_HOST).

Escape sequence data buffer

All information needed to generate and parse escape sequences is contained in the SEQUENCE data structure. It is configured by the following size constants.

SEQ_IMTMAX	Maximum number of intermediate characters.
SEQ_PARMAX	Maximum number of parameters.

It contains the following components.

short state	Processing state or introducer character to send.
char final	Final character in sequence.
char private	Private introducer character or 'X' to indicate an error.

short param[]	Private parameters (unsigned); param[0] contains the number of parameters.
char inter[]	Intermediate characters; inter[0] contains the number of intermediates.

DECTALK data buffer definition

All information needed by the DECtalk applications library is contained in the DECTALK data structure which is created by dt_open() and freed by dt_close(). It is configured by the following parameters.

PEND_SIZE	Maximum number of keypad characters that may be typed-ahead. Additional characters are discarded.
IN_BUFLEN	Size of the operating system input buffer.
OUT_BUFLEN	Size of the operating system output buffer.

The data buffer contains the following information.

DECTALK *link	Chains together all active units.
short unit	Operating system I/O channel.
short timeout	Current timeout value
short flag	Speech and dt_spkice flags.
short pend_fc	Bytes in pending buffer.
short pend_fp	Index to free byte in pending buffer.
short pend_ep	Index to next byte to return from pending buffer.
char *in_ptr	Input buffer pointer.
char *in_end	Input buffer end.
char *out_ptr	Output buffer free pointer.
SEQUENCE send	Last DCS sequence sent.
SEQUENCE reply	Last DECtalk reply received.
SEQUENCE seq	Look-ahead for string terminator processing.
char *device	Remember dt_open() device name for debug printouts.

```
        char pend[]         Type-ahead buffer.

        char in_buff[]      Input buffer.

        char out_buff[]     Output buffer.

        struct termio sity_save    Terminal characteristics
                                   block (UNIX System V).

        struct sgtty sity_save     Terminal characteristics
                                   block (UNIX 4.2 BSD).

        FILE *fildes        File descriptor (RSX).

        struct iosb iosb    I/O status block (RSX).

        QIOPARM parm        QIO parameter block (RSX).
                            (REX only).

        int*pos_xt          TRUE if PDS XK; driver
                            (REX only).

    The flag entry controls library internal states.

        _FLAG_SPEAK         Set if DECtalk is speaking.
        _FLAG_LOG           Set if LOG RAWHOST is set.
        _FLAG_TERM          Set if TERM HOST is set.
        _FLAG_EIGHTBIT      Set to read and write eight-bit
                            data and control sequences.

    FLAG_SPEAK, FLAG_LOG, and FLAG_TERM should not be changed
    by application programs.

    FLAG_EIGHTBIT must be set by the application program if
    DECtalk sends and receives C1 control sequences in their
    8-bit form.  Note that the application program must
    ensure that the operating system passes 8-bit data
    correctly and DECtalk setup must set HOST FORMAT to HOKE.

UNIX Notes
    On UNIX System V, the DECtalk terminal line is
    forced to 9600 Baud.  This may be changed to
    retain the current Baud rate.  Also, you should
    be aware that there are numerious subtle differences
    between operating systems.

Note

    UNIX and System V are trademarks of AT&T Bell Laboratories.

#endif

 /*
 * Define DT_DEBUG to enable debug printouts of transmitted
 * characters.
 */
```

```
#define DT_DEBUG

#define FALSE   0
#define TRUE    1
#ifndef EOS
#define EOS     '\0'
#endif

#ifdef  unix
#ifdef  BSD_42
#include        <sgtty.h>
#else
#ifdef  UNIX_V
#include        <termio.h>
#endif
#endif
#endif

/*
 * These error codes may not be in the ASCII range.
 */

#define DT_ERROR        (-1)
#define DT_TIMEOUT      (-2)

/*
 * C0 control characters
 */

#define NUL             0x00    /* NUL code                     */
#define STX             0x02    /* Start of text                */
#define ETX             0x03    /* End of text                  */
#define BEL             0x07    /* Bell                         */
#define BS              0x08    /* Backspace                    */
#define VT              0x0B    /* Vertical tab ('\013')        */
#define LS1             0x0E    /* LS1 (SO)                     */
#define LS0             0x0F    /* LS0 (SI)                     */
#define XON             0x11    /* DC1                          */
#define XOFF            0x13    /* DC3                          */
#define CAN             0x18    /* Cancel <CTRL/X>              */
#define SUB             0x1A    /* Substitute                   */
#define NUL             0x00    /* Null code                    */
#define ESC             0x1B    /* Escape                       */
#define DEL             0xFF    /* Delete                       */
/*
 * C1 control characters
 */

#define SS2             0x8E    /* Single shift 2               */
#define SS3             0x8F    /* Single shift 3               */
#define DCS             0x90    /* Device control sequence      */
#define OLDID           0x9A    /* ESC Z                        */
#define CSI             0x9B    /* Control Sequence Introducer  */
#define ST              0x9C    /* String terminator            */
#define OSC             0x9D    /* Operating System sequence    */
#define PM              0x9E    /* Privacy Message              */
#define APC             0x9F    /* Application Program Control  */
#define RDEL            0xFF    /* Delete in right side         */
```

```
#define CSI_DA_PRODUCT  19      /* Dectalk DA product code      */

/*
 * Basic definitions for DECtalk device control
 * strings. All DECtalk sequences have a first parameter of
 * P1_DECTALK. This provides an easy place for future DECtalk
 * products to fit into the scheme of things.
 */

#define DCS_F_DECTALK   'y'     /* DECtalk final                */
#define P1_DECTALK      0       /* DECtalk param 1              */
#define R1_DECTALK      0       /* DECtalk reply param 1        */

/*
 * The second parameter selects the basic command.
 */

#define P2_PHONEXT      5       /* Speak phonemic text          */
#define P2_STOP         10      /* Stop speaking                */
#define P2_SYNC         11      /* Synchronize                  */
#define P2_SPEAK        15      /* Enable or disable speaking   */
#define P2_INDEX        20      /* INDEX                        */
#define P2_IX_REPLY     21      /* INDEX_REPLY                  */
#define P2_IX_QUERY     22      /* INDEX_QUERY                  */
#define P2_DICT         40      /* Dictionary control           */
#define P2_PHONE        60      /* Phone control                */
#define P2_MODE         80      /* Synthesis mode control       */
#define P2_LOG          81      /* LOG information on local tty */
#define P2_TERMINAL     82      /* Local terminal control       */
#define P2_MASK         83      /* Set keypad mask              */

/*
 * Additional parameters for the phone command.
 */
#define P3_PH_STATUS    0       /* Send a status report         */
#define P3_PH_ANSWER    10      /* Answer (P4 has ring number)  */
#define P3_PH_HANGUP    11      /* Hangup                       */
#define P3_PH_KEYPAD    20      /* Raw keypad                   */
#define P3_PH_KDKEYPAD  21      /* Disable keypad               */
#define P3_PH_TIMEOUT   30      /* Status report on timeout     */
#define P3_PH_TONE      40      /* Dial out                     */
#define P3_PH_PULSE     41      /* Dial out                     */

/*
 * The second parameter in a reply specifies the general class
 * of the reply sequence.
 */

#define R2_IX_REPLY     31      /* Sent after INDEX_REPLY       */
#define R2_IX_QUERY     32      /* Sent after INDEX_QUERY       */
#define R2_DICT         50      /* Sent after DICT              */
#define R2_PHONE        70      /* Telephone status report      */
```

```
/*
 * Additional reply information is passed in the third parameter.
 */

#define R3_PH_ONHOOK     0        /* Hung up                          */
#define R3_PH_OFFHOOK    1        /* Phone is lifted                  */
#define R3_PH_TIMEOUT    2        /* No reply in 4 seconds            */
#define R3_PH_TOOLONG    3        /* Telephone # text too long        */
#define R3_DI_LOADED     0        /* Dictionary entry loaded ok       */
#define R3_DI_NOROOM     1        /* No room in dictionary            */
#define R3_DI_TOOLONG    2        /* String too long                  */

/*
 * Test specification codes for the request self test
 * (DECTST) sequence.
 */

#define TEST_POWER       1        /* Rerun power up tests             */
#define TEST_HDATA       2        /* Host line data loopback test     */
#define TEST_HCONTROL    3        /* Host line control test           */
#define TEST_LDATA       4        /* Local line data test             */
#define TEST_SPEAK       5        /* Speak a canned message           */

/*
 * Error (and success) codes for the extended DSR sequence.
 */

#define DSR_OK           20       /* All OK                           */
#define DSR_COMMFAIL     22       /* Communication failure            */
#define DSR_INBUFOVER    23       /* Input buffer overflow            */
#define DSR_DECNVRFAIL   24       /* Last DECNVR failed               */
#define DSR_PHONEME      25       /* Error in phonemic text           */
#define DSR_PRIVATE      26       /* Error in DECtalk private DCS     */
#define DSR_DECTSTFAIL   27       /* Last DECTST failed               */

/*
 * Local logging flags for the P2_LOG command.
 */

#define LOG_TEXT         0x0001   /* Log text that is spoken          */
#define LOG_PHONEME      0x0002   /* Log generated phonemes           */
#define LOG_RAWHOST      0x0004   /* Log raw host input               */
#define LOG_INHOST       0x0008   /* Log host input                   */
#define LOG_OUTHOST      0x0010   /* Log host output                  */
#define LOG_ERROR        0x0020   /* Log errors                       */
#define LOG_TRACE        0x0040   /* Log sequence trace info.         */

/*
 * Local terminal flags for the P2_TERMINAL command.
 */

#define TERM_HOST        0x0001   /* Send text to host                */
#define TERM_SPEAK       0x0002   /* Speak local terminal input       */
#define TERM_EDITED      0x0004   /* Edited                           */
#define TERM_HARD        0x0008   /* Local terminal is hardcopy       */
#define TERM_SETUPSPEAK  0x0010   /* Spoken setup mode                */
#define TERM_FILTER      0x0020   /* Filter logged esc. sequences     */
```

```
/*
 * Mode flags for the P8_MODE command.
 */

#define MODE_SQUARE      0x0001  /* [ ] are phonemic brackets     */
#define MODE_ASKY        0x0002  /* Use ASK^ alphabet             */
#define MODE_MINUS       0x0004  /* "-" is pronounced "minus"     */

/*
 * Flags for dt_splice() and ((DECTALK *)dt)->flag
 */

#define SPLICE_SPEAK     0x0001  /* Speak text if set             */
#define SPLICE_LOG       0x0002  /* Log newhost if set            */
#define SPLICE_TERM      0x0004  /* Local host if set             */
#define _FLAG_SPEAK      0x0001  /* Speaking, set by dt_splice()  */
#define _FLAG_LOG        0x0002  /* Log newhost from dt_splice()  */
#define _FLAG_TERM       0x0004  /* Term host from dt_splice()    */
#define _FLAG_EIGHTBIT   0x0008  /* Read eight-bit C1 controls    */

/*
 * These macros and structure definitions are used by the escape
 * sequence parser.
 */

#define SEQ_INTMAX       2       /* Max. # of intermediates       */
#define SEQ_PARMAX       16      /* Max. # of parameters          */
/*
 * dt_getesc() (get escape sequence) and dt_sesc() (put escape
 * sequence) use this structure for all processing.
 */

typedef struct  {
    short       state;           /* Processing state or intro     */
    char        final;           /* Final character in seq.       */
    char        privates;        /* Private introducer            */
#ifdef  decus
    unsigned            param[SEQ_PARMAX+1];
#else
    unsigned short      param[SEQ_PARMAX+1];
#endif
                                 /* Intermediate count, values    */
    char        inter[SEQ_INTMAX+1];
} SEQUENCE;

/*
 * The DECTALK structure is used to maintain all information
 * needed to program a DECtalk device.  It is allocated by
 * dt_open(), freed by dt_close() and a required parameter
 * by essentially all routines.
 */
```

```
#ifdef  rsx
/*
 * The qio parameter block controls all RSX11-M I/O requests.
 */
typedef struct qioparm {            /* QIO parameter block  */
    char        *buffer;            /* Buffer location      */
    int         size;               /* Bytes to transfer    */
    char        *p3;                /* For ctrl/c ast       */
    char        *table;             /* Terminator table     */
    int         unused[2];          /* Not used here        */
} QIOPARM;

/*
 * The I/O status block receives the status of all I/O requests.
 */
typedef struct iosb {               /* I/O status block             */
    char        status;             /* Operation status             */
    char        terminator;         /* Input terminator byte        */
    int         count;              /* Bytes read from device       */
} IOSB;
#endif

#ifndef PEND_SIZE
#define PEND_SIZE       32              /* Pending buffer size  */
#endif
#ifndef IN_BUFLEN
#define IN_BUFLEN       32
#endif
#ifndef OUT_BUFLEN
#define OUT_BUFLEN      128
#endif
#if     (IN_BUFLEN < 1 || OUT_BUFLEN < 1 || PEND_SIZE < 0)
        << error, mandatory parameters aren't correct >>
#endif

typedef struct DECtalk {
    struct DECtalk *link;           /* Chain all units together     */
    short       unit;               /* I/O channel                  */
    short       timeout;            /* For dt_timeout()             */
    short       flag;               /* Speech and "splice" flags    */
    short       pend_fc;            /* Bytes in pending buffer      */
    short       pend_fp;            /* Pending buffer fill index    */
    short       pend_ep;            /* Pending buffer empty index   */
    char        *in_ptr;            /* I/O input buffer pointer     */
    char        *in_end;            /* -> end of input buffer       */
    char        *out_ptr;           /* -> free spot in output buff. */
    SEQUENCE    send;               /* Last sequence sent           */
    SEQUENCE    reply;              /* Last sequence read           */
    SEQUENCE    seq;                /* Sequence look-ahead          */
    char        *device;            /* DECtalk hardware device      */
    char        pend[PEND_SIZE]; /* Type-ahead ring buffer       */
    char        in_buff[IN_BUFLEN];     /* I/O input buffer     */
    char        out_buff[OUT_BUFLEN];   /* I/O output buffer    */
```

```
    /*
     * The following entries are operating-system specific.
     */
#ifdef  unix
#ifdef  BSD_42
    struct sgttyb stty_save;    /* Terminal flags               */
#else
#ifdef  UNIX_V
    struct termio stty_save;    /* Terminal flags (UNIX V7)     */
#endif
#endif
#endif
#ifdef  vax
    FILE        *fildes;        /* File descriptor              */
    IOSB        iosb;           /* I/O status block             */
    QIOP4RN     parm;           /* QIO parameter block          */
    short       pos_xt;         /* Device characteristics word  */
#endif
} DECTALK;

/*
 * Certain short routines and common tests are expressed as
 * macros.  In all instances, 'dd' is a DECtalk I/O descriptor
 * as returned by dt_open().  Note that the arguments should
 * not have "side-effects".
 *
 *   dt_iskey(dd)       TRUE if something in type-ahead buffer.
 *   dt_isvalid(c)      TRUE if argument is a valid keypad key.
 *
 * The following are only useful after executing dt_phone().
 *
 *   dt_ptest(dd, r3)   TRUE if specific phone reply.
 *   dt_offhook(dd)     TRUE if last DECtalk reply is OFFHOOK.
 *   dt_onhook(dd)      TRUE if last DECtalk reply is ONHOOK.
 *   dt_listimeout(dd)  TRUE if last DECtalk reply is TIMEOUT.
 *
 * The following simple commands may be written as macros:
 *
 *   dt_phone(dd,p3,p4) Send a phone message.
 *   dt_get(dd, sec)    Read a character (with timeout)
 *   dt_put(dd, c)      Send a character to DECtalk
 *   dt_eol(dd, c)      Send "end of line", flush output buffers
 *
 * If DT_DEBUG is #defined, dt_get() and dt_put() are functions
 * which may log all characters to the standard output stream.
 */
```

```
#ifndef DT_DEBUG
#define dt_get          dt_loget
#define dt_put          dt_loput
#endif
#ifndef nomacarg
#define dt_iskey(dd)    (dd->pend.fc != 0)
#define dt_isvalid(c)   ((  (c >= '0' && c <= '9')      \
                         || c == '#' || c == '*'        \
                         || (c >= 'A' && c <= 'D'))
#define dt_ptest(dd,r3) (dt_test(dd, R2_PHONE, r3))
#define dt_offhook(dd)  (dt_ptest(dd, R3_PH_OFFHOOK))
#define dt_onhook(dd)   (dt_ptest(dd, R3_PH_ONHOOK))
#define dt_timeout(dd) (dt_ptest(dd, R3_PH_TIMEOUT))
#define dt_phone(dd,p3,p4) (dt_msg(dd, \
            P2_PHONE, p3, p4, R2_PHONE, -1))
#ifdef  unix
#define dt_eol(dd)      (dt_put(dd, '\n'), dt_put(dd, 0))
#else
#define dt_eol(dd)      (dt_put(dd, '\r'), \
            dt_put(dd, '\n'), dt_put(dd, 0))
#endif
#endif
#ifdef  decus
#ifdef  DT_DEBUG
/*
 * This forces traceback on Decus C systems.
 */
#define exit            error
#define IO_ERROR        "fatal DECtalk I/O error"
#endif
#endif

#ifndef IO_ERROR
#ifdef  vms
#include                <ssdef.h>
#define IO_ERROR        SS$_ABORT
#else
#define IO_ERROR        2
#endif
#endif
/*
 * dt_abort may be set by a user program at any time to
 * stop DECtalk.  Typically, it would be set by dt_trap()
 * when a <CNTL/C> (UNIX INTERRUPT signal) is typed by the
 * terminal user.
 */
extern int      dt_abort;               /* Set TRUE to stop     */
extern DECTALK  *dt_root;               /* Root of device chain */
#ifdef  DT_DEBUG
extern int      dt_debug;               /* TRUE if debug log    */
#endif
```

DEMO.C

The executable program's name is DEMO, derived from this program. DEMO.C is the main module of the program.

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

main(argc, argv)
int             argc;
char            *argv[];
{
        register DECTALK        *dt;    /* Dectalk device       */
        register int    retries;        /* Initializations     */
        register int    ncalls;         /* Completed calls     */
        char            *dev;
        extern DECTALK  *dt_open();

        dev = "TT2:";
        if (argc > 1)
            dev = argv[1];
        retries = 0;
        ncalls = 0;
        dt_debug = TRUE;
        if ((dt = dt_open(dev)) == NULL) {
            perror(dev);
            return;
        }
        dt_trap();                      /* Catch CTRL-C abort  */
        while (dt_init(dt)) {           /* One-time setup      */
            dt_dcs(dt, PS_MODE, NODE_SQUARE, -1);
            retries++;                  /* Count attempts      */
            while (dt_answer(dt, 1)) {  /* Answer the phone    */
                if (process(dt)) {      /* Do user process     */
                    ncalls++;           /* User ran ok         */
                    retries = 0;        /* Clear retry count   */
                }
                dt_hangup(dt);          /* Hangup the phone    */
                if (dt_abort()          /* Check interrupt     */
                    goto finis;         /* Error exit          */
            }
            if (dt_abort)
                goto finis;             /* Error exit          */
            if (retries > 2) {          /* Got lost?           */
                printf("Too many retries\n");
                break;
            }
        }
        fprintf(stderr, "Couldn't initialize DECtalk\n");
```

```
finish:  dt_abort = FALSE;                  /* Restart output       */
         dt_reset(dt);                      /* Hangup DECtalk       */
         dt_put(dt, 0);                     /* Force out buffer     */
         dt_close(dt);                      /* Close up DECtalk     */
}
process(dt)
register DECTALK        *dt;
{
         register char  c;                  /* Keypad character     */
         char           work[30];           /* For echo message     */

         dt_talk(dt, "Welcome to DECtalk");
         /* if(!dt_keypad(dt, TRUE))        /* Enable keypad        */
             return (FALSE);                /* Error occurred       */
         for (;;) {                         /* Do forever...        */
             c = dt_inkey(dt, 15);          /* Key with timeout     */
             if (!dt_isvalid(c))            /* Check for timeout    */
                 break;                     /* Exit if so           */
             sprintf(work, "You pressed %c", c);
             dt_talk(dt, work);
             if (c == '*') {                /* Make '*' special     */
                 dt_timeout(dt, 0);         /* No timeouts now      */
                 dt_talk(dt, "Long message...");
             }
         }
         /*
          *Timeout is normal, others are errors.
          */
         return ((c == '?') ? TRUE : FALSE);
}
```

DTANSW.C

This routine hangs up the phone and answers on n rings.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_answer       Answer the Telephone
index                   Answer the telephone

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"
        int
        dt_answer(dt, nrings)
        DECTALK         *dt;    /* Device descriptor    */
        int             nrings; /* Number of rings      */

description

        Hang up the phone (by calling dt_hangup()) and
        answer the phone after the specified number
        of rings.

        Return TRUE if successful, FALSE if in error.
#endif

#include        <stdio.h>
#include        "dectlk.h"

int
dt_answer(dt, nrings)
register DECTALK        *dt;
int                     nrings;
/*
 * Hang up the phone and answer on nrings.
 */
{
        register int    code;

again:  if (!dt_hangup(dt))             /* Make sure it's       */
            return (FALSE);             /* on-hook.             */
        dt_idle(dt, P2_PHONE, P3_PH_ANSWER, nrings);
        while (dt_read(dt, 0), dt_onhook(dt)) {
            if (dt_abort)
                return (FALSE);
        }
        if (dt_onhook(dt))
            goto again;
        if (!dt_offhook(dt))            /* Did it answer ok?    */
            return (FALSE);
        /*
         * OK, clear timeout flag and type-ahead counters.
         */
        dt->timeout = 0;
        dt->pend_fc = dt->pend_fp = dt->pend_ep = 0;
        return (TRUE);
}
```

DTCLOS.C

This routine closes the DECtalk channel and frees all buffers.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_close        Terminate DECtalk Operation
:ndex                   Terminate DECtalk Operation

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_close(dt)
        DECTALK         *dt;    /* DECtalk device       */

description

        Close the DECtalk channel and free all buffers.
        No error is returned.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  vax
#include        <ex.h>
#include        <qiofun.h>
#include        <qioret.h>
#include        <qiottd.h>
#define QIO_EFN 1

static QIOPARM  noparm;                 /* QIO parm (all zero)  */
#endif


dt_close(dt)
register DECTALK        *dt;
/*
 * Close the DECtalk channel.
 */
{
        register DECTALK        **linep;
#ifdef  unix
#ifdef  BSD_42
        stty(dt->unit, &dt->stty_save);         /* Restore tty flags    */
#else
#ifdef  UNIX_V
        ioctl(dt->unit, TCSETA, &dt->stty_save); /* Restore tty flags   */
#endif
```

```
#endif
        close(dt->unit);
#endif
#ifdef  vms
        sys$dassgn(dt->unit);
#endif
#ifdef  rsx
        rs_close(dt->unit);
#endif
#ifdef  rsx
        qiow(IO_DET, dt->unit, 0|0_EFN, NULL, NULL, &noparm);
        fclose(dt->fildes);
#endif
        /*
         * Unlink the device from the chain.
         */
        for (linkp = &dt_root; *linkp != NULL;
                linkp = &((*linkp)->link)) {
            if (*linkp == dt) {
                *linkp = dt->link;
                break;
            }
        }
        free(dt->device);
        free((char *)dt);
        return (NULL);
}
```

DTCMD.C

This routine sends a DCS command to the DECtalk terminal.

```
/*ILIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_cmd          Send DCS w/o String Terminator
index                   Send DCS w/o string terminator

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_cmd(dt, p2, p3)
        DECTALK         *dt;    /* Device descriptor    */
        int             p2;     /* P2_... parameter     */
        int             p3;     /* P3_... parameter     */

description

        This routine sends a DCS command to the DECtalk
        terminal.  The string terminator is not sent.
        This is needed to send phonemic text or telephone
        dial commands.

        The p2 or p3 parameter may be -1 if it is to be
        omitted.

        A phonemic text sequence would be sent as follows.

            dt_cmd(dt, p2, p3);
            dt_talk(dt, "hh'ehlow.");
            dt_sr(dt);

#endif

#include        <stdio.h>
#include        "dectlk.h"

static SEQUENCE command = {
    DCS, DCS_F_DECTALK, 0, { 3, P1_DECTALK, 0, 0 }
};

dt_cmd(dt, p2, p3)
register DECTALK        *dt;            /* Device descriptor    */
int                     p2;             /* P2_command or -1     */
int                     p3;             /* P3_command or -1     */
```

```
/*
 * Send a DCS command, no string terminator
 */
{
        if (p2 == -1)
            command.param[0] = 1;
        else {
            command.param[2] = p2;
            if (p3 == -1)
                command.param[0] = 2;
            else {
                command.param[0] = 3;
                command.param[3] = p3;
            }
        }
        di_send(di, &command);
}
```

DTDCHA.C

This routine formats characters into a visible ASCII datascope format and writes the resulting text to the indicated file. Note that this routine is independent of DECtalk definitions. Output is via the C standard library. Dumps to terminals are unbuffered.

```
/*DLIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_dchar        Dump One Character Visibly
index   :               Dump one character visibly

synopsis

        #include        <stdio.h>

        dt_dchar(c, iov)
        int             c;      /* Character to dump    */
        FILE            *iov;   /* File to write to     */

description

        The character is formatted into a visible ASCII
        Datascope format and the resulting text written
        to the indicated file.

        Note that this routine is independent of DECtalk
        definitions.

        Output is via the C standard library.  If the dump
        is to a terminal, it is unbuffered.

#endif
#include        <stdio.h>

dt_dchar(c, iov)
register int            c;
register FILE           *iov;
/*
 * Dump a character.
 */
{
        char            work[12];

        dt_visible(c, work);
        fprintf(iov, "%s", work);
        if (!isatty(fileno(iov)))
            fflush(iov);
}
```

DTDCS.C

This routine sends a DECtalk DCS control sequence using the p2, p3, and p4 parameters. Pn parameters are -1 if not sent. No errors are possible.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_dcs          Send a DECtalk DCS Command
index                   Send a DECtalk DCS command

Synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_dcs(dt, p2, p3, p4)
        DECTALK         *dt;    /* Device descriptor    */
        int             p2;     /* P2_xxx parameter     */
        int             p3;     /* P3_PH_xxxx parameter */
        int             p4;     /* timeout or rings     */

description

        This routine sends a DECtalk DCS control sequence
        using the p2, p3, and p4 parameters.

        Note that the Pn parameters are -1 if they
        are not sent.

        No errors are possible.
#endif

#include        <stdio.h>
#include        "dectlk.h"

static SEQUENCE DT_string.terminator = {
    ST                                  /* String terminator    */
};

dt_dcs(dt, p2, p3, p4)
register DECTALK        *dt;    /* Dectalk device               */
int             p2, p3, p4;     /* Parameters to send           */
```

```
/*
 * Load the parameter buffer and send the sequence.
 * dt->send.param[0] contains the number of additional parameters.
 */
{
        dt->send.state = DCS;
        dt->send.final = DCS_F_DECTALK;
        dt->send.private = 0;
        dt->send.inter[0] = 0;
        dt->send.param[0] = 1;
        if (p2 >= 0) {
            dt->send.param[0]++;
            dt->send.param[2] = p2;
        }
        if (p3 >= 0) {
            dt->send.param[0]++;
            dt->send.param[3] = p3;
        }
        if (p4 >= 0) {
            dt->send.param[0]++;
            dt->send.param[4] = p4;
        }
        dt_pesc(dt, &dt->send);
        dt_pesc(dt, &DT_string_terminator);
}
```

DTDIAL.C

This routine dials the DECtalk telephone, depending on whether the telephone used is Touch-Tone or pulse type.

```
/*DLIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_dial         Dial the telephone
index                   Dial the telephone

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_dial(dt, p3, numb, wait, msg)
        DECTALK         *dt;    /* Device descriptor    */
        int             p3;     /* P3_PH_xxxx parameter */
        char            *numb;  /* Number to dial       */
        int             wait;   /* See below            */
        char            *msg;   /* Announcement         */

description

        This routine dials the DECtalk telephone.  The P3
        parameter must be either P3_PH_TONE (tone dial)
        or P3_PH_PULSE (pulse dial).

        For tone dialing, the number text may contain any
        valid touch-tone characters ("0123456789*#ABCD")
        or the characters 'T' (for a one second delay)
        or the 'F' for a 250 millisecond switch-hook flash.
        All other characters are ignored.

        If pulse dialing is selected, only the digits, 'T' and
        'F' are interpreted.

        Note that the telephone will not be hung up before
        dialing if it is offhook when the command is issued.
```

```
Call Progress Detection

        DECtalk cannot tell if or when someone answers the
        phone.  The only way to do this is to speak a message,
        such as "This is DECtalk, please press any button
        on the keypad." and wait some limited time for the
        person to press the button.  The wait and msg
        parameters provide this capability.

        If wait is less than or equal to zero, DECtalk returns
        without attempting to verify that someone has answered
        the phone.  The return will be TRUE if the phone is
        offhook.

        If wait is greater than zero, it specifies the number
        of seconds to wait for a response, and msg is the
        message to speak.  (If msg is NULL, the sample text
        shown above will be used.)  The message is repeated
        continuously until either the alloted time has
        elapsed or a button is received.  dt_dial() then returns
        TRUE if the phone is offhook, as above.

        To cause DECtalk to silently wait for a message, use
        a zero-length string ("").  Note, however, that an
        audible message is required by some public telephone
        systems.

        When DECtalk returns after call progress detection,
        keypad data entry and keypad timeout will be disabled.

#endif

#include    <stdio.h>
#include    "dectlk.h"

#define ANNOUNCEMENT    "This is DECtalk, please press any key."

int
dt_dialf(dt, p3, number, wait, message)
register DECTALK        *dt;            /* Device descriptor    */
int                     p3;             /* P3_PH_PULSE or TONE  */
register char           *number;        /* Number to dial       */
int                     wait;           /* Call progress delay  */
char                    *message;       /* Announcement         */
```

```
/*
 * Send a phone message.
 */
{
        register int    code;
        int             dialtime;       /* Time to dial phone   */
        long            endtime;
        extern long     time();

        if (number == NULL)             /* Paranoia,            */
            number = "";                /* Ahh, paranoia        */
        dt_cmd(dt, P2_PHONE, p3);
        dialtime = strlen(number);
        if (p3 == P3_PH_PULSE)
            dialtime *= 2;
        while (*number != EOS)          /* Send the number      */
            dt_put(dt, *number++);
        dt_wt(dt);
        do {
            code = dt_read(dt, dialtime + 30);
        } while (code == ST || dt_save(dt, code));
        if (wait <= 0)
            return (dt_offhook(dt));
        /*
         * Call progress detection.
         */
        if (!dt_offhook(dt) || !dt_keypad(dt, TRUE))
            return (FALSE);
        endtime = time(NULL) + wait + 1;
        if (message == NULL)
            message = ANNOUNCEMENT;
        do {
            dt_tell(dt, message);       /* Speak announcement   */
            dt_put(dt, VT);             /* Make sure it's heard */
        } while ((code = dt_read(dt, 5)) < 0
              && time(NULL) <= endtime);
        dt_dcs(dt, P2_STOP, -1, -1);    /* Enough already       */
        if (dt_isvalid(code)) {         /* User key?            */
            dt_keypad(dt, FALSE);       /* Turn off keypad and  */
            dt_drain(dt);               /* Drop pending text    */
            return (TRUE);              /* Normal return        */
        }
        else if (dt_phone(dt, -1, -1), dt_offhook(dt))
            dt_hangup(dt);              /* No response, hangup  */
        return (FALSE);
}
```

DTDRAI.C

This routine absorbs any type-ahead characters. No errors are possible.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_drain        Drain Pending Input
index                   Drain pending input

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_drain(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        Absorb any type-ahead characters.
        No errors are possible.

note

        On UNIX systems, dt_drain() will also cancel pending
        output.  This may cause DECtalk to receive word
        fragments or partial escape sequences.

        The code is conditionally compiled for two varieties
        of UNIX: Ultrix-32 (or 4.2 bsd) and UNIX System V.
        Other varieties of UNIX and UNIX-like systems may
        need to edit this file.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  unix

dt_drain(dt)
register DECTALK        *dt;
/*
 * dt_drain() tosses out any pending type-ahead.
 */
{
        dt->pend_fc = dt->pend_fp = dt->pend_ep = 0;
#ifdef  BSD_42
        ioctl(dt->unit, TIOCFLUSH, NULL);
#else
#ifdef  UNIX_V
        ioctl(dt->unit, TCFLSH, 0);             /* UNIX V7              */
#endif
#endif
        dt->in_ptr = dt->in_end = dt->in_buff;
}
#endif
```

```
#ifdef  vms
#include        <iodef.h>
dt_drain(dt)
register DECTALK        *dt;
/*
 * dt_drain() tosses out any pending type-ahead.
 */
{
        dt->pend_fp = dt->pend_ep = 0;
        /*
         * This is probably sub-optimal.  It should be possible
         * to do "sys$qiow..."
         *      IO$_READLBLK | IO$M_PURGE | IO$M_TIMED
         * with a zero-length timeout, but I sure don't know.
         */
        while (dt_vmsread(dt,
                IO$_READLBLK | IO$M_NOECHO | IO$M_NOFILTR | IO$M_TIMED,
                IN_BUFLEN, 0) >= IN_BUFLEN)
            ;
        dt->in_ptr = dt->in_end = dt->in_buff;
}
#endif

#ifdef  rt11
#include        <rsts.h>

dt_drain(dt)
register DECTALK        *dt;
/*
 * dt_drain() tosses out any pending type-ahead.
 */
{
        dt->pend_fp = dt->pend_ep = 0;
        clrxrb(0);
        xrb.xrlen = 7;                  /* Cancel type-ahead    */
        xrb.xrad = dt->unit * 2;
        xrb.xrblkm = TTYHND;
        rutsys(.SPEC);
        dt->in_ptr = dt->in_end = dt->in_buff;
}
#endif

#ifdef  rsx
dt_drain(dt)
register DECTALK        *dt;
/*
 * dt_drain() tosses out any pending type-ahead.
 */
{
        dt->pend_fp = dt->pend_fp = dt->pend_ep = 0;
        do {
            dt->in_ptr = dt->in_end = dt->in_buff;
        } while (dt_get(dt, 1) > 0);
        dt->in_ptr = dt->in_end = dt->in_buff;
}
#endif
```

DTDUMP.C

This routine writes an escape sequence buffer to the standard output file. This mode is for debugging.

```
/*>LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_dump         Dump Escape Sequence Buffer
index                   Dump escape sequence buffer

synopsis

        #include        <stdio.h>
        #include        "dectk.h"

        int
        dt_dump(what, seq)
        char            *what;  /* Explanation          */
        SEQUENCE        *seq;   /* Buffer to dump       */

description

        The requested escape sequence buffer is written
        (visibly) to the standard output file.

        If what is not NULL, it is written as an identifier.

        Output is via the C standard library.

        For example,

            #include <stdio.h>
            #include "dectlk.h"

            DECTALK *dt;
            extern DECTALK *dt_open();
            ...
                /*
                 * Open a DECtalk device,
                 * request phone status and
                 * dump returned status sequence.
                 */
                dt = dt_open("tb2;");
                dt_phone(dt, P2_PH_STATUS, -1);
                dt_dump("status", dt->reply);

#endif

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

```
dt_dump(what, seq)
char                    *what;
register SEQUENCE       *seq;
{
        register int            i;
        register char           *wp;
        char                    work[81];
        extern char             *dt_visible();

        if (what != NULL)
            printf("%s: \"", what);
        wp = dt_visible(seq->state, work);
        switch (seq->state) {
        case ESC:
        case CSI:
        case DCS:
            if (seq->private != 0)
                wp = dt_visible(seq->private, wp);
            for (i = 1; i <= seq->param[0]; i++) {
                if (i > 1)
                    *wp++ = ';';
                if (seq->param[i] != 0) {
                    sprintf(wp, "%u", seq->param[i]);
                    wp += strlen(wp);
                }
            }
            for (i = 0; i <= seq->inter[0]; i++)
                wp = dt_visible(seq->inter[i], wp);
            break;

        default:
            break;
        }
        if (seq->final != 0)
            wp = dt_visible(seq->final, wp);
        *wp = EOS;
        printf("%s%s", work, (what == NULL) ? "" : "\"\n");
}
```

DTEOL.C

This routine writes an end of line to DECtalk and calls the operating system executive service to write the local output buffer to the terminal. No value is returned.

You need this routine on operating systems that enforce line wraparound on the terminal. DTEOL.C also improves the appearance of the debugging logs.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_eol          Write End of Line to DECtalk
index                   Write End of Line to DECtalk

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_eol(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        An "end of line" is written to DECtalk and the
        operating system executive service is called to
        cause the local output buffer to be written to
        the terminal.

        No value is returned.

        This routine is needed on operating systems that
        enforce "line wrap-around" on terminal devices.
        It also improves the appearance of debugging logs.
#endif

#include        <stdio.h>
#include        "deotlk.h"

#ifdef  dt_eol
#undef  dt_eol
#endif

dt_eol(dt)
register DECTALK        *dt;    /* Device descriptor            */
{
#ifndef unix
        dt_put(dt, '\r');
#endif
        dt_put(dt, '\n');
        dt_put(dt, 0);
}
```

DTGESC.C

This routine reads an escape sequence or keypad character.

```
/*DLIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_gesc         Read Escape Sequence or Character
index                   Read escape sequence or character

synopsis

        #include        <stdio.h>
        #include        "desill.h"

        int
        dt_gesc(dt, sec)
        DECTALK         *dt;    /* Device descriptor    */
        char            sec;    /* 0.5, timeout value   */

description

        Read an escape sequence or keypad character.

        dt_gesc() interprets a stream of 7- or 8-bit characters
        including escape sequences adhering to the coded representations
        of ISO 646, ISO 2022, and ISO 6429 with extensions to the DCS
        introducer as required by DEC Standard 138.

        The function dt_gesc() recognizes ESC, CSI, and DCS,
        and processes characters following each of these introducers
        until a complete sequence is encountered.  In the case of DCS,
        control returns to the caller after the final character of
        the DEC Standard 138 introduction sequence, but before the
        first data character of the device control string.

        When sandwiched between the application and a get character
        function (dt_get()), dt_gesc() transforms the input
        stream from a character stream to a stream of tokens consisting
        of characters, escape sequences, control sequences, and DCS
        introduction sequences.  When any of the recognized sequence
        types is encountered, the function value returned is that of
        ESC, CSI, or DCS, and the interpreted body of the sequence
        is returned in the seq structure. The caller may treat
        dt_gesc() similarly to getchar(), ignoring the returned
        structure in all cases except when the returned function value
        is ESC, CSI, or DCS.
```

```
An additional function performed by dtlgesc() is that all
C1 control functions received in their 7-bit form are returned
to the caller in their 8-bit form, thus eliminating the need
for the caller to process C1 control functions in their (7-bit)
escape sequence form and enforcing the equivalence of the 7-bit
and 8-bit forms of the C1 control functions.  The function
also enforces the sequence cancellation effect of the SUB and
CAN control characters.

The dt_gesc() function calls the user-supplied dt_get()
(read one character) function as many times as required to
complete an escape sequence, control sequence, or Digital
standard DCS introduction sequence. In the passed data
structure, it returns the final character, intermediate
characters, and parameter values.

Since 7-bit operation is a compatible subset of 8-bit
operation, there is -- normally -- no distinction in the
dt_gesc() function between the two environments.
The application program may set the _FLAG_EIGHTBIT bit
in dt->flag to receive C1 control characters in their
eight-bit form.  If _FLAG_EIGHTBIT is set on, the
application program must also ensure that the host
operating system communication line receives eight
data bits, and that DECtalk setup has set HOST FORMAT EIGHT.

Also, dt_get() may return two special values,
DT_ERR and DT_TIMEOUT, to indicate operating-system errors
and communication line timeouts respectively.

Because C0 control characters may be embedded in sequences,
and must be interpreted as if they occurred before the
sequence in the stream, the dt_gesc() function
retains internal state information in the sequence data
structure from call to call.  The seq.state value is zero
on return to indicate a complete escape sequence.  If non-zero,
it contains the sequence introducer.

If the "seq.state" element is zero, dt_gesc() assumes
that the remainder of the data structure is invalid and that
there is no data being returned from a prior call.  A non-zero
value for the "seq.state" element indicates a particular
internal state (ESC, CSI, or DCS) that the parser should assume
on the next call.
Intermediate characters and parameter values interpreted up
to the occurrence of the embedded control character are also
stored in the returned data structure and also should not be
altered by the caller.

Escape sequence syntax errors are indicated by setting the
seq.private parameter to 'X' (which is not a possible
private parameter).
```

```
        If the dt_gseq() function encounters more than the
        allowed maximum number of intermediate characters, the
        returned data structure indicates that one more intermediate
        character was received than allowed. Of course, characters
        after the maximum are not stored.

        If the dt_gseq() function encounters more than the
        allowed maximum number of parameters, the extra parameters
        are ignored and the returned data structure indicates that
        the allowed maximum number of parameters was received.

        After each call to dt_gseqf() the dt->seq SEQUENCE
        contains the following information.

          seq.state     Zero to indicate complete sequence

          seq.final     The sequence final character

          seq.private   Private parameter character;
                        EOS, <, =, >, ?, or X for errors

          seq.param[0]  The number of parameter values
                        (0:SEQ_PARMAX)

          seq.param[n]  (unsigned) The n'th parameter value

          seq.inter[0]  The number of intermediate characters
                        (0:SEQ_INTMAX+1)

          seq.inter[n]  (char) The n'th intermediate character.

        In general, the intermediate and final characters should be
        taken as a whole to determine the action.  It is easy to
        ignore sequences with too many intermediate characters since
        the returned number of intermediate characters will not match
        any action function.

        To simplify the code, this module doesn't wait for overly
        large parameter values and assumes that all overflow errors
        are due to invalid escape sequences.

*/

#include        <stdio.h>
#include        "dectlk.h"

int
dt_gseqf(dt, sec)
register DECTALK        *dt;    /* Dectalk device               */
int                     sec;    /* O.S. timeout                 */
```

```
/*
 * Return a character or sequence
 */
{
        register int    c;
#ifdef  decus
        register unsigned       *p;
#else
        register unsigned short *p;
#endif

#ifdef  DT_DEBUG
        if (dt_debug) {
            dt_put(dt, 0);
            printf("get: \"");
            if (!isatty(fileno(stdout)))
                fflush(stdout);
        }
#endif
        for (;;) {
            /*
             * Loop until end of sequence forces an exit.
             * Get the next character from the input stream.
             * Note: we assume that negative values are
             * "out of band" signals.
             *
             * Note that DT_TIMEOUT and DT_ERR must be negative values.
             */
            if ((c = dt_get(dt, sec)) > 0) {
                if ((dt->flag & _FLAG_EIGHTBIT) == 0)
                    c &= 0x7F;          /* Enforce 7-bit input  */
            }
            if (c == NUL || c == DEL)
                continue;               /* Ignore NUL, DEL      */
            /*
             * Branch to dt_continue when changing <ESC> 1
             * to CSI, etc.
             */
dt_continue:
            if (c == ESC                /* ESC, CSI, DCS        */
             || c == CSI                /* Introducer control   */
             || c == DCS) {             /* sequences.           */
                dt->seq.state           = c;
                dt->seq.inter[0]        = 0;
                dt->seq.private         = 0;
                dt->seq.param[0]        = 0;
                dt->seq.param[1]        = 0;
                continue;
            }
```

```
    else if (dt->seq.state == 0) /* No pending sequence */
        goto exit;               /* Return the character */
    else if ((c >= 0x80 && c <= 0x9F)    /* C1 control   */
          || (c == CAN)                  /* or sequence  */
          || (c == SUB)) {               /* recognizer.  */
        dt->seq.state = 0;
        goto exit;
    }
    else if (c < 0x20)           /* C0 control or error  */
        goto exit;
    else if (c <= 0x2F) {        /* Intermediate         */
        dt->seq.inter[0]++;
        if (dt->seq.inter[0] < SEQ_INTMAX)
            dt->seq.inter[dt->seq.inter[0]] = c;
    }
    else if (dt->seq.state == ESC) {    /* ESC final     */
        if (dt->seq.inter[0] == 0 && (c & 0x3F) < 0x20) {
            /*
             * This is the 7-bit form of a C1 control
             * character.  Convert it to the actual
             * C1 control character and restart the
             * parse without getting another character.
             */
            c = (c & 0x3F) + 0x80;
            goto c1_continue;
        }
        else {
            break;               /* Ordinary ESC ending  */
        }
    }
    else if (c <= 0x3F) {        /* Parameter            */
        if (c >= 0x3C) {         /* Private introducer?  */
            if (dt->seq.param[0] > 0)    /* Is it first? */
                dt->seq.private = 'X';   /* error if not */
            else {
                dt->seq.private = c;     /* Store it     */
                dt->seq.param[0]++;      /* Flag seen    */
            }
        }
        else {                   /* Not private          */
            if (dt->seq.param[0] == 0)
                dt->seq.param[0]++;      /* Record first */
            if (dt->seq.inter[0] != 0) {
                dt->seq.inter[0] = 0;    /* Syntax error */
                dt->seq.private = 'X';
            }
```

```
            if (c <='9') {      /* 0..9                     */
                if (dt->seq.param[0] <= SEQ_PARMAX) {
                    /*
                     * There is room.  Store it.
                     *
                     * p points to current parameter.
                     * This should check for value
                     * overflow.
                     */
                    p = &(dt->seq.param[dt->seq.param[0]]);
                    *p = (*p * 10) + (c - '0');
                }
                else {
                    dt->seq.private = 'F';
                }
            }
            else if (c == ';') {        /* Separator      */
                if (dt->seq.param[0] >= SEQ_PARMAX)
                    dt->seq.param[0] = SEQ_PARMAX + 1;
                else {
                    /*
                     * There's room to setup for
                     * another parameter value.
                     */
                    dt->seq.param[0]++;
                    dt->seq.param[dt->seq.param[0]] = 0;
                }
            }
            else {                      /* colon is invalid     */
                dt->seq.private = 'X';
            }
        }
    }
    else {                              /* CSI/DCS terminator   */
        if (dt->seq.param[0] == 0)  /* No parameters:   */
            dt->seq.param[0]++; /* want one zero-value  */
        break;                          /* Exit parser          */
    }
}
/*
 * Control transfers to here as result of either of the
 * two break statements. Character is the final char.
 * of ESC, CSI, or DCS.
 */
if (dt->seq.param[0] > SEQ_PARMAX) {
    dt->seq.param[0] = SEQ_PARMAX; /* Set count to max. */
    dt->seq.private = 'X';         /* Flag an error     */
};
dt->seq.final = c;                 /* Store final char. */
#ifdef __cplusplus
c/* dt->seq.state;                 /* Fetch return value */

#else
c = (unsigned short) dt->seq.state;
```

```
#endif
        dt->seq.state = 0;              /* No sequence pending  */
exit:                                   /* Here to return char  */
#ifdef  DT_DEBUG
        if (dt_debug)
            printf("\"\n");
#endif
        return (c);                     /* and return.          */
}
```

DTGET.C

This routine reads a character from the DECtalk terminal line.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_get          Read one Character from DECtalk
index                   Read one character from DECtalk

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_get(dt, sec)
        DECTALK         *dt;    /* Device descriptor    */
        int             sec;    /* O.S. timeout param.  */

description

        One character is read from the DECtalk terminal line.
        The sec parameter enables operating-system timeout;
        it is zero if no timeout is needed.

        dt_get() returns the character or an error code.

          DT_ERROR      An operating system error
                        (or <CTRL-C> interrupt) was received.
          DT_TIMEOUT    The sec parameter was nonzero and no
                        character was received in sec seconds.

        If DT_DEBUG is #defined when the library is compiled
        and the global dt_debug is set nonzero (by the
        application program), the character
        received is logged to the standard output device.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  dt_get
#undef  dt_get
#endif
```

```
int     dt_debug;

int
dt_get(dt, sec)
register DECTALK        *dt;    /* Device descriptor            */
int                     sec;    /* Operating system timeout     */
{
        register int    c;

        extern int      dt_debug;

        c = dt_toget(dt, sec);
        if (dt_debug != 0)
            dt_dchar(c, stdout);
        return (c);
}
```

DTHANG.C

Hang up the telephone connected to DECtalk.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_hangup       Hangup the telephone
index                   Hangup the telephone

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_hangup(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        Hang up the telephone connected to DECtalk.
        Return TRUE if successful, FALSE if an error.

#endif

#include        <stdio.h>
#include        "dectlk.h"

int
dt_hangup(dt)
register DECTALK        *dt;
/*
 * dt_hangup() hangs up the phone.
 */
{
        register int    code;

        dt_drain(dt);                   /* Drain pending text   */
        if (!dt_phone(dt, P3_PH_STATUS, -1)) /* Check state     */
          return (FALSE);               /* Oops                 */
        if (dt_offhook(dt)) {           /* If it's not hung up  */
            if (!dt_phone(dt, P3_PH_HANGUP, -1))
                return (FALSE);         /* Couldn't hangup?     */
            while (dt_offhook(dt)) {    /* While still off-hook */
                if (dt_abort)           /* Exit if interrupt    */
                    return (FALSE);     /* signal sets          */
                sleep(5);               /* Wait and poll again  */
                if (!dt_phone(dt, P3_PH_STATUS, -1))
                    return (FALSE);     /* Poll failed?         */
            }
        }
        if (!dt_onhook(dt))             /* Did it hang up ok?   */
          return (FALSE);
}
```

DTINIT.C

This routine initializes the DECtalk terminal on the channel opened on dt.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_init         DECtalk Initialization Routine
index                   DECtalk initialization routine

Synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_init(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        Initialize the DECtalk terminal on the channel opened
        on dt.

        Return TRUE if the device initialized successfully.
        Return FALSE on failure.

note

        This routine turns off "local mode" so a logging terminal
        does not inadvertently send a response to the "who are you"
        escape sequence.
#endif

#include        <stdio.h>
#include        "dectlk.h"

static SEQUENCE DT_who.are_you = {
    CSI, 'c'
};

int
dt_init(dt)
register DECTALK        *dt;
```

```
/*
 * dt_init() is called to initialize DECtalk.
 */
{
        register int    rows;

        dt_drain(dt);                   /* Ignore pending input */
        dt_dcm(dt, P2_TERMINAL, 0, -1); /* No local->host stuff */
        dt->flag |= _FLAG_TERM;         /* Remember this fact   */
        /*
         * Read device attributes and tell if it isn't DECtalk.
         * Expected reply is  <ESC>[?19c for the DTC01-AA
         */
        dt_puts(dt, &DT_who_are_you);
        if (dt_read(dt, 15) >= 0
         && dt->reply.final == 'c'
         && dt->reply.private >= '?'
         && dt->reply.inter[0] == 0
         && dt->reply.parm[0] >= 1
         && ((code = dt->reply.parm[1]) == CSI_DA_PRODUCT)) {
            dt_reset(dt);                       /* Reset device          */
            return (TRUE);                      /* Hang up and restart   */
        }
        return (FALSE);
}
```

DTINKE.C

This routine reads a telephone keypad button.

```
/*>LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_inkey        Read a Telephone Keypad Character
index                   Read a Telephone Keypad Character

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_inkey(dt, sec)
        DECTALK         *dt;    /* Device descriptor    */
        int             sec;    /* Seconds to wait      */

description

        This routine reads a telephone keypad button.
        The application program has previously enabled
        the keypad (by calling dt_keypad(dt, TRUE)).
        dt_inkey() will call dt_timeout() to enable
        or disable timeouts.

        If sec is nonzero, it will indicate the number of
        seconds to wait for a keypad response.  If zero,
        it will turn off keypad timeouts.
        The operating-system timeout (needed to catch
        hardware or communication line problems) will be
        set to four times the timeout value, plus an
        operating-system specific additional timeout.

        dt_inkey() returns a character as follows.

          0123456789*#ABCD      A valid keypad button
                                (Note that "ABCD" may be
                                generated by certain
                                keypad phones.)

          E                     An operating-system error

          T                     Keypad timeout

          X                     Badly parsed escape sequence.

          H                     Unexpected telephone hangup.

        The 'H' code is received if the DECtalk device hangs
        up the phone (ax may be required by specific telephone
        system requirements).
```

```
#endif

#include        <stdio.h>
#include        "dectlk.h"

/*
 * Fudge is needed because of terminal output buffering
 * capacities and strategies.  It should be tuned by
 * inspection.
 *
 * The RSTS/E value is large because RSTS/E will resume a
 * program when less than 80 bytes remain to be transmitted
 * to DECtalk.  DECtalk may have about 100 bytes in its input
 * buffers and two phrases in the letter to sound and
 * synthesizer sections.  If the value is set too low, the
 * application program may incorrectly assume that DECtalk
 * or the communication line is broken.
 */

#ifdef  RSTS
#define FUDGE   60              /* RSTS/E needs extra time      */
#else
#define FUDGE   16              /* This is just a guess         */
#endif

dt_inkey(dt, sec)
register DECTALK        *dt;    /* DECtalk device               */
int                     sec;    /* Keypad timeout               */
{
        register int    code;
        register int    os_timeout;

        dt_timeout(dt, sec);            /* Set/clear timeout    */
        if (dt_iskey(dt)) {
            code = dt->pendbt->pend.ep];
            if (++dt->pend_ep >= PEND_SIZE)
                dt->pend_ep = 0;
            dt->pend_fc--;
        }
        else {

            if ((code = dt_read(dt,
                    (sec == 0) ? 0 : (sec * 1 + FUDGE))) <= 0)
                code = 'E';
            else if (dt_istimeout(dt)) {
                code = 'T';
                dt->timeout = 0;
            }
            else if (dt_onhook(dt))
                code = 'H';
            else if (dt->reply_private == 'X'
             || !dt_isvalid(code))
                code = 'X';
        }
        return (code);
}
```

DTIOGE.C

This routine reads one character from the DECtalk terminal line. DTIOGE.C is maximized for efficiency.

```
/*LIBRARY
*/
/*
 * Edit History
 * 84.04.19     MM      PHGTTY incorrectly specified.
 */

#ifdef  DOCUMENTATION

title   dt_ioget        Read one Character from DECtalk
index                   Read one character from DECtalk

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_ioget(dt, sec)
        DECTALK         *dt;    /* Device descriptor    */
        int             sec;    /* O.S. timeout param.  */

description

        One character is read from the DECtalk terminal line.
        The sec parameter enables operating-system timeout;
        it is zero if no timeout is needed.

        dt_ioget() returns the character or an error code.

          DT_ERROR      An operating system error
                        (or <CTRL-C> interrupt) was received.

          DT_TIMEOUT    The sec parameter was nonzero and
                        no character was received in sec
                        seconds.

        dt_ioget() is the operating-system specific input
        routine.  It is the only routine to read data from
        the DECtalk terminal line.

notes

        On VMS, an internally-used routine, dt_vmsread(),
        is also defined.  Application programs should
        not call this routine.

        This module contains specific code for Ultrix-32 (UNIX
        4.2BSD) and UNIX System V.  The makefile for the library
        should #define one of these as appropriate.
```

```
#endif

#include        <stdio.h>
#include        "dectlk.h"

/*
 * Define all DECtalk library globals in this module.
 */
int     dt_abort;               /* TRUE on interrupt            */
DECTALK *dt_root;               /* Chain of all open units      */

#ifdef  unix
#include        <errno.h>
#ifdef  BSD_42
#include        <sys/types.h>
#include        <time.h>
#else
#ifdef  UNIX_V
#include        <signal.h>

static ignore() {}      /* Dummy function for signals           */
#endif
#endif

int
dt_ioget(dt, sec)
register DECTALK        *dt;    /* DECtalk device               */
int                     sec;    /* Wait time, 0 == forever      */
/*
 * UNIX:  Fill the input buffer, return the next (first) character.
 */
{
        register int    incount;        /* Count and error code */
#ifdef  BSD_42
        auto int        fdmask;         /* File descriptor mask */
        struct timeval  timeout;        /* Select() timer value */
#else
#ifdef  UNIX_V
        register int    ecode;          /* For error handling   */
        extern int      errno;          /* System error value   */
#endif
#endif

        /*
         * Return buffered character (if any)
         */
        if (dt->in_ptr < dt->in_end)
            return (*dt->in_ptr++ & 0xFF);
        /*
         * We must refill the buffer
         */
        dt->in_ptr = dt->in_end = &dt->in_buff[0];
        dt_ioput(dt, 0);                /* Flush output         */
        if (dt_abort)
            return (DT_ERROR);
```

```
#ifdef  BSD_42
        fdmask = 1 << dt->unit;         /* Select unit          */
        timeout.tv_usec = 0;            /* No milliseconds      */
        timeout.tv_sec = sec;           /* Max. seconds to wait */
        incount = select(dt->unit + 1, &fdmask, 0, 0, &timeout);
        if (incount < 0 && dt_abort)
            return (DT_ERROR);          /* Select failed?       */
        else if (incount == 0)          /* Timeout triggered?   */
            return (DT_TIMEOUT);        /* Guess so             */
        else {
            incount = read(dt->unit, dt->in_buff, IN_BUFLEN);
        }
#else
#ifdef  UNIX_V
        signal(SIGALRM, ignore);        /* Enable alarms        */
        alarm(sec);                     /* Start timeout        */
        errno = 0;                      /* Clear error flag     */
        incount = read(dt->unit, dt->in_buff, IN_BUFLEN);
        ecode = errno;                  /* Save error code      */
        alarm(0);                       /* Cancel timeout       */
        signal(SIGALRM, SIG_IGN);       /* Disable alarms       */
        if (incount < 0 && ecode == EINTR)  /* Did it timeout?  */
            return (DT_TIMEOUT);        /* Return failure       */
#endif
#endif
        if (dt_abort && incount <= 0)   /* Other error?         */
            return (DT_ERROR);          /* Return bad failure   */
        dt->in_end = dt->in_buff[incount];
        return (*dt->in_ptr++ & 0xFF);
}
#endif
#ifdef  vms
#include        <ssdef.h>       /* System status codes          */
#include        <iodef.h>       /* I/O request codes            */
/*
 * Define the possible vms input flavors
 */
#define RAW_READ        (IO$_READLBLK | IO$M_NOECHO | IO$M_NOFILTR)
#define TIMED_READ      (RAW_READ | IO$M_TIMED)

typedef struct io_status_block {
    short int   status;             /* I/O status code      */
    short int   term_offset;        /* Datum size           */
    short int   terminator;         /* Input terminator     */
    short int   term_size;          /* Terminator size      */
} IOSTAB;

int
dt_raget(dt, sec)
register DECTALK        *dt;    /* DECtalk device           */
int                     sec;    /* Wait time, 0 == forever  */
```

```
/*
 * VMS:  Fill the input buffer, return the next (first) character.
 */
{
        register int    incount;        /* Count and error code */

        /*
         * Return buffered character (if any)
         */
        if (dt->in_ptr < dt->in_end)
            return (*dt->in_ptr++ & 0xFF);
        /*
         * We must refill the buffer
         */
        dt->in_ptr = dt->in_end = &dt->in_buff[0];
        dt_oputf(dt, 0);                /* Flush output         */
        /*
         * First read anything in the system type-ahead
         * buffer by reading with a zero timeout.
         * If nothing was read, read one byte (with
         * timeout if specified).
         */
        incount = dt_vmsread(dt, TIMED_READ, IN_BUFLEN, 0);
        if (incount == DT_TIMEOUT) {
            incount = dt_vmsread(dt,
                (sec > 0) ? TIMED_READ : RAW_READ, 1, sec);
        }
        if (incount < 0)
            return (incount);   /* Return error code           */
        /*
         * Common exit from all read routines
         */
        dt->in_end = &dt->in_buff[incount];
        return (*dt->in_ptr++ & 0xFF);
}

int
dt_vmsread(dt, command, how_many, timeout)
register DECTALK        *dt;    /* DECtalk device               */
int             command;        /* QIO command                  */
int             how_many;       /* How many bytes to read       */
int             timeout;        /* Timeout value                */
```

```
/*
 * Actually read from vms.  Return
 *      (result > 0)    the number of bytes read
 *      (result < 0)    error code (EOF or TIMEOUT)
 *      (result cannot equal zero).
 */
{
        register int    lncount;        /* Status parameter     */
        IOSTAB          status;         /* I/O status block     */
        register int    i;              /* For debugging        */
        /*
         * tw-mset is a terminator mask indicating "terminate
         * on any character".  As implemented on VMS, this
         * allows the operating system to handle XOFF/XON.
         */
        static long termset[2] = { 0, 0 };

        if (dt_abort)
            return (DT_ERROR);
        /*
         * The status entries term_offset and term_size
         * will yield the number of bytes read.
         */
        lncount = sys$qiow(1,   /* Event flag                   */
                dt->unit,       /* Input channel                */
                command,        /* Timed read                   */
                &status,        /* I/O status block             */
                NULL,           /* AST block (none)             */
                0,              /* AST parameter                */
                dt->in_buff,    /* P1 - input buffer            */
                how_many,       /* P2 - buffer length           */
                timeout,        /* P3 - wait P3 seconds         */
                &termset,       /* P4 - terminator set          */
                NULL,           /* P5 - ignored (prompt buffer) */
                0);             /* P6 - ignored (prompt size)   */
        if (lncount == SS$_TIMEOUT)     /* Timeout returned     */
            return (DT_TIMEOUT);
        else if (lncount != SS$_NORMAL) /* Some other error     */
            return (DT_ERROR);
        lncount = status.term_offset + status.term_size;
        if (lncount <= 0)               /* Nothing input?       */
            return (DT_TIMEOUT);        /*  equals timeout.     */
        return (lncount);
}
#endif

#ifdef  rt11
/*
 *                              R S - S / E
 */
#include        <rsts.h>

int
dt_iget(dt, sec)
register DECTALK        *dt;    /* DECtalk device               */
int                     sec;    /* Wait time, 0 == forever      */
```

```
/*
 * RSTS/E: Fill the input buffer, return the next (first) character.
 */
{
        register int    incount;        /* Count and error code */

        /*
         * Return buffered character (if any)
         */
        if (dt->in_ptr < dt->in_end)
            return (*dt->in_ptr++ & 0xFF);
        /*
         * We must refill the buffer
         */
        dt->in_ptr = dt->in_end = &dt->in_buff[0];
        dt_ioput(dt, 0);                /* Flush output         */
        if (dt_abort)
            return (DT_ERROR);
        /*
         * RSTS/E handles timeout within
         * the operating system.
         */
        clrxrb();
        xrb.xrlen = 4;                  /* No delimiter         */
        xrb.xrci = dt->unit * 2;
        xrb.xrblkn = TTYHND;
        rsts$s(_SPEC);
        clrxrb();
        xrb.xrlen = 3;                  /* No echo              */
        xrb.xrci = dt->unit * 2;
        xrb.xrblkn = TTYHND;
        rsts$s(_SPEC);
        incount = rs_read(dt->unit, dt->in.buff,
            IN_BUFLEN, 0, 0, sec, 0);
        if (incount == (-HNGTTY))       /* 84.04.10             */
            return (DT_TIMEOUT);
        else if (incount <= 0)
            return (DT_ERROR);
        /*
         * Common exit from all read routines
         */
        dt->in_end = &dt->in_buff[incount];
        return (*dt->in_ptr++ & 0xFF);
}
#endif

#ifdef  rsx
```

```
/*
 * Load in RSX specific information:
 *      cx.h            common header
 *      qiofun.h        I/O service function codes
 *      qioret.h        I/O service error and status codes
 *      qiottd.h        Terminal I/O service bits and bytes
 *      iunbuf.h        Device characteristics buffer
 */

#include        <cx.h>
#include        <qiofun.h>
#include        <qioret.h>
#include        <qiottd.h>
#include        <iunbuf.h>

#define Q1O_EFN 1                       /* I/O event flag       */
#define MKT_EFN 2                       /* Time event flag      */
static char     gmcbuf[2] = { 1O_TEF }; /* get typeahead count  */
static Q1OPARM  gmcparm = { gmcbuf, sizeof gmcbuf };
static int      termtable[4];           /* Terminator bitmask   */

int
dt_ogetl(dt, sec)
register DECIALK        *dt;    /* DECtalk device               */
int                     sec;    /* Wait time, 0 == forever      */
/*
 * RSX:  Fill the input buffer, return the next (first) character.
 */
{
        register int    incount;        /* Count and error code */
        register char   *ip;            /* To copy to my buff   */
        int             errorcode;
        int             efn_buffer[4];  /* Event flag buffer    */

        /*
         * Return buffered character (if any)
         */
        if (dt->in_ptr < dt->in_end)
            return (*dt->in_ptr++ & 0xFF);
        /*
         * We must refill the buffer
         */
        dt->in_ptr = dt->in_end = &dt->in_buff[0];
        dt_iqput(dt, 0);                /* Flush output         */
        if (dt->abort)
            return (DT_ERROR);
        if (dt->pos_xk) {
```

```
        /*
         * The PRO-350 XX: port is actually pretty simple.
         */
        dt->parm.buffer = dt->ir_buff;
        dt->parm.size = IN_BUFLEN;
        dt->parm.p2 = 0;                        /* No timeout   */
        errorcode = qiow(IO_RLB | TF_TMO, dt->unit, QIO_EFN,
            &dt->iosb, NULL, &dt->parm);
        if ((incount = fixiosb(dt)) == 0) {
            dt->parm.size = 1;
            if ((dt->parm.p2 = (266 * sec)) == 0) {
                errorcode = qiow(IO_RLB, dt->unit, QIO_EFN,
                &dt->iosb, NULL, &dt->parm);
            }
            else {
                errorcode = qiow(IO_RLB | TF_TMO, dt->unit,
                QIO_EFN, &dt->iosb, NULL, &dt->parm);
            }
            if (errorcode != IS_SUC) {
                return ((errorcode == IS_TMO)
                    ? DT_TIMEOUT : DT_ERROR);
            }
            if ((incount = fixiosb(dt)) == 0) {
                return (DT_TIMEOUT);
            }
        }
    }
    else {
        /*
         * Read from a terminal.
         * First, check whether anything is in the
         * system type-ahead buffer.
         */
        errorcode = qiow(SF_GMC, dt->unit, QIO_EFN,
            &dt->iosb, NULL, &gmcparm);
        if (errorcode != IS_SUC)
            gmcbuf[1] = 0;
        dt->parm.buffer = dt->ir_buff;
        dt->parm.size = 1;              /* Assume 1 byte read   */
        if ((incount = (gmcbuf[1] & 0xFF)) > 0) {
            if (incount > IN_BUFLEN)
                incount = IN_BUFLEN;
            dt->parm.size = incount;
            errorcode = qiow(IO_RTT | TF_RNE, dt->unit, QIO_EFN,
                &dt->iosb, NULL, &dt->parm);
            incount = fixiosb(dt);
        }
        if (incount == 0) {
            if (esc == 0) {
                dt->parm.table = termtable;
                qiow(IO_RTT | TF_RNE, dt->unit, QIO_EFN,
                    &dt->iosb, NULL, &dt->parm);
                if ((incount = fixiosb(dt)) == 0)
                    return (DT_ERROR);
            }
            else {
```

```
                    /*
                     * VAX compatibility doesn't support read with
                     * timeout (nor does it cause an error).  Thus,
                     * we have to do this the hard way.
                     *
                     * Set a mark time (qlerm) for "timeout" seconds.
                     * Read one byte without waiting.  If the wait
                     * completes, cancel the timeout.  If the timeout
                     * completes, cancel the readin.
                     */
                    if (mrkt(MKT_EFN, sec, 2, NULL) != IS_SUC
                     || qio(IO_RTT | TF_RNE, dt->unit, QIO_EFN,
                            &dt->iosb, NULL, &dt->parm) != IS_SUC)
                        return (DT_ERROR);      /* Can't happen */
                    /*
                     * Wait until something completes.
                     * Read event flags then cancel the
                     * request that didn't complete.
                     */
                    wtlo(QIO_EFN | MKT_EFN);
                    rdef(efn_buffer);
                    if ((efn_buffer[0] & MKT_EFN) == 0)
                        cm$(MKT_EFN, NULL);     /* Cancel timer */
                    if ((efn_buffer[0] & QIO_EFN) == 0) {
                        qiow(IO_KIL, dt->unit, QIO_EFN,
                            &dt->iosb, NULL, &dt->parm);
                        return (DT_TIMEOUT);
                    }
                    if ((incount = fixiosb(dt)) == 0)
                        return (DT_ERROR);
                }
            }
        }
        /*
         * Common (success) exit from all read routines
         */
        dt->in_end = &dt->in_buff[incount];
        return (*dt->in_ptr++ & 0xFF);
}
static int
fixiosb(dt)
register DECTALK        *dt;    /* DECtalk device                */
```

```
/*
 * This routine returns the correct input count.
 * The code is unusual.
 *
 * fixiosbf() returns the true byte count.
 */
{
        extern int      **ferr;

        if (dt->iosb.terminator != NUL) {
            /*
             * Append the terminator to the buffer.
             */
            dt->in_buff[dt->iosb.count] = dt->iosb.terminator;
            dt->iosb.count++;
        }
        if (dt_abort
         || dt->iosb.status == IE_ABO
         || dt->iosb.count == 0) {
            return (0);                             /* Reqo aborted */
        }
        else if (dt->iosb.status != IS_SJO
              && dt->iosb.status != IS_TMO) {
            **ferr = dt->iosb.status;
            return (0);                             /* I/O error    */
        }
        return (dt->iosb.count);
}
#endif
```

DTIOPU.C

If the argument character is zero, or output buffer is full, this routine writes output buffer contents to the DECtalk device. Otherwise, DTIOPU.C stores the character in a local buffer.

```
/*>LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_ioput        Write one Character to DECtalk
index                   Write one character to DECtalk

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_ioput(dt, byte)
        DECTALK         *dt;    /* Device descriptor    */
        char            byte;   /* Character to write   */

description

        If the argument character is zero, or the output
        buffer is full, the output buffer contents are
        written to the DECtalk device.

        If the argument character is nonzero, it is stored
        in the output buffer for subsequent transmission.

        By buffering characters internally, the load on the
        operating system is significantly reduced.  Note that
        the input routine (dt_get(), dt_ioget()) will flush
        the output buffer before attempting to read any data.
        The "speak" routine, dt_talk(), also flushes the
        output buffer.

        No data is returned.  Errors are fatal.

        dt_ioput() is the operating-system specific output
        routine.  It is the only routine to write data to
        the DECtalk terminal line.
```

```
#endif

#include        <stdio.h>
#include        "declik.h"
#ifdef  vms
#include        <ssdef.h>
#include        <iodef.h>

typedef struct io_status_block {
    short int   status;                 /* I/O status code      */
    short int   term_offset;            /* Datum size           */
    short int   terminator;             /* Input terminator     */
    short int   term_size;              /* Terminator size      */
} IOSTAB;

#endif

#ifdef  rsx
/*
 * Load in RSX specific information:
 *      cx.h            common header
 *      qiofun.h        I/O service function codes
 *      qioret.h        I/O service error and status codes
 *      qiotid.h        Terminal I/O service bits and bytes
 */

#include        <cx.h>
#include        <qiofun.h>
#include        <qioret.h>
#include        <qiotid.h>

#define DIO_EFN 1                       /* I/O event flag       */
#endif

dl_xputc(ds, c)
register DECTALK        *dl;    /* DECtalk device               */
int                     c;      /* Character to output          */
/*
 * Store the byte (if not EOS). If the byte is EOS,
 *      or the buffer is full, write it out.
 */
{
        register int    size;
#ifdef  vms
        register int    code;
        IOSTAB          status;
#endif
#ifdef  rt11
        register int    code;
        extern int      $$ferr;
#endif
#ifdef  rsx
        register int    code;
        extern int      $$ferr;
#endif

        if (c != 0) {
#ifdef  rt11
            *dl->out_ptr++ = (c == ESC) ? (ESC | 0x80) : c;
#else
            *dl->out_ptr++ = c;
```

```
#endif
        }
        size = (dt->out_ptr - dt->out_buff);
        if ((c != 0 && size > 0) || size >= OUT_BUFLEN) {
            /*
             * We must write the buffer.
             */
            if (!dt_abort) {
#ifdef  unix
                if (write(dt->unit,
                        dt->out_buff, size) == -1) {
                    perror(dt->device);
                    exit(1);
                }
#endif
#ifdef  vms
                if ((code = sys$qiow(1,  /* Event flag            */
                    dt->unit,            /* Input channel         */
                    IO$_WRITELBLK | IO%M_NOFORMAT,  /* format    */
                    &status,             /* I/O status block      */
                    NULL,                /* No AST block          */
                    0,                   /* No AST parameter      */
                    dt->out_buff,        /* P1 - buffer           */
                    size,                /* P2 - bytes            */
                    0,                   /* P3 - ignored          */
                    0,                   /* P4 - no carriage ctl  */
                    0,                   /* P5 - ignored          */
                    0))                  /* P6 - ignored          */
                            != SS$_NORMAL) {
                    perror(dt->device);
                    exit(code);
                }
#endif
#ifdef  rt11
                if ((code = .swrite(dt->unit, dt->out_buff,
                        size, 0, 0, 0)) != 0) {
                    ##ferr = code;
                    perror(dt->device);
                    exit(IO_ERROR);
                }
#endif
#ifdef  rsx
                dt->parm.size = size;
                dt->parm.buffer = dt->out_buff;
                dt->parm.table = NULL;
                if ((code = qiow(IO_WAL, dt->unit, 0|IO_EFN,
                        &dt->iosb, NULL, &dt->parm)) != IS_SUC) {
                    ##ferr = code;
                    perror(dt->device);
                    exit(IO_ERROR);
                }
#endif

            dt->out_ptr = dt->out_buff;
        }
}
```

DTISKE.C

This routine returns TRUE if the telephone user already typed any characters.

```
/*>LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_iskey        Test for type-ahead
index                   Test for type-ahead

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_iskey(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        This routine (which may be implemented as a macro)
        returns TRUE if any characters have already been
        typed by the telephone user, or if an asynchronous
        status message (such as timeout) was received.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  dt_iskey
#undef  dt_iskey
#endif

int
dt_iskey(dt)
register DECTALK        *dt;    /* Device descriptor            */
/*
 * Test for type-ahead.
 */
{
        return(dt->pend_fc != 0);
}
```

DTISTI.C

Used to test the result of a dt_phone () message for keypad timeout.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_istimeout    Test Phone Reply for Keypad Timeout
index                   Test phone reply for keypad timeout

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_istimeout(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        This routine (which may be implemented as a macro)
        tests the result of a dt_phone() message.
        It returns TRUE if the current reply is the DECtalk
        phone reply with the R3 parameter equal to R3_PH_TIMEOUT.

#endif

#include        <stdio.h>
#include        "dectlk.h"
#ifdef  dt_istimeout
#undef  dt_istimeout
#endif

int
dt_istimeout(dt)
register DECTALK        *dt;    /* Device descriptor            */
/*
 * Test for telephone keypad timeout.
 */
{
        return (dt_test(dt, R2_PHONE, R3_PH_TIMEOUT));
}
```

DTISVA.C

This routine returns TRUE if the argument character is one of 0123456789#*ABCD.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_isvalid      Test for Valid Keypad Character
index                   Test for valid keypad character

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_isvalid(c)

description

        This routine (which may be implemented as a macro)
        returns TRUE if the argument character is one of

            0123456789#*ABCD

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  dt_isvalid
#undef  dt_isvalid
#endif

int
dt_isvalid(c)
char            c;
/*
 * Test for valid pushbutton key.
 */
{
        return (    (c >= '0' && c <= '9')
                 || c == '*' || c == '#'
                 || (c >= 'A' && c <= 'D'));
}
```

DTKEYP.C

This routine enables the telephone keypad if the flag is TRUE, and disables the keypad if it is FALSE.

```
/*LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_keypad       Enable or Disable the Telephone Keypad
index                   Enable or Disable the Telephone Keypad

Synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_keypad(dt, flag)
        DECTALK         *dt;    /* Device descriptor    */
        int             flag;   /* TRUE to enable       */

description

        Enable the telephone keypad if the flag is TRUE,
        disable it if FALSE.

        Returns TRUE if successful.  If FALSE, the telephone
        may have been hung up.

#endif

#include        <stdio.h>
#include        "dectlk.h"

int
dt_keypad(dt, enable)
register DECTALK        *dt;
int                     enable;
/*
 * Enable or disable the telephone keypad.
 */
{
        dt_phone(dt,
            (enable) ? PB_PH_KEYPAD : PB_PH_NOKEYPAD, -1);
        if (dt_offhook(dt))
            return (TRUE);
        return (FALSE);
}
```

DTMSG.C

This routine sends a DECtalk DCS control sequence using the p2, p3, and p4 parameters. The r2 and r3 parameters are not checked by the module. A FALSE reply means an error occurred.

The user may have pressed keypad buttons or a timeout may have occurred. These values are saved for use by the dt_save routine.

```
/*DLITERARY
*/

#ifdef  DOCUMENTATION

title   dt_msg          Send a DECtalk Command with Reply
index                   Send a DECtalk command with reply

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_msg(dt, p2, p3, p4, r2, r3)
        DECTALK         *dt;    /* Device descriptor    */
        int             p2;     /* P2_xxxx parameter    */
        int             p3;     /* P3_PH_xxxx parameter */
        int             p4;     /* timeout or rings     */
        int             r2;     /* R2_xxxx parameter    */
        int             r3;     /* R3_xxxx parameter    */

description

        This routine sends a DECtalk DCS control sequence
        using the p2, p3, and p4 parameters.  It then reads
        a DCS reply from DECtalk, returning TRUE if it matches
        the r2 and r3 calling parameters.

        If p2 is -1, no sequence is sent, but a DCS reply
        is read and tested.

        Note that the Pn and Rn parameters are -1 if they
        are not sent or checked respectively.

        Returns TRUE if successful.  If FALSE, something is funny.

        Note: dt_msg() saves user keypad characters in the type-ahead
        buffer.
```

```
#end;f

#include        <stdio.h>
#include        "dectlk.h"

int
dt_msg(dt, p2, p3, p4, r2, r3)
register DECTALK        *dt;    /* Device descriptor            */
int             p2, p3, p4;     /* Pr parameters to send        */
int             *r2, *r3;       /* Reply R2 and R3 parameters   */
/*
 * Send a DECtalk DCS message and wait for a reply.
 * Return TRUE if the proper reply was received.
 */
{
        register int    code;

        if (p2 != -1)
            dt_dcs(dt, p2, p3, p4);     /* Send the sequence    */
        do {
            code = dt_rmsg(dt, 60);
        } while (code == ST || dt_save(dt, code));
        return (dt_test(dt, r2, r3));   /* Check result         */
}
```

DTOFFH.C

This routine tests the result of a dt_phone () message for OFFHOOK.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_offhook      Test Phone Reply for Offhook
index                   Test phone reply for Offhook

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_offhook(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        This routine (which may be implemented as a macro)
        is used to test the result of a dt_phone() message.
        It returns TRUE if the current reply is the DECtalk
        phone reply with the R3 parameter equal to R3_PH_OFFHOOK.

#endif

#include        <stdio.h>
#include        "dectlk.h"
#ifdef  dt_offhook
#undef  dt_offhook
#endif

int
dt_offhook(dt)
register DECTALK        *dt;    /* Device descriptor            */
/*
 * Test whether the phone is off-hook.
 */
{
        return (dt_testfdt, R2_PHONE, R3_PH_OFFHOOK));
}
```

DTONHO.C

This routine tests the result of a dt_phone () message for ONHOOK.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_onhook       Test Phone Reply for Onhook
index                   Test phone reply for onhook

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_onhook(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        This routine (which may be implemented as a macro)
        is used to test the result of a dt_phone() message.
        It returns TRUE if the current reply in the DECtalk
        phone reply with the R3 parameter equal to R3_PH_ONHOOK.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  dt_onhook
#undef  dt_onhook
#endif

int
dt_onhook(dt)
register DECTALK        *dt;    /* Device descriptor            */
/*
 * Test whether the phone is on-hook.
 */
{
        return (dt_test(dt, R2_PHONE, R3_PH_ONHOOK));
}
```

DTOPEN.C

This routine performs operating-specific initializations to initiate communications with a DECtalk device. Operating systems include UNIX, RSX, RSTS/E, and VMS (either compatibility or native modes).

```
/*DLIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_open         Connect to DECtalk Terminal
index                   Connect to DECtalk terminal

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        DECTALK *
        dt_open(dev)
        char            *dev;   /* Terminal device name */

description

        Perform operating-specific initializations to
        initiate communication with a DECtalk device.
        (This routine is similar to fopen() for FILE
        devices.)  If the open fails, return NULL;
        else return a pointer to a data descriptor
        block that will be used for all other DECtalk
        operations.

        If the open failed, the standard library perror()
        routine may be called to print error information.

        This routine does not communicate with DECtalk.

        For example, the following sequence opens DECtalk,
        checks that it is responding, sets "square-bracket"
        mode, and speaks a message:

            #include <stdio.h>
            #include "dectlk.h"

            DECTALK *dt;
            ...
            main() {
                if ((dt = dt_open("kb2:")) == NULL) {
                    perror("kb2:");
                    printf("Can't open DECtalk\n");
                }
                else if (!dt_init(dt))
                    printf("Can't initiate DECtalk\n");
                else {
                    dt_dos(dt, P2_MODE, MODE_SQUARE, -1);
                    dt_talk(dt, "Hello world.");
                    dt_sync(dt);
                    dt_close(dt);
                    printf("Success.\n");
                }
            }
```

```
UNIX notes

        This routine conditionally compiles for Ultrix-32 (4.2BSD)
        and System V.  There is also a conditional for the Zilog
        Zeus version of UNIX.  This hasn't been independently checked.

        UNIX implementors are encouraged to read and understand
        this module when developing DECtalk applications.

        UNIX and System V are trademarks of AT&T Bell Laboratories.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  unix
/*
 * UNIX specific definitions
 */
#include        <signal.h>
#include        <errno.h>
#endif

#ifdef  vms
/*
 * VMS native specific definitions
 */
#include        <ssdef.h>       /* Status definitions           */
#include        <iodef.h>       /* I/O request codes            */
#include        <dscdsc.h>      /* String descriptors           */

typedef struct dsc$descriptor_s STRING; /* string descriptor    */
/*
 * The following macro builds a descriptor from an argument string.
 */
#define descriptext(p)                          \
        (p)->dsc$a_pointer = text,              \
        (p)->dsc$w_length = strlen(text),       \
        (p)->dsc$b_dtype = DSC$K_DTYPE_T,       \
        (p)->dsc$b_class = DSC$K_CLASS_S)

#endif

#ifdef  rt11
/*
 * RSTS/E native specific definitions
 */
#include        <rsts.h>
#endif

#ifdef  rsx
#include        <rsx.h>
#include        <qiofun.h>
#include        <qiorel.h>
#include        <qiofld.h>
#include        <clunbuf.h>
#define SIG_EFN 1
static QIOPARM  noparm;                 /* QIO parm (all zero)  */
#endif
```

```
DECTALK *
dt_open(name)
char            *name;                  /* Device name          */
/*
 * Initialize the DECtalk terminal line.
 */
{
        register DECTALK        *dt;
        register int    i;              /* Channel search, temp */

#ifdef  unix
        register char   *ttyname();     /* -> stdin name        */
#ifdef  BSD_42
        struct sgttyb   tty_buffer;     /* Terminal flags       */
#else
#ifdef  UNIX_V
        struct termio   tty_buffer;     /* Terminal flags       */
#endif
#endif
        extern char     *ttyname();     /* Get stdin name       */
#endif
#ifdef  vms
        $IOSB           dev;
#endif
#ifdef  rt11
        char            work[30];
        extern int      $$rsts;
        extern char     *E$$NOD;        /* Invalid device       */
        extern char     *E$$FAT;        /* Fatal error          */
        extern char     *E$$NOC;        /* No more channels     */
#endif
#ifdef  rsx
        extern int      $$rsts;         /* TRUE if RSTS/C       */
        extern int      $$pos;          /* TRUE if P/OS         */
        extern int      $dsw;           /* Dir. status word     */
        extern int      $$ferr;         /* DECUS C error value  */
        struct lunbuf   lunbuf;         /* Get lun information  */
#endif
        extern char     *calloc();
        extern char     *malloc();

        /*
         * Allocate the DECtalk buffer and save the
         * device name (for debugging).
         */
        if ((dt = (DECTALK *)calloc(sizeof (DECTALK), 1)) == NULL)
            return (NULL);
        if ((dt->device = malloc(strlen(name) + 1)) == NULL)
            goto error2;
        strcpy(dt->device, name);
```

```
#ifdef  unix
        if ((ttyname = ttyname(fileno(stdin))) != NULL
         && strcmp(ttyname, name) == 0)
            dt->unit = fileno(stdin);           /* stdin                */
        else if ((dt->unit = open(name, 2)) < 0)
            goto error1;
        if (!isatty(dt->unit)) {
            close(dt->unit);
            goto error1;
        }
        /*
         * Force the terminal into single-character, no-echo mode.
         */
#ifdef  BSD_42
        gtty(dt->unit, &stty_buffer);   /* Get current info     */
        gtty(dt->unit, &dt->stty_save); /* For restore, too     */
        stty_buffer.sg_flags &= ~ECHO;  /* Set no echo          */
        stty_buffer.sg_flags |= CBREAK; /* Single character     */
        stty(dt->unit, &stty_buffer);   /* Set temp. mode       */
        signal(SIGALRM, SIG_IGN);       /* Ignore timer signals */
#else
#ifdef  UNIX_V
        ioctl(dt->unit, TCGETA, &stty_buffer);  /* Get current info     */
        ioctl(dt->unit, TCGETA, &dt->stty_save);/* For restore, too     */
        stty_buffer.c_iflag = BRKINT | IXON | IXOFF;
        stty_buffer.c_oflag = 0;
        stty_buffer.c_cflag = B9600 | CS8 | CREAD | CLOCAL;
#ifdef  zeus
        /*
         * The following edit was reported by a customer for a Zilog
         * "Zeus" port but hasn't been independently tested.
         */
        stty_buffer.c_iflag = 0101;
#else
        stty_buffer.c_iflag = 0;
#endif
        stty_buffer.c_cc[VMIN] = 1;
        stty_buffer.c_cc[VTIME] = 2;
        ioctl(dt->unit, TCSETA, &stty_buffer);  /* Set temp. mode       */
#endif
#endif
#endif
#ifdef  vms
        descrip(name, &dev);
        if (sys$assign(&dev, &dt->unit, 0, NULL) != SS$_NORMAL)
            goto error1;
```

```
#endif
#ifndef __t11
        if (!$$rsts) {
            $$terr = (int) &E$$FNT;     /* Illegal function     */
            goto error1;
        }
        /*
         * Search for a free channel.
         */
        for (i = 12; i > 0; i--) {
            clrxrb();
            xrb.xrci = i * 2;
            if (rstsys(&_POSTN) == NOTOPN)
                break;
        }
        if (i <= 0) {                   /* Fail if all channels */
            $$terr = (int) &E$$NCC;     /* are in use.          */
            goto error1;
        }
        dl->unit = i;                   /* Save unit number     */
        /*
         * On RSTS, the terminal is opened in a special mode:
         *        1      binary
         *       16      do not abort on CTRL-C or modem hangup
         *       32      terminal service handles XOFF/XON
         */
        sprint?(work, "%s/mn:%d", name, '+'&+32);
        if (rs_open(i, work, "r") != 0)
            goto error1;
        if ((rsrqb.fq[1]&0 & 0xFF) != TTYKBD) {
            $$terr = (int) &E$$NDD;     /* Not a terminal       */
            rs_close(i);
            goto error1;
        }
#endif
#ifdef  rsx
        if ($$rsts) {
            $$terr = IE_IFC;            /* Not on RSTS/E        */
            goto error1;
        }
        /*
         * We only call fopen() to get a free lun.
         */
        if ((dl->fildes = fopen(name, "r+")) == NULL)
            goto error1;
        dl->unit = fileno(dl->fildes);
        glun(dl->unit, &lunbuf);
        if (&&pos
         && lunbuf.g_luna[0] == 'X'
         && lunbuf.g_luna[1] == 'K')
            dl->pos_xk = TRUE;
        else
            dl->pos_xk = FALSE;
        if ((i = qiow(IO_RTT, dl->unit, QIO_EFN,
                NULL, NULL, &rpgarm)) != IS_SUC) {
            fcloseall(dl->fildes);
            $$terr = i;
            goto error1;
        }
```

```
#endif
        /*
         * Normal exit, initialize other pointers
         */
        dt->link = dt_root;
        dt_root = dt;
        dt->out_ptr = dt->out_buff;     /* Out buffer setup     */
        dt->flag = _FLAG_SPEAK;         /* Normally speaking    */
        return (dt);                    /* Normal exit          */

error1: free(dt->device);               /* Error, free device   */
error2: free((char *) dt);              /* and DECTALK buffer   */
        return ((DECTALK *)NULL);       /* Error exit           */
}
```

DTPEEK.C

This routine tests if a character is pending from DECtalk. The character may be a keypad character (user selected) or part of an escape sequence.

```
/*3LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_peek         Test if Character Available from DECtalk
index                   Test if character available from DECtalk

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_peek(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        Returns TRUE if a character is pending from DECtalk.
        Note that this may be a keypad input character (as
        entered by the user) or part of an escape sequence.

        dt_peek() does not flush pending output.  It contains
        operating-system specific code.

note

        This module contains specific code for UNIX
        4.2BSD.  The makefile for the library should
        #define BSD_42.

bugs

        Tested only on VMS.

#endif

#include        <stdio.h>
#include        "dectlk.h"

/*      :
 * Define all DECtalk library globals in this module.
 */

#ifdef  unix
#include        <errno.h>
#ifdef  BSD_42
#include        <sys/types.h>
#include        <time.h>
#endif
```

```
int
dt_peek(dt, sec)
register DECTalk        *dt;    /* DECtalk device                       */
/*
 * UNIX.
 */
{
        register int    incount;        /* Count and error code */
#ifdef  BSD_42
        auto long       pending;        /* Number pending       */
#else
        register int    ecode;          /* For error handling   */
        extern int      errno;          /* System error value   */
#endif

        /*
         * Anything buffered?
         */
        if (dt->pend_fc > 0 || dt->in_ptr < dt->in_end)
            return (TRUE);
#ifdef  BSD_42
        /*
         * Works for 4.1 BSD, too.
         * Won't work for Unix V7 or System V (V >= 3)
         */
        ioctl(dt->unit, FIONREAD, &pending);
        return(pending > 0);
#else
        dt->in_ptr = dt->in_end = &dt->in_buffer[0];
        alarm(1);                       /* Start timeout        */
        errno = 0;                      /* Clear error flag     */
        incount = read(dt->unit, dt->in_buff, IN_BUFLEN);
        ecode = errno;                  /* Save error code      */
        alarm(0);                       /* Cancel timeout       */
        if (incount < 0 && ecode == EINTR)  /* Did it timeout?  */
            return (FALSE);     /* Return failure       */
        /* (dt_abort || incount <= 0)   /* Other error?         */
            return (FALSE);             /* Return bad failure   */
        dt->in_end = &dt->in_buff[incount];
        return (TRUE);
#endif
}
#endif

#ifdef  vms
#include        <ssdef.h>               /* System status codes  */
#include        <iodef.h>               /* I/O request codes    */
typedef struct io_status_block {
    short int   status;                 /* I/O status code      */
    short int   term_offset;            /* Datum size           */
    short int   terminator;             /* Input terminator     */
    short int   term_size;              /* Terminator size      */
} IOSTAT;
```

```
int
dt_peek(dt)
register DECTALK        *dt;    /* DECtalk device                */
/*
 * VMS:  Fill the input buffer, too.
 */
{
        register int    incount;
        struct type_ahead {
            short       pending_count;
            char        first_character;
            char        char_reserved;
            int         long_reserved;
        } type_ahead;
        IOSTAT          status;         /* I/O status block      */

        if (dt->pend_fx > 0 || dt->in_ptr < dt->in_end)
            return (TRUE);
        incount = sys$qiow(1,   /* Event flag                    */
                dt->unit,       /* Input channel                 */
                IO$_SENSEMODE | IO$M_TYPEAHDCNT,
                &status,        /* I/O status block              */
                NULL,           /* AST block (none)              */
                0,              /* AST parameter                 */
                &type_ahead,    /* P1 - buffer                   */
                sizeof type_ahead, /* P2 - buffer length         */
                0,              /* P3 -                          */
                NULL,           /* P4 -                          */
                NULL,           /* P5 - ignored (prompt buffer)  */
                0);             /* P6 - ignored (prompt size)    */
        return (incount == SS$_NORMAL && type_ahead.pending_count > 0);
}
#endif

#ifdef  rt11
/*
 *                              R S T S / E
 */
#include        <rsts.h>

int
dt_peek(dt)
register DECTALK        *dt;    /* DECtalk device                */
```

```
/*
 * RSTS/E: Fill the input buffer, return the next (first) character.
 */
{
        register int    incount;        /* Count and error code */

        if (dt->pend_fc > 0 && dt->in_ptr < dt->in_end)
            return (TRUE);
        /*
         * We must refill the buffer
         */
        dt->in_ptr = dt->in_end = &dt->in_buff[0];
        qlrxrb();
        xrb.xrlen = 4;                  /* No delimiter         */
        xrb.xrcc = dt->unit * 2;
        xrb.xrblkn = (TYHND;
        rsts$s(_SPEC);
        qlrxrb();
        xrb.xrlen = 0;                  /* No echo              */
        xrb.xrcc = dt->unit * 2;
        xrb.xrblkn = TTYHND;
        rsts$s(_SPEC);
        incount = ra_rread(dt->unit, dt->in_buff,
            IN_BUFLEN, 0, 0, 0, 8192);
        if (incount == -(HNG^^Y))
            return (FALSE);
        else if (incount <= 0)
            return (FALSE);
        dt->in_end = &dt->in_buff[incount];
        return (TRUE);
}
#endif

#ifdef  rsx
/*
 * Load in RSX specific information:
 *      cx.h            common header
 *      qiofun.h        I/O service function codes
 *      qioret.h        I/O service error and status codes
 *      qiotid.h        Terminal I/O service bits and bytes
 *      iunbuf.h        Device characteristics buffer
 */

#include        <cx.h>
#include        <qiofun.h>
#include        <qioret.h>
#include        <qiotid.h>
#include        <iunbuf.h>

#define QIO_EFN 1                       /* I/O event flag       */
#define MKT_EFN 2                       /* Time event flag      */
static char     gmcbuf[2] = { TC_TBF };/* get typeahead count  */
static QIOPARM  gmcparm = { gmcbuf, sizeof gmcbuf };
static int      terminate[8];           /* Terminator bitmask   */
```

```
int
dt_peek(dt)
register DECTALK        *dt;    /* DECtalk device               */
/*
 * RSX:
 */
{
        register int    inbount;        /* Count and error code */
        register char   *ip;            /* To copy to rsx buff  */
        int             errorcode;
        /*
         * Return buffered character (if any)
         */
        if (dt->peek.fc > 0 || dt->in_ptr < dt->in_end)
            return (TRUE);
        /*
         * We must refill the buffer
         */
        dt->in_ptr = dt->in_end = &dt->in_buff[0];
        if (dt->pos_xt) {
            /*
             * The PRO-350 RK: port is actually pretty simple.
             */
            dt->parm.buffer = dt->in_buff;
            dt->parm.size = IN_BUFLEN;
            dt->parm.p3 = 0;                        /* no timeout   */
            errorcode = qiow(IO_RLB | TF_TMO, dt->unit, DTC_EFN,
                &dt->iosb, NULL, &dt->parm);
            if ((inbount = fixiosb(dt)) == 0) {
                return (FALSE);
            }
            dt->in_end = &dt->in_buff[inbount];
            return (TRUE);
        }
        else {
            /*
             * Check whether anything is in the
             * system type-ahead buffer.
             */
            errorcode = qiow(SF_GMC, dt->unit, DTC_EFN,
                &dt->iosb, NULL, &gmcparm);
            return (errorcode == IS_SUC && gmcbuf[1] > 0);
        }
}
```

```
static int
fixiosb(dt)
register DECTALK        *dtk;   /* DECtalk device                   */
/*
 * This routine returns the correct input count.
 * The code is unusual.
 *
 * fixiosb() returns the true byte count.
 */
{
        extern int      errno;

        if (dt->iosb.terminator != NUL) {
            /*
             * Append the terminator to the buffer.
             */
            dt->in_buff[dt->iosb.count] = dt->iosb.terminator;
            dt->iosb.count++;
        }
        if (dt_abort
         || dt->iosb.status == SS_ABO
         || dt->iosb.count == 0) {
            return (0);                         /* Read aborted */
        }
        else if (dt->iosb.status != SS_SUC
              && dt->iosb.status != SS_TMO) {
            errno = dt->iosb.status;
            return (0);                         /* I/O error    */
        }
        return (dt->iosb.count);
}
#endif
```

DTPESC.C

This routine compiles an appropriate escape sequence from the parameter buffer.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_pesc         Transmit Escape Sequence
index                   Transmit escape sequence

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"


        dt_pesc(dt, seq)
        DECTALK         *dt;    /* Device descriptor    */
        SEQUENCE        *seq;   /* What to transmit     */

description

        Compile an appropriate escape sequence from the
        parameter buffer.  This is similar to putchar()
        except when seq->state is ESC, CSI, or DCS.  In
        these cases, the function generates an appropriate
        sequence from the passed data structure.  dt_pesc()
        calls the user-supplied dt_put() to output each
        character.

        If control sequences are sent in their eight-bit
        form if _FLAG.EIGHTBIT is set in dt->flag.  If
        this bit is off, they are sent in their <ESC>X
        form.  If the application program sets _FLAG.EIGHTBIT
        it must also ensure that the operating system
        transmits eight data bits, and that DECtalk
        was setup as HOST FORMAT EIGHT.

        No value is returned.

#endif
```

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

dt_pmsc(dt, seq)
register DECTALK        *dt;    /* Dectalk device               */
register SEQUENCE       *seq;   /* Sequence buffer              */
/*
 * Output the character (in seq->state) and maybe a sequence, too.
 */
{
        register unsigned i;    /* Index into inter[], param[], */
        unsigned        max;    /* Max for inter[] and param[], */

#ifdef  DT_DEBUG
        if (dt_debug) {
            printf("put: \"");
            if (isatty(fileno(stdout)))
                fflush(stdout);
        }
#endif
        i = seq->state;
        if ((dt->flag & _FLAG_EIGHTBIT) == 0
         && i >= 0x80 && i <= 0x9F) {
            /*
             * Output is in 7-bit mode and the character is
             * a C1 control character.  Convert it.
             */
            dt_put(dt, ESC);
            dt_put(dt, i - 0x40);
        }
        else {
            /*
             * Not the special case; output the character.
             */
            dt_put(dt, i);
        }
        switch (i) {
        case ESC:
        case CSI:
        case DCS:
            /*
             * Here is a sequence.  Output all of its components.
             *
             * First, the parameters.
             *  i       counts the parameters
             *  max     stores the parameter max.
             *  val     working copy of parameter value.
             */
            if (seq->private != 0)
                dt_put(dt, seq->private);
            max = seq->param[0];
            if (max > SEQ_PARMAX)
                max = SEQ_PARMAX;       /* too many, use limit  */
            for (i = 1; i <= max; i++) {
                if (i > 1)
                    dt_put(dt, ';');
                if (seq->param[i] != 0)
                    intout(dt, seq->param[i]);
            }
```

```
            /*
             * Output intermediates.
             *  i       counts intermediates.
             *  max     stores the number to output.
             */
            max = seq->inter[0];
            if (max > SEQ_INTMAX)
                max = SEQ_INTMAX;       /* Too many, use limit  */
            for (i = 1; i <= max;) {
                dt_putfdt, seq->inter[i++]);
            }
            dt_putfdt, seq->float);             /* Output the final     */
            break;

        default:
            break;
        }
#ifdef  DT_DEBUG
        if (dt_debug)
            printf("\"\n");
#endif
}
#ifndef INT_32
#ifdef  vax
#define INT_32
#endif

#ifdef  MC68000
#define INT_32
#endif
#endif

static unsigned power10[] =
{ /*
   * Powers of 10 for intout
   */
#ifdef  INT_32
    1000000000,
     100000000,
      10000000,
#endif
        1000000,
         100000,
          10000,
           1000,
            100,
             10,
              1,
};

#define NPOWERS ((sizeof power10) / (sizeof (unsigned)))

static
intout(dt, value)
DECTALK             *dt;    /* DECtalk device               */
register unsigned   value;  /* Value to convert             */
/*
 * Convert an unsigned number to ASCII and call dt_putf() on
 * each character.  Note, as implemented here, a zero value
 * does not output anything.
 */
```

```
{
        register unsigned       *power;
        int                     out_char;
        int                     nonzero;

        power = power10;        /* Pointer to power table       */
        nonzero = FALSE;        /* Don't output leading zeros   */
        do {
            /*
             * Loop until all places except digits place
             * have been done.
             */
            for (out_char = 0; value >= *power; out_char++)
                value -= *power;        /* Subtract a power     */
            if (nonzero || out_char > 0) {
                nonzero = TRUE;         /* Not leading zero     */
                dt_putf(dt, out_char + '0');
            }
        } while (++power < &power10[NPOWERS]);
}
```

DTPHON.C

This routine sends a DECtalk phone message.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_phone        Send a Phone Message
index                   Send a phone message

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_phone(dt, p3, p4)
        DECTALK         *dt;    /* Device descriptor    */
        int             p3;     /* P3_PH_xxxx parameter */
        int             p4;     /* timeout or rings     */

description

        This routine (which may be implemented as a macro)
        sends a DECtalk phone message (i.e., the p2 parameter
        is P2_PHONE).

        p3 and p4 should be given as -1 if no parameter is to
        be sent.

        It then reads the status reply and returns TRUE
        if the r1 and r2 parameters are R1_DECTALK and
        R2_PHONE respectively.  The application program
        should then test for offhook/onhook as appropriate.

        Returns TRUE if successful.  If FALSE, something is funny.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  dt_phone
#undef  dt_phone
#endif

int
dt_phone(dt, p3, p4)
register DECTALK        *dt;    /* Device descriptor            */
int                     p3;
int                     p4;
/*
 * Send a phone message.
 */
{
        return (dt_msg(dt, P2_PHONE, p3, p4, R2_PHONE, -1));
}
```

DTPTES.C

This routine tests a phone reply.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_ptest        Test Phone Reply
index                   Test phone reply

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_ptest(dt, r3)
        DECTALK         *dt;    /* Device descriptor    */
        int             r3;     /* R3_PH_xxxx parameter */

description

        This routine (which may be implemented as a macro)
        is used to test the result of a dt_phone() message.
        The parameter is a R3_PH_... reply value.
        It returns TRUE if the current reply is a DECtalk
        phone reply with the specified R3 parameter.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  dt_ptest
#undef  dt_ptest
#endif

int
dt_ptest(dt, r3)
register DECTALK        *dt;    /* Device descriptor            */
int                     r3;
/*
 * Test a phone message.
 */
{
        return (dt_test(dt, R2_PHONE, r3));
}
```

DTPUT.C

This routine sends one character to the DECtalk terminal line. No value is returned.

```
/*$LIBRARY -
*/

#ifdef  DOCUMENTATION

title   dt_put          Write one Character to DECtalk
index                   Write one character to DECtalk

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_put(dt, c)
        DECTALK         *dt;    /* Device descriptor    */
        int             c;      /* Character to write   */

description

        One character is written to the DECtalk terminal line.
        No value is returned.

        If DT_DEBUG is #defined when the library is compiled
        and the global dt_debug is set nonzero (by the
        application program), the character
        written is logged to the standard output device.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  dt_put
#undef  dt_put
#endif

dt_put(dt, c)
register DECTALK        *dt;    /* Device descriptor            */
register int            c;      /* Character to write           */
{
        extern int      dt_debug;

        dt_ioput(dt, c);
#ifdef  DT_DEBUG
        if(dt_debug != 0)
            dt_dchar(c, stdout);
#endif
}
```

DTREAD.C

This routine reads a sequence or character.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_read         Read Sequence or Character
index                   Read Sequence or character

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_read(dt, sec)
        DECTALK         *dt;    /* Device descriptor    */
        char            sec;    /* 0.5. timeout value   */

description

        Read an escape sequence or keypad character.  Ignore
        Any characters between the DECtalk final and the
        string terminator.  Return the character read or
        the sequence introducer.

#endif

#include        <stdio.h>
#include        "dectlk.h"

int
dt_read(dt, sec)
register DECTALK        *dt;    /* Dectalk device               */
int                     sec;    /* Operating system timeout     */
{
        register int    code;
        register int    i;

        /*
         * Read another sequence (or continue reading this one),
         * Copy the sequence read into the working "reply" buffer.
         * Note, this code is not quite general enough for all
         * escape sequence parsing.  Specifically, it cannot
         * properly deal with CC control characters embedded
         * inside of escape sequences (as is necessary if the
         * operating system cannot process XOFF/XON controls).
         */
```

```
        dt->seq.state = 0;
again:  dt->reply.state = code + dt_gesc(dt, seq);
        switch (code) {
        case CAN:
        case SUB:
            goto again;

        case ESC:
        case CSI:
        case DCS:
            dt->reply.final = dt->seq.final;
            dt->reply.private = dt->seq.private;
            for (i = 0; i <= dt->seq.inter[0]; ++i)
                dt->reply.inter[i] = dt->seq.inter[i];
            for (i = 0; i <= dt->seq.param[0]; ++i)
                dt->reply.param[i] = dt->seq.param[i];
            break;

        default:
            dt->reply.final = dt->reply.private =
            dt->reply.inter[0] = dt->reply.param[0] = 0;
            break;
        }
        if (dt->reply.state == DCS) {
            /*
             * Ignore text between DCS final and ST
             */
            dt->seq.state = 0;
            do {
                code = dt_gesc(dt, '\0');
            } while (code > 0 && code < 0x80);
        }
        return (dt->reply.state);
}
```

DTRESE.C

This routine sends a soft-reset escape sequence.

```
/*).IBRARY
*/

#:fdef  DOCUMENTATION

title   dt_reset        DECtalk Soft Reset
index                   DECtalk soft reset

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_reset(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        Send a "soft reset" escape sequence.
        No errors are possible.

#endif

#include        <stdio.h>
#include        "dectlk.h"

static SEQUENCE soft_reset = {
    CS[, 'p', 0, { 0 }, { 1, '!' }
};


dt_reset(dt)
register DECTALK        *dt;
/*
 * dt_reset() sends a soft-reset escape sequence.
 */
{
        dt_peac(dt, &soft_reset);
        dt->flag |= _FLAG_SPEAK;                /* Speaking now         */
        dt->timeout = 0;                        /* No timeout now       */
}
```

DTSAVE.C

This routine saves user type-ahead characters.

```
/*)_LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_save         Save User Type-ahead
index                   Save user type-ahead

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_save(dt, c)
        DECTALK         *dt;    /* Device description   */
        char            c;      /* Character to save    */

description

        If c is a keypad character, save it in the
        type-ahead buffer and return TRUE, else return
        FALSE.

        If the current reply is a timeout and nothing
        is stored in the type-ahead buffer, save 'T'
        and clear the timeout flag.  This is necessary
        as a timeout sequence may be returned in
        the middle of a message/reply sequence.

        This routine should not be called by application
        programs.

#endif

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

```
int
dt_save(dt, c)
register DECTALK        *dt;    /* Dectalk device                   */
int                     c;      /* Character to test                */
{
        register int    timeout;        /* Current value            */

        if (!ct_isvalid(c)) {           /* Not a keypad button?     */
            if (!dt_istimeout(dt))      /* If it isn't timeout,     */
                return (FALSE);         /* it's not for us.         */
            else {                      /* Timeout is funny         */
                /*
                 * Ignore timeout if timer is set to zero or
                 * something is already in the typeahead buffer.
                 */
                timeout = dt->timeout;  /* Get old value            */
                dt->timeout = 0;        /* Clear timer              */
                if (timeout == 0 || dt_iskey(dt))
                    return (TRUE);      /* Toss it away             */
                c = 'T';                /* Save it in typeahead     */
            }
        }
        if (dt->pend_fp < PEND_SIZE) {  /* Save it if there's       */
            dt->pend_fp++;              /* enough room, else        */
            dt->pend[dt->pend_fp] = c;  /* throw it away.           */
            if (++dt->pend_fp >= PEND_SIZE)
                dt->pend_fp = 0;
        }
        return (TRUE);
}
```

DTSPLICE.C

This routine lets you control a terminal connected to DECtalk's local port.

```
/*>LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_splice       Manage Local Terminal
index                   Manage Local Terminal

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_splice(dt, flag)
        DECTALK         *dt;    /* Device descriptor    */
        int             flag;   /* Required state       */

description

        dt_splice() allows control over a terminal
        connected to DECtalk's local port.  Note that
        the terminal must correctly process ANSI escape
        sequences.  Specifically, it must ignore any
        escape sequence that it doesn't understand.

        The flag parameter may have the following
        (bit-encoded) values.

          SPLICE_SPEAK      Speak subsequent text,
                            if set.  Do not speak text
                            if not set. Initially zero.

          SPLICE_LOG        Text sent to DECtalk is sent
                            (in raw mode) to the local
                            terminal if set.  Initially
                            not set.

          SPLICE_TERM       Text typed on the local
                            terminal is sent to DECtalk
                            if set. Initially not set.

        The bits would normally be set and cleared in combination.
        For example:

            dt_splice(dt, SPLICE_SPEAK);

        Speak text, don't log it, ignore text typed on the host.

            dt_splice(dt, SPLICE_LOG | SPLICE_TERM);

        Stop speaking text, transmit text from/to the attached
        terminal.
```

```
#endif

#include        <stdio.h>
#include        "dectlk.h"

dt_splice(dt, flag)
register DECTALK        *dt;
register int            flag;
/*
 * Manage line-splice modes.
 */
{
        splice(dt, flag & SPLICE_SPEAK, _FLAG_SPEAK,
                                P2_SPEAK, 1);
        splice(dt, flag & SPLICE_LOG,   _FLAG_LOG,
                                P2_LOG, LOG_RAW+SST);
        splice(dt, flag & SPLICE_TERM,  _FLAG_TERM,
                                P2_TERMINAL, TERM_HOST);
}

static
splice(dt, flag, bit, p2, p3)
register DECTALK        *dt;
int                     flag;           /* TRUE to set bit      */
int                     bit;            /* dt->flag bit to do   */
int                     p2, p3;         /* For DCS              */
/*
 * Do the dt_splice() work.  If dt->flag doesn't agree with flag,
 * send the appropriate dt_dcs().
 */
{
        if (((dt->flag & bit) != 0) != (flag != 0)) {
            if (flag != 0) {            /* Turn mode on,        */
                dt->flag |= bit;        /* Set flag bit and     */
                dt_dcs(dt, p2, p3, -1); /* Sends the p2/p3      */
            }
            else {                      /* Turn mode off,       */
                dt->flag &= ~bit;       /* Clear flag bit and   */
                dt_dcs(dt, p2, 0, -1);  /* Send "mode off"      */
            }
        }
}
```

DTST.C

This routine sends a string terminator to DECtalk. This string terminates phonemic text or telephone dial commands.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_st           Send String Terminator
index                   Send String Terminator

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_st(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        This routine sends a string terminator to DECtalk.
        This is needed to terminate phonemic text or telephone
        dial commands.

        A phonemic text sequence would be sent as follows.

            dt_cmd(dt, p2, p3);
            dt_talk(dt, "hh'ehlow.");
            dt_st(dt);
#endif

#include        <stdio.h>
#include        "dectlk.h"

static SEQUENCE string_terminator = {
    ST
};

dt_st(dt)
DECTALK                 *dt;            /* Device descriptor    */
/*
 * Send a string terminator
 */
{
        dt_pseq(dt, &string_terminator);
}
```

DTSYNC.C

This routine synchronizes the application with DECtalk.

```
/*DLIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_sync         Synchronize with DECtalk
index                   Synchronize with DECtalk

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_sync(dt)
        DECTALK         *dt;    /* Device descriptor    */

description

        The program delays until all text sent to DECtalk
        has been spoken.

        Returns TRUE if successful.  If FALSE, something is funny.

#endif

#include        <stdio.h>
#include        "dectlk.h"

int
dt_sync(dt)
register DECTALK        *dt;    /* Device descriptor            */
/*
 * Synchronize DECtalk and the application.
 */
{
        dt_dosfdt, P2_SYNC, -1, -1);    /* Synchronize          */
        dt->flag |= _FLAG_SPEAK;        /* Now speaking         */
        return (dt_wsp(dt, P2_IX_QUERY, -1, -1, P2_IX_QUERY, -1));
}
```

DTTALK.C

This routine speaks one line of text.

```
/*DLIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_talk         Speak One Line of Text
index                   Speak one line of text

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_talk(dt, text)
        DECTALK         *dt;    /* Device descriptor    */
        char            *text;  /* What to say          */

description

        This function sends a line of text to DECtalk.

        dt_talk(dt, NULL) flushes DECtalk by sending
        a vertical-tab sequence.

#endif

#include        <stdio.h>
#include        "dectlk.h"

static char     vtline[] = { VT, 0 };

dt_talk(dt, text)
register DECTALK        *dt;    /* Device descriptor            */
register char           *text;  /* Text pointer                 */
{
        if (text == NULL)
            text = vtline;
        while (*text != 0)
            dt_put(dt, *text++);
        dt_eol(dt);
}
```

DTTEST.C

This routine tests a DECtalk reply.

```
/*3LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_test         Test a DECtalk Reply
index                   Test a DECtalk reply

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_test(dt, r2, r3)
        DECTALK         *dt;    /* Device descriptor    */
        int             r2;     /* R2_xxx parameter     */
        int             r3;     /* R3_xxx parameter     */

description

        This routine checks the last reply received from
        DECtalk against the model.  r3 is -1 to ignore it.
        It returns TRUE if the reply is a properly parsed
        DECtalk reply sequence, or FALSE on any failure.

#endif

#include        <stdio.h>
#include        "dectlk.h"

int
dt_test(dt, r2, r3)
register DECTALK        *dt;    /* Device descriptor            */
int                     r2;
int                     r3;
/*
 * Test the current returned sequence for the proper
 * DECtalk reply.  r3 is -1 to ignore it.
 */
{
        if (dt->reply.status == DCS
          && dt->reply.final == DCS_F_DECTALK
          && dt->reply.inter[0] == 0
          && dt->reply.private == 0) {
            if (dt->reply.param[1] == R1_DECTALK
             && dt->reply.param[2] == r2
             && (r3 == -1 || dt->reply.param[3] == r3))
                return (TRUE);
        }
        return (FALSE);
}
```

DTTIME.C

This routine enables or disables telephone keypad timeout.

```
/*)LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_timeout      Enable or Disable Keypad Timeout
index                   Enable or disable keypad timeout

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_timeout(dt, sec)
        DECTALK         *dt;    /* Device descriptor    */
        int             sec;    /* Timeout in seconds   */

description

        If sec is nonzero, timeouts are being enabled;
        if zero, they are being disabled.

        Enable keypad timeouts if sec is nonzero and there
        is no data in the type-ahead buffer (and timeouts
        are not already enabled).

        Disable timeouts if they are enabled and sec is zero,
        or any data is in the type-ahead buffer (even if
        sec is nonzero).

        Before enabling timeouts, DECtalk is synchronized.

        Returns TRUE if successful.  If FALSE, the telephone
        may have been hung up.

#endif

#include        <stdio.h>
#include        "dectlk.h"

int
dt_timeout(dt, sec)
register DECTALK        *dt;    /* Device descriptor            */
register int            sec;    /* Timeout in sec seconds       */
```

```
/*
 * Enable or disable timeout.  No errors are possible.
 * Note that timeout(dt, 15) looks at the state of the
 * type-ahead buffer before deciding whether to turn
 * timeouts on, off, or to do nothing.
 */
{
        if (sec != 0) {                 /* If enabling,             */
            if (dt_inkey(dt))           /* Disable if typeahead */
                sec = 0;
            if (sec != 0) {             /* Still enabling?          */
                dt_sync(dt);            /* Synchronize and          */
                if (dt_inkey(dt))       /* Check again.             */
                    sec = 0;
            }
        }
        if (dt->timeout == sec)         /* Don't set to the         */
            return (TRUE);              /* same value               */
        dt_phone(dt, PB_PH_TIMEOUT, sec);
        dt->timeout = sec;
        if (dt_onhook(dt) || dt_offhook(dt))
            return (TRUE);
        return (FALSE);
}
```

DTTONE.C

This routine sends the msg test string as a tone dialing sequence.

```
*/

#ifdef  DOCUMENTATION

title   dt_tone         Send DTMF Tones
index                   Send DTMF tones

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        int
        dt_tone(dt, msg)
        DECTALK         *dt;    /* Device descriptor    */
        char            *msg;   /* Announcement         */

description

        This routine sends the msg text string as a tone
        dialing sequence.  If the telephone was on-hook
        when dtinref() was called, it will be returned
        to the on-hook condition.  Note, this routine
        may not work to your satisfaction in countries
        which require automatic announcement messages
        on automatically dialed calls.  See your DECtalk
        programmer's manual for more information.

        For message text  may contain any valid touch-tone
        characters ("0123456789*#ABCD") or the characters
        'T' (for a one second delay) or the 'F' for a 250
        millisecond switch-hook flash. All other characters
        are ignored.

        Note that the telephone will not be hung up before
        dialing if it is offhook when the command is issued.

#endif

#include        <stdio.h>
#include        "dectlk.h"

int
dt_tone(dt, message)
register DECTALK        *dt;            /* Device descriptor    */
char                    *message;       /* Announcement         */
```

```
/*
 * Send tones.
 */
{
        register int    state;
        register int    code;

        dt_phone(dt, -1, -1);
        state = dt_onhook(dt);
        dt_cmd(dt, P2_PHONE, P3_PH_TONE);
        dt_talk(dt, message);
        dt_st(dt);
        do {
            code = dt_read(dt, 30);
        } while (code == ST || dt_save(dt, code));
        if (state)
            dt_hangup(dt);
}
```

DTTRAP.C

This routine traps CTRL-C interrupts.

```
/*DLIBRARY.
*/

#ifdef  DOCUMENTATION

title   dt_trap         Trap <CTRL-C> interrupts
index                   Trap <CTRL-C> interrupts

synopsis

        #include        <stdio.h>
        #include        "dectlk.h"

        dt_trap()

description

        Set the global dt_abort flag if the user types
        <CTRL-C> at the command terminal.  (On UNIX,
        this is interpreted as catching the INTERRUPT
        signal, which is not necessarily <CTRL-C>, and
        which may be generated by running the "kill"
        system program.

        When the interrupt is received, pending I/O
        is cancelled (on those operating systems where
        this makes sense).

        If dt_abort is set TRUE when the interrupt is received,
        the program aborts.

        No error is returned.

#endif

#include        <stdio.h>
#include        "dectlk.h"

#ifdef  UNIX
#include        <signal.h>

static
catch()
{
        if (dt_abort)
            _exit(2);
        dt_abort = TRUE;
}

dt_trap().
```

```
/*
 * Trap CTRL-C interrupts.
 */
{
        signal(SIGINT, catch);
        signal(SIGTERM, catch);
}
#endif

#ifdef  vms
#include        <ssdef.h>
#include        <signal.h>

static
catch()
{
        register DECTALK        *dt;

        if (dt_abort)
            _exit(SS$_ABORT);
        dt_abort = TRUE;
        for (dt = dt_root; dt != NULL; dt = dt->link)
            sys$cancel(dt->unit);
}

dt_trap()
/*
 * Trap CTRL-C interrupts.
 */
{
        signal(SIGINT, catch);
}
#endif

#ifdef  vax
#include        <ex.h>
#include        <qio fun.h>
#include        <qioret.h>
#include        <qiotid.h>
#define QIO_EFN 1

static
catch()
{
        register DECTALK        *dt;

        astset();                       /* AST entry            */
        if (dt_abort)
            $$fatal();
        dt_abort = TRUE;
        /*
         * Kill all pending DECtalk I/O
         */
        for (dt = dt_root; dt != NULL; dt = dt->link) {
            qiow(IO_KIL, dt->unit, QIO_EFN,
                &dt->iosb, NULL, &dt->parm);
        }
        astx(1);                        /* AST exit             */
}
```

```
static SIGPARM  sestparm = { NULL, 0, catch };

dt_trap()
/*
 * Trap CTRL-C interrupts.
 */
{
        qiow(IO_ATN, fileno(stderr), QIOC_CTN,
            NULL, NULL, &sestparm);
}
#endif
#ifdef  rtt\

static
catch()
/*
 * Executed by the operating system if an interrupt is
 * detected.
 */
{
        if (dt_abort)
            abort();
        dt_abort = TRUE;
}

dt_trap()
/*
 * Trap CTRL-C interrupts.
 */
{
        setcc(catch);
}
#endif
```

DTVISI.C

This routine generates a visible ASCII representation of a character stored in the work buffer.

```
/*$LIBRARY
*/

#ifdef  DOCUMENTATION

title   dt_visible      Generate Visible Representation
index                   Generate Visible Representation

synopsis

        char            work[12]; /* Output buffer      */

        char *
        dt_visible(c, work)
        int             c;      /* Character to dump    */
        char            work[]; /* Work buffer          */

description

        A visible ASCII representation of the character
        is stored in the work buffer.  A pointer to the
        end of the output string is returned.

        Note that this routine is independent of DECtalk
        definitions (except that it knows about the DT_ERROR
        and DT_TIMEOUT error codes).

#endif

#include        <stdio.h>
#include        "dectlk.h"

char *
dt_visible(c, buffer)
register int    c;              /* Character to convert         */
register char   *buffer;        /* Where to store conversion    */
```

```
/*
 * Make a character "visible" ASCII.
 * Return a pointer to the trailing EOS.
 */
{
        register int    flag;

        switch (c) {
        case NUL:
            strcpy(buffer, "<NUL>");
            break;

        case DT_ERROR:
            strcpy(buffer, "<ERROR>");
            break;

        case DT_TIMEOUT:
            strcpy(buffer, "<TIMEOUT>");
            break;

        case ESC:
            strcpy(buffer, "<ESC>");
            break;

        case DCS:
            strcpy(buffer, "<DCS>");
            break;

        case CSI:
            strcpy(buffer, "<CSI>");
            break;

        case ST:
            strcpy(buffer, "<ST>");
            break;

        default:
            flag = (c >= 0x7F
                || (c < ' ' && c != '\n' && c != '\r'));
            if (flag)
                *buffer++ = '<';
            if ((c & 0xFF) >= 0x80) {
                c -= 0x80;
                *buffer++ = '~';
            }
            if (flag && c < ' ') {
                *buffer++ = '^';
                *buffer++ = c + 64;
            }
            else if (flag || c == '\n' || c >= ' ')
                *buffer++ = c;
            if (flag)
                *buffer++ = '>';
            *buffer = EOS;
            return (buffer);
        }
        return (buffer + strlen(buffer));
}
```

HELLO.C

This is a very simple test program, to show that DECtalk is operating correctly.

```
/*)BUILD
        $(INCLUDE) = ( dectlk.h )
        $(RTLIB) = ( dtlib,cretalb,c:clib )
        $(RXLIB) = ( dtlib/lb,c:ex/lb,c:c/lb )
*/

#include <stdio.h>
#include "dectlk.h"
#ifdef  vms
extern int              errno;
#define IO_ERROR        errno
#endif
#ifdef  unix
#define IO_ERROR        1
#endif

DECTALK *dt;

main(argc, argv)
int             argc;
char            *argv[];
{
        char            *dev;

        dev = "tty7:";
        if (argc > 1)
            dev = argv[1];
        if ((dt = dt_open(dev)) == NULL) {
            perror(dev);
            printf("Can't open DECtalk\n");
            exit(IO_ERROR);
        }
        dt_debug = TRUE;                /* log text             */
        dt_trap();                      /* CTRL-C trap enabled  */
        printf("calling init\n");
        if (dt_init(dt))
            printf("Can't initiate DECtalk\n");
        else {
            printf("Initialized\n");
            dt_def(dt, P2_NODE, NODE_SQUARE, -1);
            dt_talk(dt, "Hello world.");
            dt_sync(dt);
            dt_dump("after sync", &dt->reply);
            dt_close(dt);
            printf("Success.\n");
        }
}
```

BASIC-PLUS PROGRAM EXAMPLE 7

This chapter provides the source listings of a simple DECtalk telephone answering program, written in BASIC-PLUS for RSTS/E. You can copy and use this program; however, the program is only a model, and cannot cover all possible DECtalk applications.

RSTS/E SYSTEMS

On some RSTS/E systems, you may need system manager privileges to run this program. Please refer to the appropriate RSTS/E manuals for more information.

```
10      EXTEND
20      !*                                                      &
        !* DECtalk function library and a sample application.   &
        !* The function library generally duplicates the C     &
        !* library, with some minor simplifications.           &
        !*                                                     &
        !* The sample program reads a string of numbers from   &
        !* the keypad and speaks them as a number, and as a    &
        !* string of digits.  The '*' key functions as a dollar &
        !* sign, and the '#' key functions as a decimal point. &
        !* The program also illustrates how an application     &
        !* might manage keypad timeouts.                       &
        !*                                                     &

100     !*                                                      &
        !* Defaults                                            &
        !*                                                     &
        DEF.tbl$ = "KB2:"               ! DECtalk device        &
        \ DEF.log$ = "yes"              ! Assume log?           &
```

```
1000    !*                                                          &
        !* Main program                                             &
        !*                                                          &
        ! Initialize DECtalk and start the DEMO                    &
        ! Channel 1     Console keyboard for parameters            &
        ! Channel 2     Log file                                   &

1010    open "kb:" for input as file 1%                             &
        \ kb$ = FNprompt$("Declak terminal", DEF.kb$)               &
        \ DT.log% = (FNyesno%("Enable logging", DEF.log$))          &
        \ debug% = FNyesno%("Enable debug printouts", "yes")        &
        \ retries%, ncalls% = 0%          ! Clear counters          &
        \ error.count% = 0%               ! No errors yet           &
        \ if (debug% or DT.log%) then     ! Need a log file.        &
            logfile$ = FNprompt$("Debug log file", "kb:")           &
            \ open logfile$ for output as file 2%                   &

1100    while (FNinit%(kb$))              ! Initialize DECtalk      &
            \ q% = FNlog%("Initialization")                         &
            \ retries% = retries% + 1%    ! Count initializations   &
            \ while (FNanswer%)           ! Answer the phone        &
                \ if (FNprocess%) then    ! Do this call            &
                    ncalls% = ncalls% + 1%      ! Got a call        &
                    \ retries% = 0%             ! Clear retry       &

1200                goto 1800 if (debug% and error.count% > 0%)     &
                    \ if (retries% > 2%) then       ! Trouble?      &
                        q% = FNlog%("Too many retries")             &
                        \ goto 1800                 ! Fatal.        &

1300            next                        ! For all calls         &
            \ next                          ! For all restarts      &

1400    q% = FNlog%("Finished after " + num1$(ncalls%))             &
        \ close 2% if DT.log%                                       &

1900    goto 32767                          ! All done              &

2000    def* FNprocess%                                             &
        !                                                           &
        !       F N p r o c e s s %                                 &
        !                                                           &
        ! User process.  Read a number from the keypad and         &
        ! speak it out.  Return when phone is to be hung up.       &
        ! Return TRUE% if ok, FALSE% on error.                     &
        !                                                           &

2010    FNprocess% = FALSE%               ! Assume failure          &
        \ nkeys% = 0%                     ! Count button presses    &
        \ q% = FNlog%("answered")                                   &
        \ q% = FNspeak%("[:np :re 100] Welcome to DECtalk.")        &
        \ q% = FNspeak%("It is now " + time$(0%)                    &
                + " on " + date$(0%) + ".")                         &
        \ q% = FNspeak%("Enter a number, the star key means")       &
        \ q% = FNspeak%("dollar sign, while the number-sign")       &
        \ q% = FNspeak%("key means decimal point.")                 &
        \ if (not FNphone%("20")) then    ! turn the keypad on      &
            q% = FNlog%("error enabling keypad")                    &
            \ goto 2080                   ! Error exit              &
```

```
2080    if (not FNptest%(R3.FH.OFFHOOK%)) then                  &
          q% = FNlog%("enable keypad, state: " + num1$(R3%))    &
        \ goto 2080                     ! Error exit            &

2030    while TRUE%                     ! For all numbers       &
        \ timer% = 10%                  ! For first character   &
        \ work$ = ""                    ! Input buffer          &
        \ while TRUE%                   ! Get the number        &
            \ c% = FNkey%(timer%)       ! Read a character      &
            \ c$ = chr$(c%)             ! Get both flavors      &
            \ goto 2080 if c$ = "H" ! Hangup                    &
            \ goto 2080 if c$ = "E" ! Error from RSTS/E         &
            \ goto 2080 if c$ = "X" ! Escape sequence error     &
            \ goto 2050 if c$ = "T" ! Timeout                   &
            \ c$ = "*"  if c$ = "A" ! Fix funny                 &
            \ c$ = "."  if c$ = "#" ! buttons                   &
            \ work$ = work$ + c$    ! Stuff it                  &
            \ timer% = 2%           ! Short prompt now          &
        \ next                      ! Read a number loop        &

2050        goto 2060 if (work$ = "")   ! Did we read anything? &
          \ q% = FNspeak%("You entered " + work$ + ",")         &
        \ q% = FNspeak%("that )s" + FNexpand$(work$) + ".")     &
      \ next                            ! Read all numbers      &

2090    FNprocess% = TRUE%              ! Normal completion     &

2080    q% = FNphone%("21")             ! Turn off keypad       &
      \ q% = FNhangup%                  ! And hang up the phone &
      \ q% = FNlog%("process exit after " + num1$(nkeys%))      &

2090    fnend                                                   &

3000    !*                                                      &
        !*              f n e x p a n d $ ( t e x t $ )         &
        !*                                                      &
        !* Expand a number string into its component bytes.     &
        !* Note that this would be useful in a "bank by phone"  &
        !* application to speak a number, digit by digit, so    &
        !* the caller could copy it down.  If the input is      &
        !* "12.9", the output will be " 1 2 point 3".  Note     &
        !* the leading blank.                                   &
        !*                                                      &
        def* FNexpand$(text$)                                   &
        \ q$ = ""                               ! Output work   &
        \ for q% = 1% to len(text$)             ! For each byte &
            \ q1$ = mid(text$, q%, 1%)          ! Locate it     &
            \ q1$ = "point" if q1$ = "."        ! Fix the       &
            \ q1$ = "minus" if q1$ = "-"        !   special     &
            \ q1$ = "dollar sign" if q1$ = "$"  !      cases    &
            \ q$ = q$ + " " + q1$               ! and stuff it  &
        \ next q%                               ! Do 'em all    &
        \ FNexpand$ = q$                        ! That's it     &

3090    fnend                                                   &
```

```
10000   !*                                                              &
        !* Basic-Plus Support functions for DECtalk                     &
        !* Note that the code is not particularly fast and some        *
        !* of the error conditions that are handled by the C           *
        !* version of the Escape Sequence parser are ignored.          *
        !*                                                             &
        ! Note: the following channels are used:                       &
        !       8       DECtalk input                                  &
        !       9       DECtalk output                                 &
        !       2       Log file                                       &
        !               If DT.log% is TRUE, a log file is open         &
        !               on channel 2                                   &
        !                                                              &
        !*                                                             *
        !* Application programs call the following routines            *
        !*                                                             *
        ! FNinit%[(lb%)           Initialize DECtalk on lb:            *
        ! FNanswer%               Finish last call, answer next        *
        ! FNhangup%               Hangup the call                      &
        ! FNkey$[(timeout%)       Read a character with timeout        &
        !                         Returns the character, or            &
        !                           E       Error (from RSTS)          &
        !                           H       Phone hung up              *
        !                           T       Timeout                    *
        !                           X       Bad Escape sequence        *
        ! FNtimeout%(sec%)        Set specified timeout, 0 = none      &
        ! FNtest%(R2%, R9%)       Test current reply, true if ok       &
        !                           R3% is -1 to ignore it.            &
        !                           FNtest%() checks character,        &
        !                           intermediates, and finals.         *
        ! FNphtest%(R3%)          Test phone reply (R2% checked)       &
        ! FNsend%(text$)          Send text to DECtalk.                &
        ! FNspeak%(text$)         Send text followed by <CR><LF>       &
        ! FNlog%(text$)           Log text message                     &
        ! FNvisible$(char%)       Make character printable for         *
        !                           logging and debugging msgs.       &
        ! FNmessage%(text$, R2%, R3%)                                 &
        !                         Send DCS seq. test reply.            &
        !                           text is "P2;P3...",                &
        !                           return TRUE if ok.                 &
        !                           Note: FNmessage%() ignores         &
        !                           R3.PH.TIMEDUT replies.             &
        ! FNphone%(text$)         Send DCS sequence, test reply        &
        !                           text is "P3;P4..."                 &
        !                           R2% must be R2.PHONE%              &
        !                           R3% not tested                     &
        !                           FNfunny% called if error           &
        !                           returns as FNmessage%()            &
        ! FNfunny%(text$)         Print bad sequence on the log        &
        ! FNdump%(text$)          Dump the current reply               &
```

```
10010   !*;                                                             &
        !* The application program generally doesn't call the          &
        !* following routines.                                          &
        !*                                                             &
        ! FNsave%(char)          Save type-ahead character,            &
        !                          return TRUE if saved.               &
        ! FNmsg%(text$)          Send DECtalk DCS message.             &
        !                          text is "P2;P3...";                 &
        ! FNcsi%(text$)          Send DECtalk CSI message.             &
        !                          text has parm, inter, final.        &
        ! FNfromdectalk%(time%) Read key or escape sequence.           &
        ! FNgetseq%(time%)      Read key or escape sequence.           &
        ! FNget%(timeout%)      Read one character.                    &
        !                          parity is stripped.                 &
        !                          <NUL> and <DEL> are ignored.        &
        !                          Return 0% on timeout.               &
        !                          Other errors are fatal.             &
        !                        NOTE: do not use fnget%() to          &
        !                        read from the telephone keypad.       &
        ! FNread%(timeout%)     Read a record from DECtalk             &
        !                                                             &
        !*                                                            &
        !* Globals:                                                    &
        !*                                                            &
        ! @1%, R2%, R3% current reply parameters                       &
        !               set by FNgetsequence%()                       &
        ! DT.timeout%   TRUE if keypad timeouts are                   &
        !               currently enabled.                            &
        ! error.count%  Incremented on serious errors                 &
        ! ESC%          ESC character (partly but cleared)            &
        ! CAN%          CTRL-U character (cancel sequence)            &
        ! SUB%          CTRL-Z character                              &
        ! CSI%          CSI character                                 &
        ! DCS%          DCS character                                 &
        ! ST%           ST character                                  &
        ! ESC$          An escape to send chr$(155%)                  &
        ! CRLF$         Carriage-return, Line-feed                    &
        ! VT$           Vertical Tab (DECtalk flush)                  &
        ! R2.PHONE%       R2% phone reply                             &
        ! R3.PH.ONHOOK%   R3% (phone hung up)                         &
        ! R3.PH.OFFHOOK%  R3% (phone is alive)                        &
        ! R3.PH.TIMEOUT%  R3% (keypad timeout)                        &
        !                                                             &
        ! DT.anything    reserved for local buffers                   &
        ! SEQ.anything   reserved for sequence parser                 &
        ! qf%anything:   general temporaries                          &
        !                                                             &
```

```
10100   def* FNinit%(kb$)                                       &
        !                                                       &
        ! F N i n i t % ( k b $ )                               &
        !                                                       &
        ! Initialize the DECtalk device                         &
        ! Return TRUE% if ok, FALSE% if error                   &
        !                                                       &

10110   !*                                                      &
        !* Open the terminal in "binary" mode.                  &
        !* Then initialize all constants.                       &
        !*                                                      &
        open kb$ for input as file 8%, mode 32%+16%+4%+1%       &
        \ open kb$ for input as file 9%, mode 32%+16%+4%+1%     &
        \ DT.incount%, DT.inend% = 0%   ! Clear input buffer    &
        \ SEQ.state% = 0%               ! Clear input state     &
        \ DIM DT.p%(3), SEQ.p%(3)       ! 3 parameters          &
        \ DIM q%(256)                   ! For debugging         &
        \ TRUE% = (1% = 1%)             ! TRUE                  &
        \ FALSE% = not TRUE%            ! FALSE                 &
        \ ESC% = 27%                    ! Escape                &
        \ ESC$ = chr$(ESC% + 128%)      ! Define escape char    &
        \ VT$ = chr$(ascii('K') - 64%)  ! DECtalk flush char    &
        \ CRLF$ = chr$(13%) + chr$(10%) ! <CR><LF> string       &
        \ CAN% = ascii('U') - 64%       ! CANcel (CTRL-U)       &
        \ SUB% = ascii('Z') - 64%       ! SUBstitute (CTRL-Z)   &
        \ CSI% = ascii('[') - 64% + 128%        ! Define        &
        \ DCS% = ascii('P') - 64% + 128%        ! C1 control    &
        \ ST%  = ascii('\') - 64% + 128%        ! characters    &
        \ R2.PHONE% = 70%                                       &
        \ R3.PH.ONHOOK% = 0%                                    &
        \ R3.PH.OFFHOOK% = 1%                                   &
        \ R3.PH.TIMEOUT% = 2%                                   &

10120   q% = FNsend%(chr$(ascii('G') - 64% + 128%)) ! CTRL-G    &
        \ q% = FNget%(2%) while (q% > 0%) ! Drain text          &
        \ q% = FNdcs%("38")             ! No local echoback     &
        \ q% = FNcsi%("c")              ! "Who are you"         &
        \ q% = FNfromdectalk%(5%)       ! Read escape sequence  &
        \ IF (DT.char%    <> CSI%       ! Check                 &
           or DT.final$   <> "c"        !   for                 &
           or DT.private% <> "?"        !     DECtalk           &
           or R1%         <> 10%) then  !       reply           &
            q% = FNfunny%("initialization")                     &
            \ FNinit% = FALSE%          ! Return failure        &
            \ goto 10190                ! from FNinit%()        &

10130   q% = FNsend%(ESC$ + "!p")       ! Soft Terminal Reset   &
        \ q% = FNdcs%("30;1")           ! Set MODE SQUARE       &
        \ DT.timeout% = 0%              ! No timeouts now       &
        \ FNinit% = TRUE%               ! Return TRUE           &

10190   fnend                                                   &
```

```
10200   def* FNanswer%                                          &
        !                                                       &
        !       F N a n s w e r %                               &
        !                                                       &
        ! Finish off any current call (hanging up the phone)    &
        ! then setup and answer the next call.                  &
        ! Return TRUE% if the call was answered.                &
        ! Return FALSE% if there's serious problems.            &
        !                                                       &

10210   FNanswer% = FALSE%                      ! Assume error  &
        \ q% = FNget%(2%) while (q% > 0%)       ! Drain text    &
        \ goto 10290 if (not FNphone%(""))      ! poll status   &
        \ if (R3% = R3.PH.OFFHOOK%) then        ! if alive,     &
          goto 10280 if (not FNhangup%)         ! hangup phone  &

10220   if (R3% <> R3.PH.OFFHOOK%) then          ! still alive?  &
            q% = FNfunny%("hangup/poll")        ! Urk.          &
            \ goto 10290                        ! exit this     &

10230   goto 10290 if (not FNphone%("80|"))     ! answer 1 ring &
        \ if (R3% <> R3.PH.OFFHOOK%) then       ! ok?           &
          : q% = FNfunny%("enable answer")      ! Urk.          &
            \ goto 10290                        ! exit this     &

10240   q% = FNfromdecmit%(0%)                  ! wait for ring &
        \ if (q% <> DCS%) then                  ! ok?           &
            q% = FNfunny%("waiting for ring")   ! oops.         &
            \ goto 10290                        ! exit this     &

10250   if (not FNpiset%(R3.PH.OFFHOOK%)) then                  &
          : q% = FNfunny%("expecting offhook")                  &
          : \ goto 10290                                        &

10280   DT.timeout% = 0%                ! No timeouts now       &
        \ DT.pending$ = ""              ! Nothing pending now   &
        \ FNanswer% = TRUE%             ! ok.                   &
        .
10290   fnend                                                   &

10300   def* FNtimeout%(seconds%)                               &
        !                                                       &
        !       F N t i m e o u t % ( s e c o n d s % )         &
        ! .                                                     &
        ! Enable or disable keypad timeout.  Note that         &
        ! FNtimeout%(non-zero%) will examine the state of the  &
        ! type-ahead buffer before actually enabling timeouts  &
        !                                                       &

10310   if (seconds% > 0%) then                                 &
          : seconds% = 0% if (len(DT.pending$) > 0%)            &
          : \ if (seconds% > 0%) then                           &
              q% = FNsync%              ! make sure all heard   &
              \ seconds% = 0% if (len(DT.pending$) > 0%)        &
          ! If the program requests that timeouts be turned     &
          ! on, perform some special checks that the user       &
          ! hasn't already entered any text (which would be     &
          ! stored in one of the type-ahead buffers.  If        &
          ! something is pending, turn timeouts off.  This is  &
          ! needed because RSTS allows a program to run even    &
          ! if all output has not been sent to the device.      &
```

```
10320   goto 10390 if fseconds% = DT.timeout%)  ! Don't resend  &
        \ print #23, "timeouts set "; seconds% if (DT.log%)     &
        \ q% = FNphone%("30;" + num1$(seconds%))                &

10330   DT.timeout% = seconds%          ! save timeout state    &
        \ if (not FNptest%(R3.PH.OFFHOOK%)) then                &
            q% = FNfunny%("timeout")                            &

10390   fnend                                                   &

10400   def* FNsync%                                            &
        !                                                       &
        !       F N s y n c %                                   &
        !                                                       &
        ! Synchronize with DECtalk.  This function returns     &
        ! when all text sent to DECtalk has been spoken.       &
        ! Warning: if you have sent much text to DECtalk and   &
        ! the modem is in the wrong phase, there is a very     &
        ! slight chance that this code could get an operating  &
        ! system timeout, even though there are no errors.     &
        !                                                       &

10410   q% = FNsend%(vTs)               ! Flush speech          &
        \ q% = FNdcs%("1")              ! Send sync             &
        \ if (not FNmessage%("32", 92, -1)) then                &
            q% = FNfunny%("sync")                               &

10490   fnend                                                   &

10500   def* FNhangup%                                          &
        !                                                       &
        !       F N h a n g u p %                               &
        !                                                       &
        ! Hangup the telephone.  Returns when the phone is     &
        ! properly on-hook (TRUE%) or an error is detected.    &
        !                                                       &

10510   FNhangup% = FALSE%              ! Assume problems       &
        \ goto 10590 if (not FNphone%("11"))    ! send hangup   &
        \ while (R3% = R3.PH.OFFHOOK%)  ! wait until            &
            \ sleep 5%                  ! it's hung up          &
            \ goto 10590 if (not FNphone%(""))                  &
        \ next                          ! loop forever          &
        \ FNhangup% = TRUE%             ! OK now.               &

10590   FNend                                                   &

10600   def* FNphone%(text$)                                    &
        !                                                       &
        !       F N p h o n e % ( t e x t $ )                   &
        !                                                       &
        ! Send a phone message, return the FNmessage% code.    &
        ! You should then call FNtest% to see just what the    &
        ! phone state actually is.                             &
        !                                                       &

10610   if (text$ <> "") then           ! If extra parameters   &
            text$ = "50;" + text$       ! tack them on, else    &
            else text$ = "50"           ! just do status report &
```

```
10620   FNphone% + FNmessage%(text$, R2.PHONE%, -1%)                    &

10690   fnend                                                           &

10700   def* FNsave%(char%)                                             &
        !                                                               &
        !       F N s a v e % ( c h a r % )                             &
        !                                                               &
        ! If the char% came from a user data entry, save it in         &
        ! the DT.pending$ buffer and return TRUE%, otherwise,          &
        ! return FALSE%.  Note that FNsave%() watches for              &
        ! asynchronous keypad timeouts.                                &
        !                                                               &
        ! Note that unreasonable amounts of type-ahead may              &
        ! cause the program to overflow memory.                        &
        !                                                               &

10710   FNsave% = TRUE%                                                 &
        \ if FNptest%(R3.PH.TIMEOUT%) then      ! Timeout?            &
          \ goto 10790 if (DT.timeout% = 0%)    ! Disabled?           &
          \ DT.timeout% = 0%                    ! None now            &
          \ goto 10790 if clen(DT.pending$) > 0%)                       &
          \ char% = ascii('T')                  ! Save 'T'            &

10720   \ if (instr(0%, "0123456789*#ABCDT", chr$(char%)) = 0%)         &
          \ then FNsave% = FALSE%                                       &
          \ else DT.pending$ = DT.pending$ + chr$(char%)                &

10790   fnend                                                           &

10800   def* FNkey%(timeout%)                                           &
        !                                                               &
        !       F N k e y % ( t i m e o u t % )                         &
        !                                                               &
        ! Read a keypad character (if there is one in the              &
        ! type-ahead buffer, or read a character or escape             &
        ! sequence from DECtalk.  The timeout% parameter is            &
        ! non-zero to enable timeouts.                                 &
        !                                                               &
        ! Note that the timeout parameter, if non-zero, will be        &
        ! extended to compensate for @STS/E output buffering.          &
        !                                                               &
        ! FNkey% ignores user timeout if timeout was disabled.         &
        !                                                               &

10810   q% = FNtimeout%(timeout%)       ! Set/clear timeouts            &
        \ if clen(DT.pending$) > 0% then                                &
            FNkey% = ascii(DT.pending$)                                 &
          \ DT.pending$ = right(DT.pending$, 2%)                        &
          \ goto 10890                                                  &

10820   timeout% = (timeout% * 1%) + 60% if timeout% > 0%               &
        \ q% = FNfromdectalk%(timeout%)                                 &
        \ c% = ascii("T")       if FNptest%(R3.PH.TIMEOUT%)             &
        \ DT.timeout% = 0%      if (q% = ascii("T"))                    &
        \ c% = ascii("H")       if FNptwat%(R3.PH.HHOOK%)               &
        \ q% = ascii("E")       if (q% <= 0%)   ! D.S. error            &
        \ FNkey% = q%                                                   &

10890   fnend                                                           &
```

```
12000   def* FNmessage%(text$, t2%, t3%)                        &
        !                                                       &
        ! F N m e s s a g e % ( t e x t $ ,   t 2 % ,   t 3 % ) &
        !                                                       &
        ! Send a DECtalk DCS sequence to DECtalk and wait      &
        ! for a reply.  Make sure the reply matches the t2%    &
        ! and t3% parameters.  Return TRUE% if ok, else FALSE%. &
        !                                                       &
        ! A keyped timeout (escape sequence) may be read when  &
        ! we are expecting some other reply.  In this case,    &
        ! the timeout is ignored, the timeout status  flag is  &
        ! set FALSE and we read another sequence.              &
        !                                                       &

12010   q% = FNdcs%(text$)              ! Send the sequence     &
        \ FNmessage% = TRUE%            ! Assume success        &

12020   q% = FNfromdectalk%(60%)        ! get something         &
        \ goto 12020 if (q% = ST%)      ! ignore string term.   &
        \ goto 12020 if (FNsave%(q%)    ! save type-ahead       &
        \ if not (FNtest%(t2%, t3%)) then       ! Check seq.    &
            q% = FNfunny%("message test error")                 &
            \ FNmessage% = FALSE%                               &

12090   fnend                                                   &

12100   def* FNfromdectalk%(timeout%)                           &
        !                                                       &
        !       F N f r o m d e c t a l k % ( t i m e o u t % ) &
        !                                                       &
        ! Read an escape sequence or keyped character.  Dump   &
        ! junk between DCS final and string terminator.        &
        !                                                       &

12110   if (SEQ.state% <> 0% and SEQ.state% <> ST%) then         &
            gosub 12200                 ! Grab the sequence     &
            \ goto 12160                ! And return char value &

12120   SEQ.state% = 0%                 ! Nothing pending now   &
        \ q% = FNgetsequence%(timeout%) ! Get something         &
        \ gosub 12200                   ! Make it current       &
        \ q% = FNtoss%  if (q% = DCS%)  ! Toss junk until ST    &

12160   FNfromdectalk% = DI.char%       ! Return character      &

12190   fnend                                                   &

12200   !                                                       &
        ! Subroutine called from FNfromdectalk% to copy the    &
        ! last escape sequence read into the "current sequence" &
        ! buffer.  This is needed to skip over junk between    &
        ! the DCS final and the string terminator.             &
        !                                                       &
```

```
12210   DT.char% = SEQ.char%                ! Sequence type          &
        \ DT.final$ = SEQ.final$            ! Sequence terminator    &
        \ DT.private% = SEQ.private%        ! private characters     &
        \ DT.inter% = SEQ.inter%            ! Intermediates          &
        \ DT.parm% = SEQ.parm%              ! Parameter count        &
        \ R1%, DT.p%(1%) = SEQ.p%(1%)       ! Param's                &
        \ R2%, DT.p%(2%) = SEQ.p%(2%)       !                        &
        \ R3%, DT.p%(3%) = SEQ.p%(3%)       !                        &
        ! \ print #2%, "dt.parm%) "parms: "; R1%; R2%; R3%           &

12220   return

12300   def* FNtoss%                                                 &
        !                                                            &
        !       F N t o s s %                                        &
        !                                                            &
        ! Called after reading a DCS, this function reads           &
        ! text to the terminating string terminator.                 &
        !                                                            &

12310   SEQ.state% = 0%                                              &

12320   q% = FNgetsequence%(5%)                                      &
        \ if (q% <= 0% or (q% >= 128% and q% <= 159%))               &
            then FNtoss% = q%                                        &
            else goto 12320                                          &

12390   fnend                                                        &

13000   def* FNgetsequence%(timeout%)                                &
        !                                                            &
        !       F N g e t s e q u e n c e % ( t i m e o u t % )      &
        !                                                            &
        ! Read the next character or the next ANSI standard          &
        ! Escape Sequence.                                           &
        !                                                            &
        ! Initialize by setting SEQ.state% to zero.  Returns:        &
        ! SEQ.status%             sequence final character           &
        !                                                            &
        ! Note the following goto's:                                 &
        ! goto 13010 to read the next character in a sequence.       &
        ! goto 13020 to continue processing (needed when escape      &
        !               followed by a second character turns        &
        !               into a C1 control character).                &
        ! goto 13120 to exit an ESC sequence                         &
        ! goto 13130 to exit after reading a DCS/CSI sequence.      &
        ! goto 13140 to exit a C0 control within a sequence.        &
        !                                                            &
        ! The following is set by this module:                       &
        ! SEQ.char%       the character or sequence type             &
        ! SEQ.final%      the sequence final for CSI/DCS/OSC         &
        ! SEQ.state%      zero when sequence ends.                   &
        ! SEQ.parm%       number of parameters                       &
        ! SEQ.p%()        each parameter as read                     &
        ! SEQ.inter%      intermediates                              &
        ! SEQ.private%    private introducer, "X" if error seen      &
        !                                                            &
```

```
13010   DT.c% = fnget%[:inmout%)          ! Get a character       &

13030   if (DT.c% = ESC%                  ! If the character      &
         or DT.c% = CSI%                  ! introduces a new      &
         or DT.c% = DCS%) then            ! sequence, initialize  &
            SEQ.state% = DT.c%            ! all work areas.       &
            ! \ print #2%, "seq start: "; fnvisible$(dt.c%)      &
            \ SEQ.inter$ = ""                                     &
            \ SEQ.private$ = ""                                   &
            \ SEQ.parm% = 0%                                      &
            \ SEQ.p%(1%), SEQ.p%(2%), SEQ.p%(3%) = 0%             &
            \ goto 13010                  ! go read another byte  &

13035   goto 13140 if (SEQ.state% = 0%) ! done if no sequence    &
        !                                                         &
        ! Continue processing the current sequence               &
        !                                                         &
        \ if ((DT.c% >= 128% and DT.c% < 160%)  ! C1 control      &
           or (DT.c% = CAN%)              ! or CTRL-U             &
           or (DT.c% = SUB%)) then        ! or CTRL-Z             &
            SEQ.state% = 0%               ! force sequence exit   &
            ! \ print #2%, "c0 control: "; fnvisible$(dt.c%)     &
            \ goto 13140                  ! and return C0 control &

13040   goto 13140 if (DT.c% < 32%)       ! Exit if C0 control    &
        !                                                         &
        ! Process C1 introducers, intermediates, parameters,      &
        ! sequence terminators and other strange stuff           &
        !                                                         &
        \ if (DT.c% < 48%) then           ! Intermediate          &
            SEQ.inter$ = SEQ.inter$ + chr$(DT.c%)                 &
            ! \ print #2%, "intermediate: "; fnvisible$(dt.c%)    &
            \ goto 13010                  ! Go get another        &

13050   if (SEQ.state% = ESC%) then       ! <ESC> -> C1 control?  &
            q% = DT.c% and 63%            ! Mask out lower 6 bits &
            \ goto 13130 if (SEQ.inter$ <> "" or q% >= 32%)       &
            \ DT.c% = q% + 128%           ! Make it a C1 control  &
            ! \ print #2%, "c0 -> c1: "; fnvisible$(dt.c%)        &
            \ goto 13020                  ! Process C1 control    &

13060   goto 13120 if (DT.c% >= 64%)      ! Sequence terminator   &
        ! \ print #2%, "not terminator "; fnvisible$(dt.c%)       &
        \ goto 13080 if (DT.c% < 60%)     ! private introducer?   &
        ! \ print #2%, "private introducer "; fnvisible$(dt.c%)   &
        \ if (SEQ.parm% > 0%)             ! maybe, but illegal    &
            then SEQ.private$ = "X"       ! after first param.    &
            else SEQ.private$ = chr$(DT.c%)                       &
                \ SEQ.parm% = 1%          ! Mark "param"          &

13070       goto 13010                    ! Read another char.    &
```

```
13080   !                                                               &
        ! We know the character is in the range '0'..'9' or            &
        ! ';' (separator) or '!' (illegal separator)                   &
        !                                                               &

        SEQ.parm% = 1% if (SEQ.parm% = 0%)                              &
        ! \ print #2%, "param or sep: "; fnvisible$(dt.c%)             &
        \ if (SEQ.inter$ <> "") then    ! No param's after             &
            SEQ.inter$ = ""             ! intermediates.               &
            \ SEQ.private% = "X"        ! Mark it invalid.             &
            ! \ print #2%, "parm or separator after inter"             &

13090   if (DT.c% <= ascii('9')) then   ! Parameter digit               &
          SEQ.p%(SEQ.parm%) =           ! Make it a number             &
            (SEQ.p%(SEQ.parm%) * 10%) + (DT.c% - ascii('0'))            &
          ! \ print #2%, "digit, param :="; seq.p%(SEQ.parm%)          &
          \ goto 13010                  ! Go read another byte         &

13100   if (DT.c% = ascii(';')) then    ! parameter separator          &
          SEQ.parm% = SEQ.parm% + 1%                                    &
          \ goto 13010                  ! and read another byte        &

13110   SEQ.private$ = "X"              ! ';' isn't a separator        &
        ! \ print #2%, "bad separator "; fnvisible$(dt.c%)             &
        \ goto 13010                    ! read another byte            &

13120   !                                                               &
        ! Character is a sequence terminator.  If no parameters       &
        ! were read, return a single zero-valued parameter.           &
        !                                                               &
        SEQ.parm% = 1% if (SEQ.parm% = 0%)                              &
        ! \ print #2%, "terminator: "; fnvisible$(dt.c%)                &

13130   !                                                               &
        ! Jump here at the end of the sequence.                        &
        !                                                               &
        SEQ.final$ = chr$(DT.c%)        ! Set the final                &
        \ DT.c% = SEQ.state%            ! Get return value             &
        \ SEQ.state% = 0%               ! Not in a sequence            &

13140   SEQ.char% = DT.c%               ! Character code               &

13180   FNgetsequence% = SEQ.char%      ! Return value                 &

13190   fnend                           ! That's all, folks            &

14100   def* FNsend%(text$)                                             &
        !                                                               &
        !       F N s e n d % ( t e x t $ )                             &
        !                                                               &
        !Send a string of text to DECtalk.  Note, the text             &
        !length must be less than the DECtalk terminal buffer         &
        !size.                                                          &
        !                                                               &
```

```
14110   field #9%, len(text$) as q$                                     &
        \ lset q$ = text$                                               &
        \ put #9%, record 4096%, count len(text$)                       &
        \ if (debug% and DT.log%) then                                  &
            print #2%, using "sent: ### ""; len(text$);                 &
            \ change text$ to q%                                        &
            \ print #2%, FNvisible$(q%(q%));                            &
                for q% = 1% to q%(0%)                                   &
            \ print #2%, """                                            &

14190   fnend                                                           &

14200   def* FNesc%(text$) = FNsend%(ESC$ + "[" + text$)                &
        !                                                               &
        !       F N e s c % ( t e x t $ )                               &
        !                                                               &
        ! Send a Control Sequence to DECtalk.                          &
        !                                                               &

14300   def* FNdcs%(text$) =                                            &
            FNsend%(ESC$ + "P0;" + text$ + "z" + ESC$ + "\")            &
        !                                                               &
        !       F N d c s % ( t e x t $ )                               &
        !                                                               &
        ! Send a DECtalk Device Control Sequence.                       &
        ! Note that the DECtalk P1 parameter, final, and               &
        ! string terminator are automatically included.                &
        !                                                               &

14400   def* FNspeak%(text$) = FNsend%(text$ + CRLF$)                   &
        !                                                               &
        !       F N l i n e % ( t e x t $ )                             &
        !                                                               &
        ! Send a line of text to DECtalk, followed by <CR><LF>          &
        !                                                               &

15000   def* FNget%(timeout%)                                           &
        !                                                               &
        !       F N g e t % ( t i m e o u t % )                         &
        !                                                               &
        ! Read the next character from DECtalk.                        &
        ! timeout% =    0%      means none                             &
        ! timeout% >    0%      wait timeout% seconds                  &
        ! timeout% =   -1%      return immediately if none             &
        ! return 0% on timeout, fatal exit on other errors             &
        ! The character is forced into the range 000 to 127            &
        ! and <NUL> (000) and <DEL> (127) are ignored                   &
        !                                                               &

15010   while (DT.incount% >= DT.inend%)        ! None saved?           &
            \ goto 15000 if (not FNread%(timeout%))                     &

15020   next                                                            &
```

```
15090   field #0%, DT.incount% as q$, 1% as q%  ! get char      &
        \ DT.incount% = DT.incount% + 1%        ! step index   &
        \ q% = ascii(q$) and %27%               ! drop parity  &
        \ goto 15110 if (q% = 0% or q% = 127%)  ! ignore nulls &
        \ FNget% = q%                           ! return char  &
        \ goto 15090                            ! exit         &

15080   FNget% = 0%                                            &

15090   fnend                                                  &

15100   def* FN-read%(timeout%)                                &
        !                                                      &
        !       F N r e a d % ( t i m e o u t % )              &
        !                                                      &
        ! Read a record from DECtalk.                          &
        ! timeout% =    0%      means none                     &
        ! timeout% >    0%      wait% (timeout% seconds        &
        ! timeout% =   -1%      return immediately if none     &
        ! return FALSE% on timeout, fatal exit on other errors &
        ! return TRUE% on success.                            &
        !                                                      &

15110   goto 15120 if DT.incount% < DT.inend%  ! Still stuff   &
        \ on error goto 15150                  ! grab error    &
        \ q$ = sys(chr$(3%) + chr$(9%))        ! no echo       &
           + sys(chr$(4%) + chr$(9%))          ! odt mode      &
        \ wait timeout% if timeout% > 0%       ! timeout       &
        \ get #0% if timeout% >= 0%            ! read buffer   &
        \ get #0%, record 8192% if timeout% < 0%               &
        \ DT.inend% = recount                  ! got it        &
        \ wait 0%                              ! no timeout    &
        \ DT.incount% = 0%                     ! clear index   &
        \ on error goto 19000                  ! common exit   &
        \ if (debug% and DT.log%) then                         &
            print #2%, using "read: ### ##", DT.inend%;        &
            \ field #0%, DT.inend% as q$                       &
            \ change q$ to q%                                  &
            \ print #2%, FNvisible$(q%(q%));                   &
                for q% = 1% to q%(0%)                          &
            \ print #2%, "~"                                   &

15120   FNread% = TRUE%                                        &
        \ goto 15190                                           &

15150   resume 15180                                           &
          if ((err = 15% and timeout% > 0%)                    &
          or (err = 13% and timeout% < 0%))                    &
        \ goto 19000                                           &

15180   FNread% = FALSE%                                       &

15190   fnend                                                  &
```

```
10050   def* FNtest%(t2%, t3%) =
            (DT.char% = DCS%)           ! Make
        and (DT.final$ = 'z')           !   sure
        and (len(DT.inter$) = 0%)       !     it's
        and (len(DT.private$) = 0%)     !       from
        and (R1% = 0%)                  !         DECtalk
        and (t2% = R2%)                 ! Check R2%
        and (t3% = R3% or t3% = -1%)    ! maybe check R3%
    !
    !       F N t e s t % ( t 2 % , t 3 % )
    !
    ! Return TRUE% if the current reply is a properly-
    ! formed DECtalk reply sequence whose R2% and R3%
    ! parameters match T2% and T3%.  T3% is ignored if
    ! it is -1%.
    !

16000   def* FNptest%(t3%) =            ! Test phone reply
        FNtest%(R2.PHONE%, t3%)
    !
    !       F N p t e s t % ( t 3 % )
    !
    ! Return TRUE% if the current reply R2% parameter
    ! is R2.PHONE% and the T3% matches %3%
    !

17000   def* FNfunny%(text$)
    !
    !       F N f u n n y % ( t e x t $ )
    !
    ! Log an error message and dump the current reply.
    !

17010   error.count% = error.count% + 1%
        \ if (DT.log%) then
            print #2%, if (ascii$(2%) <> 0%)
            \ print #2%, "Illegal reply at "; text$; "."
            \ FNfunny% = FNdump%("")

17090   fnend

17100   def* FNdump%(text$)
    !
    !       F N d u m p % ( t e x t $ )
    !
    ! Dump the current reply.
    !
```

```
17110   if (DT.log%) then                                               &
            print #2%, "Last sequence read";                            &
            \ print #2%, " at "; text$; if (text$ <> "")                &
            \ print #2%, ": ";                                          &
            \ if (DT.char% = 0%)                                        &
                then print #2%, "<TIMEOUT>"                             &
                else print #2%, FNvisible$(DT.char%);                   &
                    \ print #2%, DT.private$; DT.inter$;                &
                    \ for q% = 1% to DT.parm%                           &
                        \ print #2%, num1$(DT.p%(q%));                  &
                            if (DT.p%(q%) <> 0%)                        &
                        \ print #2%, ";";                               &
                            if ((q% + 1%) < DT.parm%)                   &
                    \ next q%                                           &
                    \ print #2%, DT.final$;                             &
                    \ print #2%, "<ST>"; if (DT.char% = DCS%)           &
                    \ print #2%                                         &

17180   fnend                                                           &

17200   def* FNvisible$(c%)                                             &
        !                                                               &
        !       F N v i s i b l e $ ( c % )                             &
        !                                                               &
        ! Return "datascope" version of c%                             &
        !                                                               &

17210   if (c% = ESC%)          then FNvisible$ = "<ESC>"               &
        else if (c% = DCS%)     then FNvisible$ = "<DCS>"               &
        else if (c% = CSI%)     then FNvisible$ = "<CSI>"               &
        else if (c% = ST%)      then FNvisible$ = "<ST>"                &
        else if (c% = TD%)      then FNvisible$ = CRLF$                 &
        else if (c% = VT%)      then FNvisible$ = "<VT>"                &
        else if (c% = 13%)      then FNvisible$ = ""                    &
        else q.vis% = (c% >= 127% or c% < 32%)                          &
            \ q$ = ""                                                   &
            \ q$ = "<~" if (c% >= 128%)                                 &
            \ q$ = "<" if (c% <  32%)                                   &
            \ c% = c% and 127%                                          &
            \ q$ = q$ + "^" + chr$(c% + 64%) if (c% < 32%)              &
            \ q$ = q$ + chr$(c%) if (c% >= 32%)                         &
            \ q$ = q$ + ">" if q.vis%                                   &
            \ FNvisible$ = q$                                           &

17290   fnend                                                           &

17300   def* FNlog%(text$)                                              &

        !                                                               &
        ! F N l o g %  ( t e x t $ )                                   &
        !                                                               &
        ! Log a text message                                           &
        !                                                               &

17310   if (DT.log%) then                                               &
            print #2%, date$(0%); " "; time$(0%); " "; text$            &
```

```
17390   fnend                                                           &

17400   def* FNyesno%(prompt$, default$)                                &
        !                                                               &
        ! F N y e s n o % ( p r o m p t $ , d e f a u l t $ )           &
        !                                                               &
        ! Prompt and get a yes/no answer                               &
        !                                                               &

17410   q% = 0%                                                         &
        \ until (q% = 1% or q% = 5%)                                    &
            \ q$ = FNprompt$(prompt$ + "(Yes/No)", default$)            &
            \ q% = instr(1%, "YES NO", cvt$%(q$, -1%))                  &
        \ next                                                          &
        \ fnyesno% = (q% = 1%)                                          &

17490   fnend                                                           &

17500   def* FNprompt$(prompt$, default$)                               &
        !                                                               &
        ! F N p r o m p t $ ( p r o m p t $ , d e f a u l t $ )       &
        !                                                               &
        ! Prompt and get a response                                    &
        !                                                               &

17510   print #1%, prompt$; " <"; default$; ">? ";                      &
        \ input line #1%, q$                                            &
        \ FNprompt$ = cvt$$(q$, 32%)                                    &
        \ FNprompt$ = cvt$$(default$, 32%) if len(q$) = 0%              &

17590   fnend                                                           &

19000   !                                                               &
        !       F a t a l   E r r o r   T r a p                         &
        !                                                               &

19010   error% = err                    ! save error number             &
        \ error.line% = erl             ! and error line                &
        \ resume 19100                  ! and take fatal exit           &

19100   print                           ! force new line                &
        \ print "Fatal Error ";         ! print error message           &
            cvt$%(right(sys(chr$(6%) + chr$(9%)                         &
                + chr$(error%)), 3%), 4%);                              &
            " at line "; error.line%    ! and line number               &

19180   stop                                                            &

19190   goto 32767

32767   end
```

DECtalk ESCAPE SEQUENCES A

This appendix summarizes the escape sequences (and their parameters) described in this manual. The following tables list escape sequence mnemonics and their ASCII representations.

You can verify each ASCII character by checking the decimal value that appears below the character.

Table A-1 Escape Commands

Mnemonic	Escape Sequence Decimal Value
DA Primary	**ESC [0 c** 027 091 048 099
	Request DECtalk to identify itself. See Table A-2 for reply.
DECAC1	**ESC SP 7** 027 032 055
	Select 8-bit C1 control character reception (accept the high-order bit).
DECSTR	**ESC [! p** 027 091 033 112
	Reset to power-up state.
DECTC1	**ESC SP 6** 027 032 054
	Select 7-bit C1 control character reception (truncate the high-order bit).
DECTST	**ESC [5 ; Pn y** 027 091 053 059 ... 121
	Initiate local self-tests. See Table A-6 for Pn parameters.
DSR Brief	**ESC [5 n** 027 091 053 110
	Give a brief status report. See Table A-2 for replies.
DSR Extended	**ESC [n** 027 091 110
	Give an extended device status report. See Table A-2 for replies.
DT_DICT	**ESC P 0 ; 4 0 ; z name sub ESC ** 027 080 048 059 052 048 122 ... 027 092
	Load user dictionary. See Table A-2 for replies.

Table A-1 Escape Commands (Cont)

Mnemonic	Escape Sequence Decimal Value
DT_INDEX	ESC P 0 ; 2 0 ; P2 z ESC \ 027 080 048 059 050 048 059 *** 122 027 092 Insert index flag in text. P2 range is 0 to 32767 (sent as ASCII characters).
DT_INDEX_QUERY	ESC P 0 ; 2 1 z ESC \ 027 080 048 059 050 049 122 027 092 Request DECtalk to return last index marker spoken. See Table A-2 for reply.
DT_INDEX_REPLY	ESC P 0 ; 2 1 ; P3 z ESC \ 027 080 048 059 050 049 059 *** 122 027 092 Insert index flag in text and inform host. P3 range is same as for DT_INDEX. See Table A-2 for reply.
DT_LOG	ESC P 0 ; 5 1 ; P3 z ESC \ 027 080 048 059 053 049 059 *** 122 027 092 Set trace and debugging log functions. See Table A-7 for P3 parameters.
DT_MASK	ESC P 0 ; 5 3 ; P3 z ESC \ 027 080 048 059 053 051 059 *** 122 027 092 Controls how DECtalk sends escape sequences and keypad characters to the host. See Table A-8 for P3 parameters.
DT_MODE	ESC P 0 ; 5 0 ; P3 z ESC \ 027 080 048 059 053 048 059 *** 122 027 092 Set DECtalk mode. See Table A-3 for P3 parameters.
DT_PHONE	ESC P 0 ; 5 0 Pn ; Po z text ESC \ 027 080 048 059 053 048 059 *** 059 *** 122... 027 092 Control attached telephone and telephone keypad interface. See Table A-5 for Po parameters.

Table A-1 Escape Commands (Cont)

Mnemonic	Escape Sequence Decimal Value
DT_PHONTEXT	**ESC P 0 ; 0 z** text **ESC ** 027 080 048 059 048 122 ^^ 027 092 Speak phonemic text.
DT_SPEAK	**ESC P 0 ; 1 2 ; P3 z ESC ** 027 080 048 059 049 050 059 ^^ 122 027 092 Enable (P3=1) or disable (P3=0) speaking.
DT_STOP	**ESC P 0 ; 1 0 z ESC ** 027 080 048 059 049 048 122 027 092 Stop speaking and dump any pending unspoken text.
DT_SYNC	**ESC P 0 ; 1 1 z ESC ** 027 080 048 059 049 049 122 027 092 Finish speaking current text before processing next command.
DT_TERMINAL	**ESC P 0 ; 2 ; P3 z ESC ** 027 080 048 059 050 059 ^^ 122 027 092 Set local terminal mode. See Table A-4 for P3 parameters.
DECID	**ESC Z** 027 090 Obsolete identify terminal request. Not recommended.
RIS	**ESC c** 027 099 Reset to power-up state.
S7C1T	**ESC SP F** 027 040 070 Select 7-bit C1 control character transmission.

Table A-1 Escape Commands (Cont)

Mnemonic	Escape Sequence Decimal Value
SSCTC	ESC SP G 027 040 071 Select 8-bit C1 control character transmission.

Table A-2 DECtalk Status Replies

Mnemonic	Escape Sequence Decimal Value
DT_DICT Reply	ESC P 0 $ 5 0 ; P3 z ESC \ 027 080 048 036 053 048 059 ??? 122 027 092 P3 characters are as follows: 0 No dictionaries entered correctly. 048 1 No room in dictionary. 049 2 Entry too long. 050
DA Primary Reply	ESC [? 1 ; 0 c 027 091 063 049 059 048 099 Reply from DECtalk to DA primary sequence.
DECT Baud Replies	ESC [0 n 027 091 048 110 No malfunctions. ESC [3 n 027 091 051 110 Malfunction, reprimand.

Table A-2 DECtalk Status Replies (Cont)

Mnemonic	Escape Sequence Decimal Value										
DSR Extended Replies	ESC	[	0	n	ESC	[	?	2	1	p	
	027	091	048	110	027	091	063	050	049	110	
	No malfunctions, first reply.										
	ESC	[	0	n	ESC	[	?	2	0	n	
	027	091	048	110	027	091	063	050	048	110	
	No malfunctions, second or later reply.										
	ESC	[	3	n	ESC	[	?	Pn	;	...	Pn	n
	027	091	051	110	027	091	063		059	...		110
	Malfunction occurred. Pn parameter values are as follows:											
	2	2		Communication failure								
	050	050										
	2	3		Input buffer overflow								
	050	051										
	2	4		Last NVR operation failed								
	050	052										
	2	6		Error in phonemic transcription								
	050	054										
	2	6		Error in DECtalk private control sequence								
	050	054										
	2	7		Last DIGTST failed								
	050	055										
DT INDEX QUERY Reply	ESC	P	0	;	3	2	;	P3	z	ESC	\	
	027	080	048	059	051	050	059	...	122	027	092	
	P3 is the ASCII value of the last index spoken.											
DT INDEX REPLY Reply	ESC	P	0	;	3	1	;	P3	z	ESC	\	
	027	080	048	059	051	049	059	...	122	027	092	
	Reply sent by DECtalk after speaking index/all text. P3 is the ASCII value of the last index spoken.											

Table A-3 DT_MODE Parameters

Mnemonic	Value	Function
MODE_SQUARE	1	Set MODE SQUARE on.
MODE_ASKY	2	Use the single-character phonemic alphabet.
MODE_MINUS	4	Speak hyphen as "minus".

Table A-4 DT_TERMINAL Parameters

Mnemonic	Value	Function
TERM_HOST	1	Send all local terminal characters to host line.
TERM_SPEAK	2	Speak all characters typed on local terminal.
TERM_EDITED	4	Line edit all local terminal input.
TERM_HARD	8	Do local terminal editing in hardcopy format.
TERM_SETUP	16	Speak all characters when terminal is in set-up mode.
TERM_FILTER	32	Do not send DECtalk-specific escape sequences to the terminal.

Table A-5 DT_PHONE Parameters

Mnemonic	ASCII Code Decimal Value		Function
PH_STATUS	0 048		Send a telephone status report.
PH_ANSWER	1 049	0 048	Enable telephone autoanswer.
PH_HANGUP	1 049	1 049	Hang up the telephone and disable the keypad.
PH_KEYPAD	2 050	0 048	Enable the telephone keypad.
PH_NOKEYPAD	2 050	1 049	Disable the telephone keypad.
PH_TIMEOUT	3 051	0 048	Send telephone status message after n seconds (Chapter 4).
PH_TONE_DIAL	4 052	0 048	Dial the telephone by using Touch-Tone dialing.
PH_PULSE_DIAL	4 052	1 049	Dial the telephone by using pulse dialing.

Table A-6 DECTST Parameters

Mnemonic	ASCII Code Decimal Value	Function
TEST_POWER	1 049	Rerun power-up tests.
TEST_HDATA	2 050	Run host port data loopback test.
TEST_HCONTROL	3 051	Run host port control loopback test.
TEST_LDATA	4 052	Run local port data loopback test.
TEST_SPEAK	5 053	Speak a built-in message.

Table A-7 DT_LOG Parameters

Mnemonic	Value	Function
LOG_TEXT	1	Speak all ASCII text.
LOG_PHONEME	2	Log all spoken text in phoneme format.
LOG_RAWHOST	4	Log all text read from the host on the local terminal.
LOG_INHOST	8	Log all text read from the host on the local terminal.
LOG_OUTHOST	16	Log all text sent to the host on the local terminal.
LOG_ERROR	32	Log all error messages on the terminal.
LOG_TRACE	64	Log all escape sequences as readable text on the local terminal.
LOG_DEBUG	128	Reserved for DECtalk internal use.

Table A-8 DT_MASK Parameters

Value	Bit	Character
1	0	0
2	1	1
4	2	2
8	3	3
16	4	4
32	5	5
64	6	6
128	7	7
256	8	8
512	9	9
1024	10	:
2048	11	#
4096	12	A
8192	13	a
16384	14	d
32768	15	D

PHONEMIC ALPHABET B

This appendix summarizes the phonemic symbols that DECtalk uses. DECtalk recognizes all 17 vowel phonemes and 24 consonant phonemes in the English language (Table B-1).

DECtalk uses two-character symbols for each English phoneme. DECtalk also recognizes a one-character system of representing phonemes. Use of the one-character system is discouraged, as it is not in wide use and may not be supported on future releases of DECtalk. However, DECtalk can be set to the one-character system. Refer to Chapter 4 of the *DECtalk DTC01 Owner's Manual*.

Table B-2 lists emphasis characters, for adding stress and suggesting proper phrasing (syntax).

Table B-1 Phonemic Inventory

2-Char. Symbol	1-Char. Symbol	Example	2-Char Symbol	1-Char. Symbol	Example
Vowels					
ey	e	bake	ah	^	but
ae	a	Bob	aw	W	bough
iy	i	beat	ya	Y	cute
eh	E	bet	rr	R	bird
ay	A	bite	ao	c	bought
ih	I	bit	se	@	bat
oy	O	boy	oh	U	book
ow	o	boat	ix	1	kissies
uw	u	boot	ax	x	about
ir		beer	er		bear
ar		bar	or		bore
ur		poor			
Consonants					
p	p	pet	b	b	bet
t	t	test	d	d	debt
k	k	key	g	g	guess
f	f	fin	v	v	vest
th	T	thin	dh	D	this
s	s	sit	z	z	zoo
sh	S	shin	zh	Z	azure
ch	C	coin	jh	J	gin
w	w	wet	hh	hh	met
yx	y	yet	rl	rr	met
hx	h	head	nx	G	sing
l	l	let	en	N	button
r	r	red			silence
el	L	bottle			
em	M	ransom			
Allophones (Override DECtalk internal values.)					
rx	r	proton		postvocalic r	
lx	l	electric		postvocalic l	
q	q	we eat		glottal stop (w'iy qx.iyt)	
dx	R	rider		flapped d	
tx	Q	kitten		glottalized t	

Table B-2 Phonemic Emphasis Markers

2-Character Symbol	1-Character Symbol	Meaning
Stress and Syntax Phonemic Symbols		
		Primary lexical stress
		Secondary lexical stress
		Emphatic stress
/`	/`	Pitch rise
\`	\`	Pitch fall
/^	/^	Pitch rise/fall
Syntactic Structure Symbols		
	-	Syllable boundary (dash)
	+	Morpheme boundary
#	#	Compound noun boundary
		Word boundary (space)
(`	(`	Beginning of relative clause
)`	)`	End relative clause, begin next phrase
		End of clause
		End of sentence
?`	?`	End of question
	!`	End of exclamation
¶	¶	Paragraph introducer

DOCUMENTATION C

RELATED DOCUMENTATION

You can order the following DECtalk documents from Digital.

Title	Description
DECtalk DTC01 Owner's Manual (EK-DTC01-OM)	This manual gives an overview of DECtalk operations and a detailed description of DECtalk off-line (local) operations, phonemic codes, and spoken text conventions.
DECtalk DTC01 Programmer Reference Manual (EK-DTC01-RM)	This manual describes DECtalk-computer connections, DECtalk escape sequences, and programming methods for interfacing DECtalk with a host computer and telephone.
DECtalk DTC01 Programmer Reference Card (EK-DTC01-RC)	This card summarizes DECtalk phonemic codes, commands, and escape sequences.
DECtalk DTC01 Installation Manual (EK-DTC01-IN)	This manual explains how to install and operate DECtalk.

ORDERING INFORMATION

You can obtain ordering information by telephone from 8:30 a.m. to 6:00 p.m. Eastern Standard Time (EST) or by mail.

By phone

Continental U.S.A. and Puerto Rico

1-800-258-1710

New Hampshire, Alaska, Hawaii

1-603-884-6660

By mail

In **the U.S.A. and Puerto Rico**

Digital Equipment Corporation
PO Box CS2008
Nashua, New Hampshire 03061

Outside the U.S.A. and Puerto Rico

Digital Equipment Corporation
Attn: Accessories and Supplies Business Manager
c/o Local Subsidiary or Digital-Approved Distributor

INDEX

C