DECntalk DTC01

Programmer Reference Manual

Prepared by Educational Services of Digital Equipment Corporation
The reproduction of this material, in part or whole, is strictly prohibited. For copy information, contact the Educational Services Department, Digital Equipment Corporation, Maynard, Massachusetts 01754.

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.
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-OT-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 equip-
ment. This certification means that the equipment meets certain telecommuni-
cations network protective, operational, and safety requirements. The depart-
ment 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 com-
pany’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 degra-
dation 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 connec-
tions 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

INTRODUCTION

CHAPTER 1  HOW DECTalk PROGRAMMING WORKS

Communicating with DECTalk .................................................. 2
Types of Data ............................................................................. 2
Operating Modes ....................................................................... 4
Using Escape Sequences and Control Characters ....................... 4
  Escape Sequences ..................................................................... 4
  Escape Sequence Format ......................................................... 6
Control Characters ..................................................................... 7
Control Character Logging ......................................................... 9
Effect of the Backspace (BS) Character ..................................... 9
DECTalk – Computer Communication ......................................... 10
  DECTalk Setups ........................................................................ 12
  Program Control ....................................................................... 13
Data Synchronization .................................................................. 13
DECTalk – Host Program Sequence ............................................. 15
Developing Your Application ..................................................... 15
  Names, Part Numbers, and Alphanumeric Text ......................... 16
Direct Numeric Encoding .......................................................... 16
Two-Character Encoding ........................................................... 17
Ending Commands and Data ..................................................... 17
Application Development Tips .................................................. 18
CHAPTER 2  SETUP ESCAPE SEQUENCES

Selecting ASCII Character Sets .................................................. 20
Coding Standards ................................................................... 22
Code Table ............................................................................... 23
    7-Bit ASCII Code Table .................................................. 23
    8-Bit Code Table ............................................................ 25
Character Sets ........................................................................... 27
    Selecting Alternate Character Sets (G0 – G3) ............... 27
DEC Multinational Character Set .............................................. 30
Working with 7-Bit and 8-Bit Environments ......................... 32
    Conventions for Codes Transmitted to the Terminal ....... 32
    Mode Selection (DT__MODE) .......................................... 32

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

Speech Control ........................................................................... 35
Speech Timeout .......................................................................... 36
English Text .............................................................................. 36
Speak Phonemic Text (DT__PHOTEXT) .................................. 37
Stop Speaking (DT__STOP) ........................................................ 38
Data Synchronization (DT__SYNC) ........................................... 38
Enable or Disable Speaking (DT__SPEAK) ............................... 39
Indexing .................................................................................... 39
    Index Text (DT__INDEX) ............................................... 40
    Index Reply (DT__INDEX__REPLY) ............................... 40
    Index Query (DT__INDEX__QUERY) ............................ 41
Load Dictionary (DT__DICT) ....................................................... 43

CHAPTER 4  TELEPHONE COMMUNICATIONS

Telephone Management (DT__PHONE) ...................................... 46
    PH__ANSWER ..................................................................... 48
    PH__HANGUP .................................................................... 48
    PH__KEYPAD ..................................................................... 48
    PH__NOKEYPAD ............................................................ 48
    PH__TIMEOUT ................................................................. 48
PH__TONE__DIAL and PH__PULSE__DIAL ................................. 49
CHAPTER 5 MAINTENANCE AND DEBUGGING COMMANDS

Device Attribute Request .................................................. 51
   Device Attribute Request (DA Primary) .......................... 51
   Identify Terminal (DECID) ........................................... 52
Device Test and Status .................................................... 52
   DEConv Power-Up Status ............................................. 52
   Device Self-Test (DECTST) .......................................... 54
   Device Status Request (DSR) (Brief Report) ................. 55
   Device Status Request (DSR) (Extended Report) ............ 55
   Reset to Initial State (RIS) ...................................... 56
   Soft Terminal Reset (DECSTR) .................................... 57
   NVR Feature Settings (DECNVR) ................................... 58
Tracing and Debugging Commands ....................................... 58
   Local Log Control (DT_LOG) ...................................... 58
   LOG__TEXT ............................................................. 61
   LOG__PHONEME ...................................................... 61
   LOG__RAWHOST ..................................................... 61
   LOG__INHOST ....................................................... 61
   LOG__OUTHOST ..................................................... 61
   LOG__ERROR .......................................................... 61
   LOG__TRACE .......................................................... 61
Local Terminal Command (DT_TERMINAL) ............................ 62
Keypad Mask Command (DT_MASK) ..................................... 64
Determining Firmware Revision Level ............................... 67
   Phonemic Alphabet ............................................... 67

CHAPTER 6 C PROGRAM EXAMPLE

Program Language and Structure ....................................... 70
How the Program Works .................................................. 71
Variable Names and Definitions ...................................... 72
   Flags ................................................................. 72
   Error Codes ....................................................... 72
   DEConv-Specific Parameters ..................................... 73
   DEConv Commands ................................................ 73
   Telephone Control Parameters .................................. 74
   DEConv Replies ................................................... 75
   Self-Test Parameters ............................................. 76
   Logging Command Parameters ................................... 77
The Sequence Data Structure .......................................... 78
<table>
<thead>
<tr>
<th>Application Programs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DECTLK.H</td>
<td>80</td>
</tr>
<tr>
<td>DEMO.C</td>
<td>83</td>
</tr>
<tr>
<td>DTANSW.C</td>
<td>98</td>
</tr>
<tr>
<td>DTCLOS.C</td>
<td>100</td>
</tr>
<tr>
<td>DTCMD.C</td>
<td>101</td>
</tr>
<tr>
<td>DTDCHA.C</td>
<td>103</td>
</tr>
<tr>
<td>DTDCS.C</td>
<td>105</td>
</tr>
<tr>
<td>DTDIAL.C</td>
<td>106</td>
</tr>
<tr>
<td>DTDRAI.C</td>
<td>108</td>
</tr>
<tr>
<td>DTDUMP.C</td>
<td>111</td>
</tr>
<tr>
<td>DTEOL.C</td>
<td>113</td>
</tr>
<tr>
<td>DTGESC.C</td>
<td>115</td>
</tr>
<tr>
<td>DTGET.C</td>
<td>116</td>
</tr>
<tr>
<td>DTHANG.C</td>
<td>123</td>
</tr>
<tr>
<td>DTINIT.C</td>
<td>125</td>
</tr>
<tr>
<td>DTINKE.C</td>
<td>126</td>
</tr>
<tr>
<td>DTIOGE.C</td>
<td>128</td>
</tr>
<tr>
<td>DTIOPU.C</td>
<td>130</td>
</tr>
<tr>
<td>DTISKE.C</td>
<td>140</td>
</tr>
<tr>
<td>DTISTI.C</td>
<td>143</td>
</tr>
<tr>
<td>DTISVA.C</td>
<td>144</td>
</tr>
<tr>
<td>DTKEYP.C</td>
<td>145</td>
</tr>
<tr>
<td>DTMSG.C</td>
<td>146</td>
</tr>
<tr>
<td>DTOFFH.C</td>
<td>147</td>
</tr>
<tr>
<td>DTONHO.C</td>
<td>149</td>
</tr>
<tr>
<td>DTOPEN.C</td>
<td>150</td>
</tr>
<tr>
<td>DTPEEK.C</td>
<td>151</td>
</tr>
<tr>
<td>DTPESC.C</td>
<td>157</td>
</tr>
<tr>
<td>DTPHON.C</td>
<td>163</td>
</tr>
<tr>
<td>DTPTES.C</td>
<td>167</td>
</tr>
<tr>
<td>DTPUT.C</td>
<td>168</td>
</tr>
<tr>
<td>DTRACE.C</td>
<td>169</td>
</tr>
<tr>
<td>DTRAD.C</td>
<td>170</td>
</tr>
<tr>
<td>DTRERE.C</td>
<td>172</td>
</tr>
<tr>
<td>DTSAVE.C</td>
<td>173</td>
</tr>
<tr>
<td>DTSPLICE.C</td>
<td>175</td>
</tr>
<tr>
<td>DTST.C</td>
<td>177</td>
</tr>
<tr>
<td>DTSYNC.C</td>
<td>178</td>
</tr>
<tr>
<td>DTTALK.C</td>
<td>179</td>
</tr>
<tr>
<td>DTTEST.C</td>
<td>180</td>
</tr>
<tr>
<td>DTTHM.E.C</td>
<td>181</td>
</tr>
<tr>
<td>DTTONE.C</td>
<td>183</td>
</tr>
<tr>
<td>DTTRAP.C</td>
<td>185</td>
</tr>
<tr>
<td>DTVISI.C</td>
<td>188</td>
</tr>
<tr>
<td>HELLO.C</td>
<td>190</td>
</tr>
</tbody>
</table>
CHAPTER 7  BASIC-PLUS PROGRAM EXAMPLE

RSTS/E Systems ................................................................. 191

APPENDIX A  DECTalk ESCAPE SEQUENCES

APPENDIX B  PHONEMIC ALPHABET

APPENDIX C  DOCUMENTATION

FIGURES

1-1  DECTalk, Terminal, and Host Communications ................................. 3
1-2  Typical Escape Sequence Format .................................................. 5
1-3  Escape Sequence Representations ................................................. 6
1-4  DECTalk-Computer Program Interaction ....................................... 11
1-5  Synchronizing DECTalk and Host Communications ........................... 14
2-1  Mapping 7-Bit and 8-Bit Tables .................................................. 21
2-2  7-Bit ASCII Code Table .......................................................... 23
2-3  7-Bit Code ................................................................................. 24
2-4  8-Bit ASCII Code Table ............................................................ 25
2-5  8-Bit Code ................................................................................. 26
2-6  Loading 8-Bit Characters ............................................................. 28
2-7  Selecting Active Character Sets ..................................................... 28
2-8  DEC Multinational Character Set .................................................... 30
3-1  Using DT_SYNC and DT_INDEX_QUERY to Coordinate Communications ................................................. 42
4-1  Telephone Communications .......................................................... 50
5-1  Data Paths for Logging and Debugging ........................................... 60
5-2  Data Paths for Local Terminal Operations ...................................... 62
6-1  Calling Tree of DECTalk Application Program .................................. 70

TABLES

1-1  Control Characters and Host Communications ................................. 8
2-1  Selecting 7-Bit or 8-Bit Mode ....................................................... 21
2-2  Selecting the Active Character Set ................................................ 29
2-3  Selecting the Character Set ........................................................... 29
2-4  DT_MODE Parameters ................................................................... 33
4-1  DT_PHONE Parameters .................................................................. 47
4-2  Phone Status Reply Codes ............................................................ 47
5-1  Restoring DECTalk Operating Features .......................................... 53
5-2  DECTalk Actions Performed at Resets .......................................... 53
5-3  Self-Test Parameters ..................................................................... 54
5-4  DT_LOG Parameters ...................................................................... 59
5-5 DT__TERMINAL Parameters ........................................ 63
5-6 DT__MASK Parameters ............................................. 65
6-1 Application Program Modules .................................... 80
A-1 Escape Commands .................................................. 210
A-2 DECTalk Status Replies ........................................... 213
A-3 DT__MODE Parameters ............................................ 215
A-4 DT__TERMINAL Parameters ....................................... 215
A-5 DT__PHONE Parameters ........................................... 216
A-6 DECTST Parameters ................................................ 217
A-7 DT__LOG Parameters ............................................... 217
A-8 DT__MASK Parameters ............................................. 218
B-1 Phonemic Inventory ................................................. 220
B-2 Phonemic Emphasis Markers ...................................... 221
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.
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 \xSET LOG PHONEME ON
```

is sent from a computer as an escape sequence

```
ESC P 0 ; 8 1 ; 2 z ESC \n```

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.

![Diagram of Typical Escape Sequence Format]

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Pronounced as</th>
</tr>
</thead>
<tbody>
<tr>
<td>a BS _</td>
<td>a</td>
</tr>
<tr>
<td>_ BS a</td>
<td>a</td>
</tr>
<tr>
<td>a BS b</td>
<td>b</td>
</tr>
<tr>
<td>ab BS BS de</td>
<td>de</td>
</tr>
<tr>
<td>a BS &quot;</td>
<td>a</td>
</tr>
<tr>
<td>a BS &quot; BS _</td>
<td>a</td>
</tr>
</tbody>
</table>
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 database 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.
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 à 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>
<thead>
<tr>
<th>Mnemonic</th>
<th>Escape Sequence</th>
<th>Decimal Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transmit</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S7C1T</td>
<td>ESC SP F</td>
<td>027 032 070</td>
<td>Select 7-bit C1 character transmission.</td>
</tr>
<tr>
<td>S8C1T</td>
<td>ESC SP G</td>
<td>027 032 071</td>
<td>Select 8-bit C1 control character transmission.</td>
</tr>
<tr>
<td><strong>Receive</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECTC1</td>
<td>ESC SP 6</td>
<td>027 032 054</td>
<td>Select 7-bit character reception. The high-order (eighth) bit is ignored in all received C1 control characters.</td>
</tr>
<tr>
<td>DECAC1</td>
<td>ESC SP 7</td>
<td>027 032 055</td>
<td>Select 8-bit C1 control character reception. The high-order (eighth) bit is accepted in all received C1 control characters.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI X3.4 – 1977</td>
<td>American Code for Information Interchange (ASCII)</td>
</tr>
<tr>
<td>ISO 646 – 1977</td>
<td>7-Bit Coded Character Set for Information Processing Interchange</td>
</tr>
<tr>
<td>ANSI X3.41 – 1974</td>
<td>Code Extension Techniques for Use with the 7-Bit Coded Character Set of American National Code Information Interchange</td>
</tr>
<tr>
<td>ISO Draft International Standard 2022.2</td>
<td>7-Bit and 8-Bit Coded Character Sets – Code Extension Techniques</td>
</tr>
<tr>
<td>ANSI X3.64 – 1979</td>
<td>Additional Controls for Use with American National Standard for Information Interchange</td>
</tr>
<tr>
<td>ISO Draft International Standard 6429.2</td>
<td>Additional Control Functions for Character Imaging Devices</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>ROW</th>
<th>COLUMN</th>
<th>BITS b7 b6 b5</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>NUL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>DLE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>SOH</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>DC1 (XON)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>STX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>ETX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>DCS (XOFF)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>EOT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>ENQ</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>ACK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>BEL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>ETB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>BS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>CAN</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>HT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>EM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>LF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>ESC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>FF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>GS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>RS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>SI</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

KEY

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>OCTAL</th>
<th>DECIMAL</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC</td>
<td>33</td>
<td>27</td>
<td>18</td>
</tr>
</tbody>
</table>

Figure 2-2 7-Bit ASCII Code Table
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.

| COLUMN | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ROW    | NUL| DLE| SP |    |    |    |    |    |    | DCS| / |||    |    |    |    |
| 01     | SOH| DC1|    |    |    |    |    |    |    |    |    | PU1|    |    |    |
| 02     | STX| DC2|    |    |    |    |    |    |    |    |    |    |    | STS|    |
| 03     | ETX| DC3|    |    |    |    |    |    |    |    |    |    |    |    |    |
| 04     | EOT| DC4|    |    |    |    |    |    |    |    |    |    |    |    | IND|
| 05     | ENQ| NAK|    |    |    |    |    |    |    |    |    |    |    |    |    |
| 06     | ACK| SYN|    |    |    |    |    |    |    |    |    |    |    |    |    |
| 07     | BEL| ETB|    |    |    |    |    |    |    |    |    |    |    |    |    |
| 08     | BS | CAN|    |    |    |    |    |    |    |    |    |    |    |    | HTS|
| 09     | HT | EM |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 10     | LF | SUB|    |    |    |    |    |    |    |    |    |    |    |    |    |
| 11     | VT | ESC|    |    |    |    |    |    |    |    |    |    |    |    |    |
| 12     | FF | FS |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 13     | CR | GS |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 14     | SO | RS |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 15     | SI | US |    |    |    |    |    |    | DEL| SS3|    |    |    |    |    |

Figure 2-4  8-Bit ASCII Code Table
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, CO (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_7 (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_7 or multinational

Figure 2-6 shows how you can map the ASCII_7 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

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

* 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

<table>
<thead>
<tr>
<th>Character Set</th>
<th>G0</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII_G</td>
<td>ESC ( B</td>
<td>ESC ) B</td>
<td>ESC * B</td>
<td>ESC + B</td>
</tr>
<tr>
<td></td>
<td>027 040 066</td>
<td>027 041 066</td>
<td>027 042 066</td>
<td>027 043 066</td>
</tr>
<tr>
<td>Multinational</td>
<td>ESC ( &lt;</td>
<td>ESC ) &lt;</td>
<td>ESC * &lt;</td>
<td>ESC + &lt;</td>
</tr>
<tr>
<td></td>
<td>027 040 060</td>
<td>027 041 060</td>
<td>027 042 060</td>
<td>027 043 060</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>140</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>161</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td></td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>162</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>163</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>164</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>165</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>166</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>'</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>167</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>168</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>169</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>10</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>170</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>11</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>171</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>FS</td>
<td></td>
<td>12</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td>172</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>13</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>173</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>14</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>174</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>15</td>
<td>O</td>
<td>`</td>
<td>o</td>
<td>175</td>
</tr>
</tbody>
</table>

---

### C0 Codes (ASCII Graphics)

- **KEY**
  - **CHARACTER**
    - **ESC** 33
    - **OCTAL** 27
    - **DECIMAL** 18
    - **HEX**

---

**Figure 2-8** DEC Multinational Character Set (Left Half)
<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>gb6</td>
</tr>
<tr>
<td>200</td>
<td>201</td>
<td>202</td>
<td>203</td>
<td>IND</td>
<td>NEL</td>
<td>SSA</td>
<td>ESA</td>
<td>HTS</td>
</tr>
<tr>
<td>128</td>
<td>129</td>
<td>130</td>
<td>131</td>
<td>132</td>
<td>133</td>
<td>134</td>
<td>135</td>
<td>136</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
</tr>
<tr>
<td>DCS</td>
<td>PU1</td>
<td>PU2</td>
<td>STS</td>
<td>CCH</td>
<td>MW</td>
<td>SPA</td>
<td>EPA</td>
<td>X</td>
</tr>
<tr>
<td>220</td>
<td>221</td>
<td>222</td>
<td>223</td>
<td>224</td>
<td>225</td>
<td>226</td>
<td>227</td>
<td>228</td>
</tr>
<tr>
<td>144</td>
<td>145</td>
<td>146</td>
<td>147</td>
<td>148</td>
<td>149</td>
<td>150</td>
<td>151</td>
<td>152</td>
</tr>
<tr>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
</tr>
<tr>
<td>160</td>
<td>161</td>
<td>162</td>
<td>163</td>
<td>164</td>
<td>165</td>
<td>166</td>
<td>167</td>
<td>168</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
</tr>
<tr>
<td>A0</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>A5</td>
<td>A6</td>
<td>A7</td>
<td>A8</td>
</tr>
<tr>
<td>C0</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
<td>C6</td>
<td>C7</td>
<td>C8</td>
</tr>
<tr>
<td>192</td>
<td>193</td>
<td>194</td>
<td>195</td>
<td>196</td>
<td>197</td>
<td>198</td>
<td>199</td>
<td>200</td>
</tr>
<tr>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
</tr>
<tr>
<td>240</td>
<td>241</td>
<td>242</td>
<td>243</td>
<td>244</td>
<td>245</td>
<td>246</td>
<td>247</td>
<td>248</td>
</tr>
</tbody>
</table>

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. (Naive 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 \n027 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.

\[
\begin{align*}
\text{MODE\_SQUARE} & \quad = \quad 1 \\
\text{MODE\_MINUS} & \quad = \quad 4 \\
\text{Desired P3 value} & \quad = \quad 5
\end{align*}
\]

```
ESC P 0 ; 8 0 ; 5 z ESC \n027 080 048 059 056 048 059 053 122 027 092
```

<table>
<thead>
<tr>
<th>Table 2-4 DT_MODE Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mnemonic</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>MODE_SQUARE 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>MODE_ASKY 2</td>
</tr>
<tr>
<td>MODE_MINUS 4</td>
</tr>
</tbody>
</table>
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 sentence 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 sentences 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 \n027 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.

\[\text{ESC} \ P \ 0 \ ; \ 1 \ 0 \ z \ \text{ESC} \ \backslash\]

\[027 \ 080 \ 048 \ 059 \ 049 \ 048 \ 122 \ 027 \ 092\]

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.

\[\text{ESC} \ P \ 0 \ ; \ 1 \ 1 \ z \ \text{ESC} \ \backslash\]

\[027 \ 080 \ 048 \ 059 \ 049 \ 049 \ 122 \ 027 \ 092\]

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.

\[
\begin{array}{cccccccc}
\text{ESC} & \text{P} & 0 & ; & 2 & 0 & ; & \text{P3} & \text{z} & \text{ESC} \\
027 & 080 & 048 & 059 & 050 & 048 & 059 & *** & 122 & 027 & 092
\end{array}
\]

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.

\[
\begin{array}{cccccccc}
\text{ESC} & \text{P} & 0 & ; & 2 & 1 & ; & \text{P3} & \text{z} & \text{ESC} \\
027 & 080 & 048 & 059 & 050 & 049 & 059 & *** & 122 & 027 & 092
\end{array}
\]

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.

\[
\begin{array}{cccccccc}
\text{ESC} & \text{P} & 0 & ; & 3 & 1 & ; & \text{P3} & \text{z} & \text{ESC} \\
027 & 080 & 048 & 059 & 051 & 049 & 059 & *** & 122 & 027 & 092
\end{array}
\]

P3 has the original value specified in DT_INDEX_REPLY.
Index Query (DT_INDEX_QUERY)

DT_INDEX_QUERY requests DECltalk 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 \n027  080  048  059  050  050  122  027  092
```

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

```
ESC  P  0 ;  3  2 ;  P3  z  ESC \n027  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 \n027 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 \nESC P 0 ; 4 0 ; z ms. m'ihz ESC \nESC P 0 ; 4 0 ; z DEC d'ehk ESC \nESC P 0 ; 4 0 ; z dec d'iyeuhember ESC \nESC P 0 ; 4 0 ; z Goethe g'owth 'iy ESC \nESC P 0 ; 4 0 ; z GOSLOW :ra 120 ESC \n```

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 \n027 080 048 059 053 048 059 *** 122 027 092
```

P3 may have one of the following values.

- **0**
  - Word entered correctly.
  - 048

- **1**
  - No room in dictionary.
  - 049

- **2**
  - Entry too long (256 characters maximum).
  - 050
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 \n027 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 \n027 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  DT_PHONE Parameters

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>ASCII Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PH_STATUS</td>
<td>0 048</td>
<td>Send a telephone status report.</td>
</tr>
<tr>
<td>PH_ANSWER</td>
<td>1 0 049 048</td>
<td>Enable autoanswer of the telephone.</td>
</tr>
<tr>
<td>PH_HANGUP</td>
<td>1 1 049 049</td>
<td>Hang up the telephone and disable the keypad.</td>
</tr>
<tr>
<td>PH_KEYPAD</td>
<td>2 0 050 048</td>
<td>Enable the keypad and select direct keypad decoding.</td>
</tr>
<tr>
<td>PH_NOKEYPAD</td>
<td>2 1 050 049</td>
<td>Disable the keypad (without hanging up the telephone).</td>
</tr>
<tr>
<td>PH_TIMEOUT</td>
<td>3 0 051 048</td>
<td>Enable timeouts on telephone keypad input. Timeout equals P4 parameter value.</td>
</tr>
<tr>
<td>PH_TONE.Dial</td>
<td>4 0 052 048</td>
<td>Dial an outgoing phone call using Touch-Tones.</td>
</tr>
<tr>
<td>PH_PULSE.Dial</td>
<td>4 1 052 049</td>
<td>Dial an outgoing phone call using pulse dialing.</td>
</tr>
</tbody>
</table>

### Table 4-2  Phone Status Reply Codes

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>ASCII Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3_PH_ONHOOK</td>
<td>0 048</td>
<td>Telephone is on-hook (hung up).</td>
</tr>
<tr>
<td>R3_PH_OFFHOOK</td>
<td>1 049</td>
<td>Telephone is off-hook (active).</td>
</tr>
<tr>
<td>R3_PH_TIMEOUT</td>
<td>2 050</td>
<td>No response after TIMEOUT command.</td>
</tr>
<tr>
<td>R3_PH_TOOLONG</td>
<td>3 051</td>
<td>Number dialed is longer than 256 characters.</td>
</tr>
</tbody>
</table>
**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 DECSRT.

**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.
HOST ENABLES COMMUNICATION WITH DT__PHONE/PH__ANSWER SEQUENCE. 
DECtalk SENDS R3__PH__ONHOOK REPLY.

ENDERING CALL

CALLER DIALS IN. DECtalk SENDS R3__PH__OFFHOOK SEQUENCE.
HOST ENABLES 20 SECOND TIMEOUT AND BEGINS COMMUNICATION CONTROL.

CASE 1. USER HANGS UP. AFTER NO SIGNAL IS RECEIVED FOR 20 SECONDS,
DECtalk SENDS R3__PH__TIMEOUT TO HOST.

CASE 2. HOST TERMINATES SESSION AND SENDS PH__HANGUP.
DECtalk HANGS UP PHONE AND REPLIES WITH R3__PH__ONHOOK.

Figure 4-1 Telephone Communications
MAINTENANCE AND DEBUGGING COMMANDS

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)
DE Ctalk 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
```

DE Ctalk 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 DE Ctalk hardware self-tests. Another set of escape sequences forces DE Ctalk to return status reports. The following paragraphs describe these sequences.

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

1. Power-up (PUP) is the state that DE Ctalk 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 DE Ctalk to its power-up state.

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

Table 5-1 lists the DE Ctalk 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>
<thead>
<tr>
<th>Table 5-1</th>
<th>Restoring DECTalk Operating Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feature</strong></td>
<td><strong>Factory Default</strong></td>
</tr>
<tr>
<td>Local line speed</td>
<td>9600 baud</td>
</tr>
<tr>
<td>Local line format</td>
<td>No parity</td>
</tr>
<tr>
<td>Host line speed</td>
<td>1200 baud</td>
</tr>
<tr>
<td>Host line format</td>
<td>No parity</td>
</tr>
<tr>
<td>Host C1 transmit mode</td>
<td>7 bit</td>
</tr>
<tr>
<td>Host C1 receive mode</td>
<td>7 bit</td>
</tr>
<tr>
<td>Local log flags</td>
<td>0</td>
</tr>
<tr>
<td>Local terminal flags</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5-2</th>
<th>DECTalk Actions Performed at Resets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DECTalk Action</strong></td>
<td><strong>Performed By</strong></td>
</tr>
<tr>
<td>Implied DT__SPEAK.</td>
<td>PUP, RIS, DECSTR</td>
</tr>
<tr>
<td>Hang up telephone.</td>
<td>PUP, RIS, DECSTR</td>
</tr>
<tr>
<td>Delete user dictionary.</td>
<td>PUP, RIS</td>
</tr>
<tr>
<td>Flush all pending text.</td>
<td>PUP, RIS</td>
</tr>
<tr>
<td>Turn host speech on.</td>
<td>RIS, DECSTR</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Mnemonic</th>
<th>ASCII Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST_POWER</td>
<td>1</td>
<td>Rerun power-up tests.</td>
</tr>
<tr>
<td></td>
<td>049</td>
<td></td>
</tr>
<tr>
<td>TEST_HDATA</td>
<td>2</td>
<td>Run host port data loopback test.</td>
</tr>
<tr>
<td></td>
<td>050</td>
<td></td>
</tr>
<tr>
<td>TEST_HCONTROL</td>
<td>3</td>
<td>Run host port control loopback test.</td>
</tr>
<tr>
<td></td>
<td>051</td>
<td></td>
</tr>
<tr>
<td>TEST_DATA</td>
<td>4</td>
<td>Run local port data loopback test.</td>
</tr>
<tr>
<td></td>
<td>052</td>
<td></td>
</tr>
<tr>
<td>TEST_SPEAK</td>
<td>5</td>
<td>Speak a built-in message.</td>
</tr>
<tr>
<td></td>
<td>053</td>
<td></td>
</tr>
</tbody>
</table>
Device Status Request (DSR) (Brief Report)
The brief DSR escape sequence is as follows.

\[
\text{ESC} \ [ \ 5 \ n \\
027 \ 091 \ 053 \ 110
\]

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

\[
\text{ESC} \ [ \ 0 \ n \\
027 \ 091 \ 048 \ 110
\]

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

\[
\text{ESC} \ [ \ 3 \ n \\
027 \ 091 \ 051 \ 110
\]

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.

\[
\text{ESC} \ [ \ n \\
027 \ 091 \ 110
\]

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.

\[
\text{ESC} \ [ \ 0 \ n \ \text{ESC} \ [ \ ? \ 2 \ 1 \ n \\
027 \ 091 \ 048 \ 110 \ 027 \ 091 \ 063 \ 050 \ 049 \ 110
\]

For later requests, DECTalk replies with the following sequence.

\[
\text{ESC} \ [ \ 0 \ n \ \text{ESC} \ [ \ ? \ 2 \ 0 \ n \\
027 \ 091 \ 048 \ 110 \ 027 \ 091 \ 063 \ 050 \ 049 \ 110
\]
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 [ ! 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  !  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  \n027  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>
<thead>
<tr>
<th>Table 5-4</th>
<th>DT_LOG Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mnemonic</td>
<td>Value</td>
</tr>
<tr>
<td>LOGTEXT</td>
<td>1</td>
</tr>
<tr>
<td>LOG_PHONEME</td>
<td>2</td>
</tr>
<tr>
<td>LOG_RAWHOST</td>
<td>4</td>
</tr>
<tr>
<td>LOG_INHOST</td>
<td>8</td>
</tr>
<tr>
<td>LOG_OUTHOST</td>
<td>16</td>
</tr>
<tr>
<td>LOG_ERROR</td>
<td>32</td>
</tr>
<tr>
<td>LOG_TRACE</td>
<td>64</td>
</tr>
<tr>
<td>LOG_DEBUG</td>
<td>128</td>
</tr>
</tbody>
</table>
For example, assume you want to set LOG_TEXT and LOG_RAWHOST.

\[
\begin{align*}
\text{LOG\_TEXT} & = 1 \\
\text{LOG\_RAWHOST} & = 4 \\
\end{align*}
\]

Desired P3 value \(= 5\)

\[
\text{ESC P 0 ; 8 1 ; 5 z ESC } \backslash 027 080 048 059 056 049 059 053 122 027 092
\]

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 \n027 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.

\begin{align*}
\text{TERM_HOST} & = 1 \\
\text{TERM_EDITED} & = 4 \\
\end{align*}

Desired P3 value = 5

\begin{verbatim}
ESC P 0 ; 8 2 ; 5 z ESC \ 
027 080 048 059 056 050 059 053 122 027 092
\end{verbatim}

Table 5-5 lists the possible P3 values.

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

\textit{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 on to have DECtalk send text to the local terminal.
TERM_FILTER is useful when you use DE Ctalk as a link between a general-purpose operating system and an applications terminal. TERM_FILTER modifies the operation of LOG_RAWHOST to prevent sending DE Ctalk-specific escape sequences to the local terminal.

When you set TERM_FILTER, the following escape sequences usually processed by DE Ctalk 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 DE Ctalk 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 DE Ctalk sends escape sequences and keypad characters to the host. DT_Mask simplifies application development when DE Ctalk 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 \n027 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>
<thead>
<tr>
<th>Table 5-6 DT MASK Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>512</td>
</tr>
<tr>
<td>1024</td>
</tr>
<tr>
<td>2048</td>
</tr>
<tr>
<td>4096</td>
</tr>
<tr>
<td>8192</td>
</tr>
<tr>
<td>16384</td>
</tr>
<tr>
<td>32768</td>
</tr>
</tbody>
</table>
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 \n027 080 059 056 057 059 051 048 055 050 122 027 092
```


text_for_math_delimiter_start

$(3072 = 1024 + 2048)$

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 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 \n   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 DECtlk phonemic alphabet.
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 timeouts 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.

MODEASKY
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_BUFSIZE 
Size of the operating system input buffer.

OUT_BUFSIZE 
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_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 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>
<thead>
<tr>
<th>Module</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECTLK.H</td>
<td>This file must be included in all modules that use the DECTalk applications library. Contains common definitions (for example, escape sequence parameters).</td>
</tr>
<tr>
<td>DEMO.C</td>
<td>This is the main module of the program.</td>
</tr>
<tr>
<td>DTANSW.C</td>
<td>Hangs up the phone and answers on n rings.</td>
</tr>
<tr>
<td>DTCLOS.C</td>
<td>Closes the DECTalk channel and frees all buffers.</td>
</tr>
<tr>
<td>DTCMD.C</td>
<td>Sends a DCS command to the DECTalk terminal.</td>
</tr>
<tr>
<td>DTDCHA.C</td>
<td>Formats characters into a visible ASCII datascope format and writes the text to the indicated file.</td>
</tr>
<tr>
<td>DTDCS.C</td>
<td>Sends a DECTalk DCS control sequence.</td>
</tr>
<tr>
<td>DTDIAL.C</td>
<td>Dials the DECTalk telephone.</td>
</tr>
<tr>
<td>DTDRAI.C</td>
<td>Absorbs any type-ahead characters.</td>
</tr>
<tr>
<td>DTDUMP.C</td>
<td>Writes an escape sequence buffer to the standard output file (for debugging).</td>
</tr>
<tr>
<td>DTEOL.C</td>
<td>Writes an end of line to DECTalk.</td>
</tr>
<tr>
<td>DTGESC.C</td>
<td>Reads an escape sequence or keypad character.</td>
</tr>
<tr>
<td>Module</td>
<td>Brief Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DTGET.C</td>
<td>Reads a character from the DECTalk terminal line.</td>
</tr>
<tr>
<td>DTHANG.C</td>
<td>Hangs up the telephone connected to DECTalk.</td>
</tr>
<tr>
<td>DTINIT.C</td>
<td>Initializes the DECTalk terminal on the channel opened on dt.</td>
</tr>
<tr>
<td>DTINKE.C</td>
<td>Reads a telephone keypad button.</td>
</tr>
<tr>
<td>DTIOGE.C</td>
<td>Reads one character from the DECTalk terminal line.</td>
</tr>
<tr>
<td>DTIOPU.C</td>
<td>Either writes the output buffer contents to the DECTalk device or stores the character in a local buffer.</td>
</tr>
<tr>
<td>DTISKE.C</td>
<td>Indicates if the telephone user typed any characters.</td>
</tr>
<tr>
<td>DTISTI.C</td>
<td>Tests the result of a dtphone () message for keypad timeout.</td>
</tr>
<tr>
<td>DTISVA.C</td>
<td>Indicates if the argument character is one of 0123456789#*ABCD.</td>
</tr>
<tr>
<td>DTKEYP.C</td>
<td>Enables or disables the telephone keypad.</td>
</tr>
<tr>
<td>DTMSG.C</td>
<td>Sends a DECTalk DCS control sequence and reads a DCS reply.</td>
</tr>
<tr>
<td>DTOFFH.C</td>
<td>Tests the result of a dt_phone () message for OFFHOOK.</td>
</tr>
<tr>
<td>DTONHO.C</td>
<td>Tests the result of a dt_phone () message for ONHOOK.</td>
</tr>
<tr>
<td>DTOPEN.C</td>
<td>Initiates communications by performing operating system specific initializations.</td>
</tr>
<tr>
<td>DTPEEK.C</td>
<td>Tests if a character is pending from DECTalk.</td>
</tr>
<tr>
<td>DTPESC.C</td>
<td>Compiles an appropriate escape sequence from the parameter buffer.</td>
</tr>
<tr>
<td>DTPHON.C</td>
<td>Sends a DECTalk phone message.</td>
</tr>
<tr>
<td>DTPTES.C</td>
<td>Tests a phone reply.</td>
</tr>
<tr>
<td>Module</td>
<td>Brief Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DTPUT.C</td>
<td>Sends one character to the DECTalk terminal line. No value is returned.</td>
</tr>
<tr>
<td>DTREAD.C</td>
<td>Reads a sequence or character.</td>
</tr>
<tr>
<td>DTRESE.C</td>
<td>Sends a soft-reset escape sequence.</td>
</tr>
<tr>
<td>DTSAVE.C</td>
<td>Saves user type-ahead characters.</td>
</tr>
<tr>
<td>DTSPLICE.C</td>
<td>Lets you control a terminal connected to DECTalk’s local port.</td>
</tr>
<tr>
<td>DTST.C</td>
<td>Sends a string terminator (for phonemic text and telephone dial commands) to DECTalk.</td>
</tr>
<tr>
<td>DTSYNC.C</td>
<td>Synchronizes DECTalk and the application.</td>
</tr>
<tr>
<td>DTTALK.C</td>
<td>Speaks one line of text.</td>
</tr>
<tr>
<td>DTTEST.C</td>
<td>Tests a DECTalk reply.</td>
</tr>
<tr>
<td>DTTIME.C</td>
<td>Enables or disables a telephone keypad timeout.</td>
</tr>
<tr>
<td>DTTONE.C</td>
<td>Sends the msg text string as a tone dialing sequence.</td>
</tr>
<tr>
<td>DTTRAP.C</td>
<td>Traps CTRL-C interrupts.</td>
</tr>
<tr>
<td>DTVISI.C</td>
<td>Generates visible ASCII character representations.</td>
</tr>
<tr>
<td>HELLO.C</td>
<td>Tests that DECTalk is operating correctly.</td>
</tr>
</tbody>
</table>
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.

/*
 * Definitions and Globals
 *
 * 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
#ifdef BSD_42
#endif
#define UNIX_V
#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 intercepted 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_ptest(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 CO control characters are defined.

NUL  STX  ETX  BEL  BS  VT  LS1
LS0  XON  XOFF  CAN  SUB  ESC  DEL

The following C1 control characters are defined.

SS2  SS3  DCS  OLDID  CSI  ST  OSC
PM  APC  RDEL

The following DECTalk-specific parameters are also defined.

CSI_DA_PRODUCT The DECTalk product identification code.

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

P1_DECTALK    All DCT01 DCS sequences transmit this for their first (P1) parameter.

R1_DECTALK    All DCT01 DCS replies transmit this for the first R1 reply parameter.
The P2 and P3 parameters select the specific DECTalk command.

<table>
<thead>
<tr>
<th>P2 parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2_PHRTEXT</td>
<td>Speak phonemic text.</td>
</tr>
<tr>
<td>P2_STOP</td>
<td>Stop speaking.</td>
</tr>
<tr>
<td>P2_SYNC</td>
<td>Synchronize.</td>
</tr>
<tr>
<td>P2_SPEAK</td>
<td>Enable/disable speech.</td>
</tr>
<tr>
<td>P2_INDEX</td>
<td>Index text.</td>
</tr>
<tr>
<td>P2_IX_REPLY</td>
<td>Index with reply.</td>
</tr>
<tr>
<td>P2_IX_QUERY</td>
<td>Return last spoken index.</td>
</tr>
<tr>
<td>P2_DICT</td>
<td>Load user dictionary.</td>
</tr>
<tr>
<td>P2_PHONE</td>
<td>Telephone control.</td>
</tr>
<tr>
<td>P2_MODE</td>
<td>Synthesis mode control.</td>
</tr>
<tr>
<td>P2_LOG</td>
<td>Local terminal log control.</td>
</tr>
<tr>
<td>P2_TERMINAL</td>
<td>Local terminal control.</td>
</tr>
<tr>
<td>P2_MASK</td>
<td>Keypad mask control.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>P3 parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3_PH_STATUS</td>
<td>Return a status report.</td>
</tr>
<tr>
<td>P3_PH_ANSWER</td>
<td>Answer on P4 rings.</td>
</tr>
<tr>
<td>P3_PH_HANGUP</td>
<td>Hangup the phone.</td>
</tr>
<tr>
<td>P3_PH_KEYPAD</td>
<td>Enable keypad data entry.</td>
</tr>
<tr>
<td>P3_PH_NOKEYPAD</td>
<td>Disable keypad data entry.</td>
</tr>
<tr>
<td>P3_PH_TIMEOUT</td>
<td>Send a timeout report if no data entered in P4 seconds if P4 is greater than zero; disable timeouts if P4 is zero.</td>
</tr>
<tr>
<td>P3_PH_TONE</td>
<td>Dial out using tones.</td>
</tr>
<tr>
<td>P3_PH_PULSE</td>
<td>Dial out using pulses.</td>
</tr>
</tbody>
</table>

Several P2 commands return messages to the host.

<table>
<thead>
<tr>
<th>P2 command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2_IX_REPLY</td>
<td>Reply to P2_IX_REPLY. R3 contains the last index processed.</td>
</tr>
<tr>
<td>R2_IX_QUERY</td>
<td>Reply to P2_IX_QUERY. R3 contains the last index processed.</td>
</tr>
<tr>
<td>R2_DICT</td>
<td>Reply to P2_DICT. R3 contains the dictionary entry status code.</td>
</tr>
<tr>
<td>R2_PHONE</td>
<td>Reply to P2_PHONE. R3 contains the telephone status.</td>
</tr>
</tbody>
</table>

The following R3 parameters are returned after a P2_PHONE command.

<table>
<thead>
<tr>
<th>R3 parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3_PH_ONHOOK</td>
<td>Telephone is hung up (inactive).</td>
</tr>
<tr>
<td>R3_PH_OFFHOOK</td>
<td>Telephone is answered (active).</td>
</tr>
<tr>
<td>R3_PH_TIMEOUT</td>
<td>No data was entered by the telephone user within the required number of seconds.</td>
</tr>
<tr>
<td>R3_PH_TOOLONG</td>
<td>A telephone number to dial is too long.</td>
</tr>
</tbody>
</table>
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_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.

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 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 hard-copy edit conventions.
TERM_SETUSPEAK  Speak SETUP dialog.
TERM_FILTER     Filter escape sequences sent to the local terminal.

The following flags are for the P2_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 (P2_LOG, LOG_RAWHOST).
SPLICE_TERM     The terminal may send text to DECTalk (P2_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_INTMAX      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_splice 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 stty_save Terminal characteristics block (UNIX System V).

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

FILE *fildes File descriptor (RSX).

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

QIOPARM parm QIO parameter block (RSX).

(RSX only).

int#pos_xk TRUE if POS XK: driver (RSX only).

The flag entry controls library internal states.

_FLAGSPEAK Set if DECTalk is speaking.

_FLAGSLOG Set if LOG RAWHOST is set.

_FLAGSTERM Set if TERM HOST is set.

_FLAGSIGHTBIT 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 NONE.

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 numerous 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
#if defined EOS
#define EOS '\0'
#endif

#if defined unix
#if defined BSD_42
#include <sgtty.h>
#else
#if defined 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 0x1B /* Cancel <CTRL/X> */
#define SUB 0x1A /* Substitute */
#define NUL 0x00 /* Null code */
#define ESC 0x1B /* Escape */
#define DEL 0x7F /* Delete */

/*
 * C1 control characters
 */

#define SS2 0x8E /* Single shift 2 */
#define SS3 0x8F /* Single shift 3 */
#define DCS 0x90 /* Device control sequence */
#define DLID 0x9A /* ESC Z */
#define CSI 0x9B /* Control Sequence Introducer */
#define ST 0x9C /* String terminator */
#define OSC 0x9D /* Operating System sequence */
#define PM 0x9F /* Privacy Message */
#define APC 0x9F /* Application Program Control */
#define RDEL 0xFF /* Delete in right side */
#define CSI_DA_PRODUCT 19 /* Deictalk DA product code */

/*
 * Basic definitions for Deictalk device control
 * strings. All Deictalk sequences have a first parameter of
 * P1.DEICTALK. This provides an easy place for future Deictalk
 * products to fit into the scheme of things.
 */

#define DCS_F_DEICTALK 'Z' /* Deictalk final */
#define P1_DEICTALK 0 /* Deictalk param 1 */
#define R1_DEICTALK 0 /* Deictalk reply param 1 */

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

#define P2_PHOTEXT 0 /* Speak phonemic text */
#define P2_STOP 10 /* Stop speaking */
#define P2_SYNC 11 /* Synchronize */
#define P2_SPEAK 12 /* 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_NOKEYPAD 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 N 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 DSRDECLNVRFAIL 24 /* Last DECLNR 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_SETUSPEAK 0x0010 /* Spoken setup mode */
#define TERM_FILTER 0x0020 /* Filter logged esc. sequences */
C PROGRAM EXAMPLE

/*
 * Mode flags for the P2_MODE command.
 */
#define MODE_SQUARE 0x0001 /* [ ] are phonemic brackets */
#define MODE_ASKY 0x0002 /* Use ASKY 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 rawhost if set */
#define SPLICE_TERM 0x0004 /* Local host if set */
#define _FLAG_SPEAK 0x0001 /* Speaking, set by dt_splice() */
#define _FLAG_LOG 0x0002 /* Log rawhost 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_gesc() (get escape sequence) and dt_pesc() (put escape
 * sequence) use this structure for all processing.
 */
typedef struct {
    short state;     /* Processing state or intro */
    char final;      /* Final character in seq. */
    char private;   /* Private introducer */
    #ifdef decus
        union
                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 process 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 qiparm { /* 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 */
} QIODPARM;

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

#define PEND_SIZE 32 /* Pending buffer size */
#endif
#define IN_BUFLEN 32
#endif
#define OUT_BUFLEN 128
#endif

#endif

#define IN_BUFLEN < 1 || OUT_BUFLEN < 1 || PEND_SIZE < 1)
<< error, mandatory parameters aren't correct >>
#endif

typedef struct DECTalk { /* Chain all units together */
 struct DECTalk *link; /* I/O channel */
 short unit; /* For dt_timeout() */
 short timeout; /* Speech and "splice" flags */
 short flag; /* Bytes in pending buffer */
 short pend_fc; /* Pending buffer fill index */
 short pend_fp; /* 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. */
 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.
 */

#define unix
#define BSD_42
    struct sgttyb stty_save; /* Terminal flags */
#else
#define UNIX_V
    struct termios stty_save; /* Terminal flags (UNIX V7) */
#endif
#endif
#endif
#endif
#if defined rsx
    FILE *fildes; /* File descriptor */
    IOSB iosb; /* I/O status block */
    QIODPARM parm; /* QIO parameter block */
    short pos_xk; /* 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_pctest(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_isttimeout(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.
 */
```c
#define dt_get     dt_ioget
#define dt_put     dt_ioput

 ifdef 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_pptest(dd,r3) (dt_test(dd, R2_PHONE, r3))
#define dt_offhook(dd)  (dt_pptest(dd, R3_PH_OFFHOOK))
#define dt_onhook(dd)   (dt_pptest(dd, R3_PH_ONHOOK))
#define dt_timeout(dd)  (dt_pptest(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

 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
 ifndef 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.

```c
#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, P2_MODE, MODE_SQUARE, -1);
    retries++; /* Count attempts */
    while (dt_answer(dt, 1)) { /* Answer the phone */
        if (process(dt)) { /* Do user process */
            ncalls++;
            retries = 0;
            /* User ran ok, */
            /* 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");
```
```c
finis:  dt_abort = FALSE;  /* Restart output */
        dt_reset(dt);      /* Hangup DECTalk */
        dt_putchar(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 */
        printf(work, "You pressed %c", c);
        dt_talk(dt, work);
        if (c == 'T') {        /* Make 'T' special */
                dt_timeout(dt, 0); /* No timeouts now */
                dt_talk(dt, "Long message...");
        }
    }
    /*
    * Timeout is normal, others are errors.
    */
    return ((c == 'T') ? TRUE : FALSE);
```
DTANSW.C
This routine hangs up the phone and answers on n rings.

/*)LIBRARY
*/
#endif 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_dcs(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);
DTCLOSE.C
This routine closes the DECTalk channel and frees all buffers.

/*LIBRARY*/

#define DOCUMENTATION

title dt_close Terminate DECTalk Operation
index
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 rsx
#include <cx.h>
#include <qiofun.h>
#include <qioret.h>
#include <qiofd.h>
#define QIO_EFN 1

static QIOPARM nopr;

#endif

 dt_close(dt)
 register DECTALK *dt;
 /*
 * Close the DECTalk channel.
 * /
 {
 register DECTALK **linkp;
#endif 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
typedef
#endif
close(dt->unit);
#endif
#ifdef vms
sysdassgn(dt->unit);
#endif
#ifdef rt11
rs_close(dt->unit);
#endif
#ifdef rx
qioq(ID_DET, dt->unit, QIO_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);
free((char *)dt);
return (NULL);
}
DTCMD.C
This routine sends a DCS command to the DECTalk terminal.

/****LIBRARY
*/

 ifndef 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_st(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;
        }
    }
    dt_pesc(dt, &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.

```c
/*)LIBRARY */
#endif DOCUMENTATION

#define dt_dchar Dump One Character Visibly
#define 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
*/

#define DOCUMENTATION

title dt_dcs
index Send a DECTalk DCS Command
        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_xxx 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.

#undef DOCUMENTATION

#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_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.

```c
/* Library */

#define DOCUMENTATION

title dt_dial
index Dial the Telephone

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_*** 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 '! ' (for a one second delay) or the '"' for a 250 millisecond switch-hook flash. All other characters are ignored.

If pulse dialing is selected, only the digits, '! ' and '"' 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 allotted 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.

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

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

int dt_dial(dt, p3, number, wait, message)
register DECTALK
int register char
int char
*dt; /* Device descriptor */
p3; /* P3_PULSE or TONE */
*number; /* Number to dial */
wait; /* Call progress delay */
*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_st(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_talk(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))
        dtופ(dt);                 /* No response, hangup */
    return (FALSE);
DTDRAI.C
This routine absorbs any type-ahead characters. No errors are possible.

/*)LIBRARY
*/

#define DOCUMENTATION

title dt_drain Drain Pending Input
index Drain pending input

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

dt_drain(dt)
DECTLK *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 DECTLK *dt;
/*
 *  dt_drain() tosses out any pending type-ahead.
 */
{
  dt->pendFc = dt->pendFp = dt->pendEp = 0;
#ifdef BSD_42
    ioctl(dt->unit, TIOCFLUSH, NULL);
#else
#ifdef UNIX_V
    ioctl(dt->unit, TCFLUSH, 0);        /* UNIX V7 */
#endif
#endif
  dt->inPtr = dt->in_end = dt->in_buff;
}
#endif
#ifdef vms
#include <iodef.h>

dt->pend_fc = dt->pend_fp = dt->pend_ep = 0;
/*
 * This is probably sub-optimal. It should be possible
 * to do "sys$qiow(...
 * IO$READBLK | IO$M_PURGE | IO$M_TIMED
 * with a zero-length timeout, but I sure don't know.
 */
while (dt->vmsread(dt,
      IO$READBLK | IO$M_NOECHO | IO$M_NOFILTER | IO$M_TIMED,
      IN_BUFFLEN, 0) >= IN_BUFFLEN)
{
    dt->in_ptr = dt->in_end = dt->in_buff;
}
#endif

#ifdef rt11
#include <rsts.h>

dt->pend_fc = dt->pend_fp = dt->pend_ep = 0;
clrxb();
xrb.xrlen = 7; /* Cancel type-ahead */
xrb.xrci = dt->unit * 2;
xrb.xrbklm = TTYHND;
rstsys(_SPEC);
dt->in_ptr = dt->in_end = dt->in_buff;
#endif

#ifdef rsx

dt->pend_fc = 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
*/
#endif DOCUMENTATION

title dt_dump Dump Escape Sequence Buffer
index Dump escape sequence buffer

synopsis

#include <stdio.h>
#include "dectlk.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("kb2:");
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 = 1; 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
 */

#ifndef 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 "dectlk.h"

#ifndef dt_eol
#undef dt_eol
#endif

dt_eol(dt) DECTALK *dt; /* Device descriptor */
{
#ifndef unix
    dt_putchar(dt, '\r');
#endif
    dt_putchar(dt, '\n');
    dt_putchar(dt, 0);
}
DTGESC.C
This routine reads an escape sequence or keypad character.

/*)LIBRARY
*/

#define DOCUMENTATION

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

int
dt_gesc(dt, sec)
DECTALK char *dt; /* Device descriptor */
char sec; /* O.S. 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 dt_gesc() 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 retained 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_gesc() 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_gesc() 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_gesc() 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_PARAMMAX)
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 test for overly
large parameter values and assumes that all overflow errors
are due to invalid escape sequences.

# endif

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

int dt_gesc(dt, sec)
register DECTALK
   *dt;    /* Dectalk device */
   int sec;   /* O.S. timeout */
/ * Return a character or sequence
 * /
{
    register int c;
#endif decus
    register unsigned  *p;
#else
    register unsigned short *p;
#endif
#endif DT_DEBUG
    if (dt_debug) {
        dt_put(dt, 0);
        printf("get: \"\n");
        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 c1_cont when changing <ESC> [
         * to CSI, etc.
         */
        c1_cont: /*
            if (c == ESC        /* ESC, CSI, DCS */
            || c == CSI        /* Introduce control */
            || c == DCS) {
                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)) { /* resetter. */
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 = 'X';
   \}
   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. */
#endif
decus

c = dt->seq.state;            /* Fetch return value */

#else

  c = (unsigned short) dt->seq.state;
  #endif
endif
dt->seq.state = 0;
exit:
#endif
 DT_DEBUG
if (dt_debug)
    printf("\n");
#endif
    return (c);

/* No sequence pending */
/* Here to return char */
/* and return. */
DTGET.C
This routine reads a character from the DECTalk terminal line.

/**)LIBRARY */

 ifndef DOCUMENTATION

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

 synopsis

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

#ifndef 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_ioget(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
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*/

#define 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 code;

dt_drain(dt);        /* Ignore pending input */
dt_dcs(dt, P2_TERMINAL, 0, -1); /* No local-host stuff */
dt->flag &= ~ _FLAG_TERM;      /* Remember this fact */
/*
 * Read device attributes and fail if it isn't DECtalk.
 * Expected reply is <ESC>?[?19c for the DTC01-AA
 */
dt_pesc(dt, & DT_who_are_you);
if (dt_read(dt, 15) == CSI
    && dt->reply.final == 'c'
    && dt->reply.private == '?'
    && dt->reply.inter[0] == 0
    && dt->reply.param[0] == 1
    && ((code = dt->reply.param[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.

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

#define DOCUMENTATION

#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 (as may be required by specific telephone
system requirements).
#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.
 */

#define FUDGE 60   /* RSTS/E needs extra time */
#undef FUDGE
#define FUDGE 15   /* This is just a guess */

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

dt_timeout(dt, sec); /* Set/clear timeout */
    if (dt_iskey(dt)) {
        code = dt->pend[dt->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 * 4 + FUDGE))) <= 0)
            code = 'E';
        else if (dt_isftimeout(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
* 84.04.10      MM      HNGTTY incorrectly specified.
*/
#endif DOCUMENTATION

title dt_ioget      Read one Character from DECTalk
index
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; if it is zero 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.

note
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.
#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 */

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

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

int dt_input(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 */
#ifndef BSD_42
    auto int fdmask;    /* File descriptor mask */
    struct timeval timeout;    /* Select() timer value */
#else
#ifndef UNIX_V
    register int ede;
    /* For error handling */
#endif
    extern int errno;    /* System error value */
#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_input(dt, 0);    /* Flush output */
#endif
    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_BUFSIZE);
  }
#else
  #ifdef UNIX_V
    signal(SIGALRM, ignore); /* Enable alarms */
    alarm(sec); /* Start timeout */
    error = 0; /* Clear error flag */
    incount = read(dt->unit, dt->in_buff, IN_BUFSIZE);
    ecode = error; /* 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
#endif

#include <ssdef.h> /* System status codes */
#include <iodef.h> /* I/O request codes */
/
* Define the possible vms input flavors */

#define RAW_READ (IO$READBLK | IO$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_ioget(dt, sec)
register DECTALK
  *dt; /* DECTalk device */
  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->ioput(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_BUFFLEN, 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 incount;  /* Status parameter */
IOSTAB status;       /* I/O status block */
register int i;       /* For debugging */

/*
 * termset 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.
 */
incount = sys$qiow(1, /* Event flag */
  dt->unit, /* Input channel */
  command,  /* Timed read */
  &status,  /* I/O status block */
  NULL,     /* AST block (none) */
  0,        /* AST parameter */
  dt->in_buff, /* P1 - input buffer */
  how_many, /* P2 - buffer length */
  timeout, /* P3 - wait P3 seconds */
  &termset, /* P4 - terminator set */
  NULL,     /* P5 - ignored (prompt buffer) */
  0);       /* P6 - ignored (prompt size) */
if (incount == SS$_TIMEOUT) /* Timeout returned */
  return (DT_TIMEOUT);
else if (incount != SS$_NORMAL) /* Some other error */
  return (DT_ERROR);

incount = status.term_offset + status.term_size;
if (incount < 0) /* Nothing input? */
  return (DT_TIMEOUT); /* equals timeout. */
return (incount);

#endif

#ifdef rt11

/*
 */
#endif

#include <rts.h>

int
dt_ioget(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.
    */
    clrxb();
    xrb.xrlen = 4; /* No delimiter */
    xrb.xrci = dt->unit * 2;
    xrb.xrblkm = TTYHND;
    rtsys(_SPEC);
    clrxb();
    xrb.xrlen = 3; /* No echo */
    xrb.xrci = dt->unit * 2;
    xrb.xrblkm = TTYHND;
    rtsys(_SPEC);
    incount = rs_read(dt->unit, dt->in_buff,
        IN_BUFFLEN, 0, 0, sec, 0);
    if (incount == (-HNTTY)) /* 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
#endif 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
 *  lunbuf.h   Device characteristics buffer
 */

#include <cx.h>
#include <qiofun.h>
#include <qioret.h>
#include <qiottd.h>
#include <lunbuf.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    termtable[16];      /* Terminator bitmask */

int
dt_ioset(dt, sec)
register DECTALK  *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 rsx 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_ioset(dt, 0);               /* Flush output */
    if (dt_abort)
        return (DT_ERROR);
    if (dt->pos_xk) {
/*
 * The PRO-350 XK: 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, QIO_EFN,
                 &dt->iosb, NULL, &dt->parm);
if (((incount = fixiosb(dt)) == 0) { 
    dt->parm.size = 1;
    if (((dt->parm.p3 = (256 * 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_ERRDR);
    }
    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->in_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_RME, dt->unit, QIO_EFN,
                         &dt->iosb, NULL, &dt->parm);
        incount = fixiosb(dt);
    }
    if (incount == 0) { 
        if (sec == 0) {
            dt->parm.table = termtable;
            qiow(IO_RTT | TF_RME, dt->unit, QIO_EFN,
                 &dt->iosb, NULL, &dt->parm);
            if (((incount = fixiosb(dt)) == 0) 
                return (DT_ERRDR);
        }
        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 (alarm) 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.
 */
wtlo0(QIO_EFN | MKT_EFN);
rdfs(efn_buffer);
if (((efn_buffer[0] & MKT_EFN) == 0)
   && cmkt(MKT_EFN, NULL));  /* Cancel timer */
if (((efn_buffer[0] & QIO_EFN) == 0) {
   qiolw(IO_KIL, dt->unit, QIO_EFN,
         &dt->iosb, NULL, &dt->parm);
   return (DT_TIMEOUT);
}
if (((incount = fixiosb(dt)) == 0)
    return (DT_ERROR);
}

} /* fixture() */

/*
 * Common (success) exit from all read routines
 */
dt->in_end = &dt->in_buff[incount];
return (*dt->in_ptr++ & 0xFF);

} /* fixiosb(dt) */

register DECTALK
*dt;   /* DECTalk device */
/ * This routine returns the correct input count.  
  * The code is unusual. 
  *  
  * fixiosb() 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_ABORT 
  || dt->iosb.count == 0) { 
  return (0); 
  /* Read aborted */ 
 } 
 else if (dt->iosb.status != IS_SUCCESS 
  && dt->iosb.status != IS_TOKEN) { 
  $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
*/

#define DOCUMENTATION

title   dt_ioput
index   Write one Character to DECTalk
        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.
#ifndef vms
#include <ssdef.h>
#include <stdio.h>
#endif

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

#ifndef 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
 */
#endif

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

#define QIO_EFN 1       /* I/O event flag */

#endif

int dt_ioput(int dt, int c) {  /* 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
        *dt->out_ptr++ = (c == ESC) ? (ESC | 0x80) : c;
        #else
        *dt->out_ptr++ = c;
        #endif
    }
#endif
}
size = (dt->out_ptr - dt->out_buff);
if ((c == 0 && size > 0) || size >= OUT_BUFFLEN) {
    /*
     * 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$qiow(1, /* Event flag */
            dt->unit, /* Input channel */
            IO$WRITEBLK : 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 rl11
        if ((code = rs_write(dt->unit, dt->out_buff, size, 0, 0, 0)) != 0) {
            $$ferr = code;
            perror(dt->device);
            exit(IO$ERROR);
        }
        #endif
        #ifdef rsx
        dt->parm.size = size;
        dt->parm.buffer = dt->out_buff;
        dt->parm.table = NULL;
        if ((code = qiow(IO$WAL, dt->unit, IO$EFN,
            &dt->iosb, NULL, &dt->parm)) != IS$SUC) {
            $$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.

/ * ) L I B R A R Y
 *

 ifndef DOCUMENTATION

title dt_iskey Test for type-ahead
index

 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.

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

 ifndef dt_iskey
# undef dt_iskey
#else
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
*/

#define DOCUMENTATION

title dt_istimeout Test Phone Reply for Keypad Timeout
index

summary

#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.

#include <stdio.h>
#include "dectlk.h"
#endif
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
*/
#endif DOCUMENTATION

title     dt_isvalid    Test for Valid Keypad Character
index     dt_isvalid    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"
#endif dt_isvalid
#endif 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.

```c
#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.
```

```c
#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) ? P3_PH_KEYPAD : P3_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.

/**)LIBRARY
*/

#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.
*endif

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

int
dt_msg(dt, p2, p3, p4, r2, r3)
register DECTALK *dt;  /* Device descriptor */
int  p2, p3, p4;  /* Pn 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_read(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 dt_offhook 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"
#undef dt_offhook

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

/*
 * Test whether the phone is off-hook.
 */
{
    return (dt_test(dt, 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   dt_onhook   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 is the DECTalk phone reply with the R3 parameter equal to R3_PH_ONHOOK.

# endif

 # ifndef 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, R2PHONE, 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).

/ * ) L I B R A R Y
 * /

# 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->dcs(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 <descrip.h>    /* String descriptors */

  typedef struct dsc$descriptor_s STRING; /* string descriptor */

  * The following macro builds a descriptor from an argument string.
  *
  #define descrip(text, 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 <cx.h>
#include <qiofun.h>
#include <qioret.h>
#include <qiotld.h>
#include <lunbuf.h>
#define QIO_EFN I
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
  #ifdef BSD_42
    struct sgttyb  stty_buffer;   /* Terminal flags */
  #else
    #ifdef UNIX_V
      struct termio              stty_buffer;   /* Terminal flags */
    #endif
  #endif
  #endif
  extern char *ttlname();   /* Get stdin name */
#endif
#ifdef vms
  STRING             dev;
#endif
#ifdef rt11
  char                work[30];
  extern int          $$rsts;  /* Invalid device */
  extern char *E$$$NOD;     /* Fatal error */
  extern char *E$$$FAT;     /* No more channels */
#endif
#ifdef rsx
  extern int          $$rsts;  /* TRUE if RSTS/E */
  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);


/*
 * 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
#endif

#endif

*/

/*
 * The following edit was reported by a customer for a Zilog
 * "Zeus" port but hasn't been independently tested.
 */

#endif

#endif

#endif

#ifdef vms

descrip(name, &dev);
if (sys$assign(&dev, &dt->unit, 0, NULL) != SS$NORMAL)
goto error1;

#endif
#endif
#endif

if (!*$rsts) {
    $ferr = (int) &E$FAT; /* Illegal function */
    goto error1;
}

/*
 * Search for a free channel.
 */
for (i = 12; i > 0; i--) {
    cirxrb();
    xrb.xrci = i * 2;
    if (rstsyst(_POSTN) == NOTOPN)
        break;
}
if (i <= 0) { /* Fail if all channels */
    $ferr = (int) &E$NDC; /* are in use. */
    goto error1;
}

dt->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
 */
sprintf(work, "%s/mo:%d", name, 1+16+32);
if (rs_open(i, work, "r") != 0)
    goto error1;
if (((rirqb fqflag & 0xFF) != TTYHND) {
    $ferr = (int) &E$NOD; /* Not a terminal */
    rs_close(i);
    goto error1;
}
#endif
#endif

if (!*$rsts) {
    $ferr = IE_IFC; /* Not on RSTS/E */
    goto error1;
}

/*
 * We only call fopen() to get a free lun.
 */
if (((dt->fildes = fopen(name, "rn")) == NULL)
    goto error1;

dt->unit = fileno(dt->fildes);
glun(dt->unit, &lunbuf);
if ($pos
    & lunbuf.g_luna[0] == 'X'
    & lunbuf.g_luna[1] == 'K')
    dt->pos_xk = TRUE;
else
    dt->pos_xk = FALSE;
if (((i = qiow(IO_ATT, dt->unit, QIO_EFN,
        NULL, NULL, &noparm)) != IS_SUC) {
    fclose(dt->fildes);
    $ferr = i;
    goto error1;
}
*/
  * Normal exit, initialize other pointers
  */
dt->link = dt_root;
dt_root = dt;
dt->out_ptr = dt->out_buff;
dt->flag = _FLAG_SPEAK;
return (dt);
  /* Out buffer setup */
  /* Normally speaking */
  /* 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.

/*) LIBRARY
 */
 ifndef 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.
 */
 ifndef unix
 include <errno.h>
 ifndef BSD_42
 include <sys/types.h>
 include <time.h>
#endif
int
dt_peek(dt, sec)
register DECTALK *dt; /* DECTalk device */
/*
 * UNIX.
 */
{
#ifdef BSD_42
register int incount; /* Count and error code */
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 N (N >= 3)
 */
ioctl(dt->unit, FIONREAD, &pending);
    return(pending > 0);
#else
dt->in_ptr = dt->in_end = &dt->in_buff[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 */
if (dt->abort || incount <= 0) /* Other error? */
    return (FALSE); /* Return bad failure */
dt->in_end = &dt->in_buff[incount];
return (TRUE);
#endif
#endif
#endif
#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 */
} IOSTAB;
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;
    IOSTAB status; /* I/O status block */

    if (dt->pend_fc > 0 || dt->in_ptr < dt->in_end)
        return (TRUE);
    incount = sys$qiow(1, /* Event flag */
        dt->unit, /* Input channel */
        IO$SENSEMODE I 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
/*
 * 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];
  clrxrb();
  xrb.xrlen = 4;  /* No delimiter */
  xrb.xrci = dt->unit * 2;
  xrb.xrbkm = TTYHND;
  rstsystx(SPEC);
  clrxrb();
  xrb.xrlen = 3;  /* No echo */
  xrb.xrci = dt->unit * 2;
  xrb.xrbkm = TTYHND;
  rstsystx(SPEC);
  incount = rs_read(dt->unit, dt->in.buff,
                   IN_BUFLEN, 0, 0, 0, 8192);
  if (incount == -1(HNOTTY))
    return (FALSE);
  else if (incount <= 0)
    return (FALSE);
  dt->in.end = &dt->in.buff[incount];
  return (TRUE);
}
#endif

/*
 * 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
 * lunbuf.h Device characteristics buffer
 */
#include <cx.h>
#include <qiofun.h>
#include <qioret.h>
#include <qiottd.h>
#include <lunbuf.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 QIO Parm gmcParm = { gmcbuf, sizeof gmcbuf };
static int termtab[16];  /* Terminator bitmask */
int dt_peek(dt)
register DECTALK *dt; /* DECTalk device */

/*
 * RSX:
 */

register int incount;    /* Count and error code */
register char *ip;        /* To copy to rsx buff */
int errorcode;

/*
 * Return buffered character (if any)
 */
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];
if (dt->pos_xk) {
    /*
     * The PRD-350 Xk: port is actually pretty simple.
     */
    dt->parm.buffer = dt->in_buff;
dt->parm.size = IN_BUFFLEN;
dt->parm.p3 = 0;    /* No timeout */
errorcode = qiow(IO_RLB | TF_TMO, dt->unit, QIO_EFN,
     &dt->iosb, NULL, &dt->parm);
if ((incount = fixiosb(dt)) == 0) {
    return (FALSE);
}
dt->in_end = &dt->in_buff[incount];
return (TRUE);
}
else {
    /*
     * Check whether anything is in the
     * system type-ahead buffer.
     */
    errorcode = qiow(SF_GMC, dt->unit, QIO_EFN,
     &dt->iosb, NULL, &gmcparm);
    return (errorcode == IS_SUC && gmcbuf[1] > 0);
}
static int ffixiosb(dt)
register DECTALK *dt; /* DECTalk device */

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

    if (dt->iosb.terminator != NULL) {
        /*
         * 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); /* Read aborted */
    }
    else if (dt->iosb.status != IS_SUC
             && dt->iosb.status != IS_TMO) {
        $$ferr = 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.

```c
/*)LIBRARY */
#define DOCUMENTATION

#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.

C1 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
```c
#include <stdio.h>
#include "deictk.h"

dt_pesc(dt, seq)  
register DECTALK *dt;  /* Deictalk 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_putstr(dt, seq->inter[i++]);
}
dt_putstr(dt, seq->final);  /* Output the final */
break;

default:
    break;
}
#endif
#ifdef DT_DEBUG
    if (dt_debug)
        printf("\n");
#endif
#endif
#endif
#endif
#endif

static unsigned power10[] =
{ /*
* Powers of 10 for intout
*/
#ifdef INT_32
    10000000,
    1000000,
    100000,
#endif
    10000,
    1000,
    100,
    10,
    1,
};

#define NPOWERS ((sizeof power10) / (sizeof (unsigned)))

static
intout(dt, value)
DECTALK
register unsigned
    *dt;  /* DECTalk device */
        /* Value to convert */
/*
* Convert an unsigned number to ASCII and call dt_putchar() 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_put(dt, out_char + ‘0’);
    }
} while (++power < &power10[NPOWERS]);
}
DTPhon.C
This routine sends a DECTalk phone message.

/ * ) LIBRARY
* /
 ifndef DOCUMENTATION

title dt_phone Send a Phone Message
index
synopsis
  #include <stdio.h>
  #include "dectlk.h"

  int dt_phone(dt, p3, p4)
  DECTALK *dt; /* Device descriptor */
  int p3; /* P3_PHxxxx 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 P2PHONE).

  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
R2PHONE respectively. The application program
should then test for offhook/onhook as appropriate.

  Returns TRUE if successful. If FALSE, something is funny.

 endif

 ifndef 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, P2PHONE, p3, p4, R2_PHONE, -1));
}
DTPTES.C
This routine tests a phone reply.

/*)LIBRARY
*/

 ifndef DOCUMENTATION

title dt_ptest Test Phone Reply
index dt_ptest 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

 ifndef 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
*/
#endif DOCUMENTATION

title    dt_put     Write one Character to DECTalk
index    dt_put     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"
#endif dt_put
#endif dt_put

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
index       Read Sequence or Character
            Read sequence or character
synopsis
            
            #include <stdio.h>
            #include "dectlk.h"

            int
dt_read(dt, sec)
            DECTALK    *dt;  /* Device descriptor */
            char      sec;  /* O.S. 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

#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 CO 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, sec);
    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, 1);
        } while (code > 0 && code < 0x80);
    }
    return (dt->reply.state);
}
DTRESE.C
This routine sends a soft-reset escape sequence.

/*)LIBRARY
*/

#define DOCUMENTATION

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

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

description

Send a "soft reset" escape sequence.
No errors are possible.

#define

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

static SEQUENCE soft_reset = {
    CSI, 'p', 0, { 0 }, { 1, '!'}
};

dt_reset(dt)
register DECTALK  *dt;
/*
 * dt_reset() sends a soft-reset escape sequence.
 */
{
    dt_pesc(dt, &soft_reset);
    dt->flag1 = _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 dt_save    Save user type-ahead

 synopsis

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

int dt_save(dt, c)
DECTALK *dt;    /* Device descriptor */
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
int
dt;    /* Dectalk device */
c;    /* Character to test */
{
    register int
    timeout;    /* Current value */
    if (!dt_isvalid(c))    /* Not a keypad button? */
        if (!dt_isftimeout(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 type-ahead 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_fc < PEND_SIZE)    /* Save it if there's */
        dt->pend_fc++;    /* 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
index Manage Local Terminal

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.
#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_RAWHOST);
    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.

 /**************************************************************************
 /*
 #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_pesc(dt, &string_terminator);
 }

*/
*/
DTSYNC.C
This routine synchronizes the application with DECTalk.

/***************************************************************************/
#define DOCUMENTATION

title dt_sync Synchronize with DECTalk
index
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.

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

int
dt_sync(dt)
register DECTALK
  *dt; /* Device descriptor */
  /*
  * Synchronize DECTalk and the application.
  */
{
  dt_dcs(dt, P2_SYNC, -1, -1); /* Synchronize */
  dt->flag |= _FLAG SPEAK; /* Now speaking */
  return (dt_msg(dt, P2_IX_QUERY, -1, -1, R2_IX_QUERY, -1));
DTTALK.C
This routine speaks one line of text.

/ * )LIBRARY
* /

#define DOCUMENTATION

title dt_talk
index 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.

/*)LIBRARY
 */

#ifdef DOCUMENTATION

title      dt_test       Test a DECTalk Reply
index      dt_test       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.state == 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
*/
#endif 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 (dt_iskey(dt)) /* If enabling, */
    sec = 0; /* Disable if typeahead */
  if (sec != 0) {
    dt_sync(dt); /* Still enabling? */
    if (dt_iskey(dt)) /* Synchronize and */
      sec = 0; /* Check again. */
  }
}
if (dt->timeout == sec) /* Don’t set to the */
  return (TRUE); /* same value */
dt_phone(dt, P3_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.

*/

#ifndef DOCUMENTATION

title dt_tone
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 test string as a tone dialing sequence. If the telephone was on-hook
when dt_tone() 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
'!' (for a one second delay) or the '"' 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.

/**)LIBRARY */

#define DOCUMENTATION

#define title dt_trap  Trap <=CTRL-C> Interrupts
#define index  Trap <=CTRL-C> Interrupts

#define 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.

#define

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

#define ifdef unix
#define include <signal.h>

#define static
#define 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 rsx
#include <cx.h>
#include <qiofun.h>
#include <qioret.h>
#include <qiottd.h>
#define QIO_EFN 1

static
catch()
{
        register DECTALK  *dt;

        astset(); /* AST entry */
        if (dt_abort)
                $$fail();
        dt_abort = TRUE;
        /* Kill all pending DECTalk I/O */
        for (dt = dt_root; dt != NULL; dt = dt->link) {
                qiofw(IO_KIL, dt->unit, QIO_EFN,
                        &dt->iosb, NULL, &dt->parm);
        }
        astx(1); /* AST exit */
}
static QIOPARM asparm = { NULL, 0, catch }; 

dt_trap() 
/*
 * Trap CTRL-C interrupts.
 */
{
    qiow(IO ATA, fileno(stderr), QIO EFN,
         NULL, NULL, &asparm);
}
#endif
#endif

static
catch()
/*
 * Executed by the operating system if an interrupt is
 * detected.
 */
{
    if (dt_abort)
        **fail();
    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

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 "declk.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 != '\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.

```c
/* BUILD */
$(include) = { dectlk.h }
$(rclib) = { dtlib,c:rstslb,c:lib }
$(rxlib) = { dtlib/lb,c:cx/lb,c:c/lb }
*/

#include <stdio.h>
#include "dectlk.h"
#endif vms
extern int errno;
deinic IOERROR errno
#else unix
#define IOERROR 1
#endif

DECTALK *dt;

main(argc, argv)
int argc;
char *argv[];
{
    char *dev;

dev = "ttg7:";
    if (argc > 1)
        dev = argv[1];
    if ((dt = dt_open(dev)) == NULL) {
        perror(dev);
        printf("Can't open DECTalk\n");
        exit(IOERROR);
    }
    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_scs(dt, P2_MODE, MODE_SQUARE, -1);
        dt_talk(dt, "Hello world.\n");
        dt_sync(dt);
        dt_dump("after sync", &dt->reply);
        dt_close(dt);
        printf("Success.\n");
    }
```
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   "*
20   "DECTalk function library and a sample application."
20   "The function library generally duplicates the C
20   "library, with some minor simplifications.
20   "*
20   "The sample program reads a string of numbers from
20   "the keypad and speaks them as a number, and as a
20   "string of digits. The "$" key functions as a dollar
20   "sign, and the "#" key functions as a decimal point.
20   "The program also illustrates how an application
20   "might manage keypad timeouts.
20   "****
100  "*
100  "Defaults"
100  "****
   DEF.kb$ = "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("DeCTalk terminal", DEF.kb$)  
  \ DT.log$ = (FNyeno("Enable logging", DEF.log$))  
  \ debug$ = FNyeno("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%)  
  \ ! Trouble?  
  \ q$ = FNlog("Too many retries")  
  \ goto 1800  
  \ ! Fatal.  

1300  next  
  \ ! For all calls  
  \ ! For all restarts  

1800  q$ = FNlog("finished after " + num1$(ncalls$))  
  \ close 2% if DT.log$  

1900  goto 32767  
  \ ! All done  

2000  def* FNprocess$  
  \ !  
  \ ! FN process $  
  \ !  
  \ ! 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.  

2100  FNprocess$ = FALSE%  
  \ ! Assume failure  
  \ ! Count button presses  
  \ q$ = FNlog("answered")  
  \ q$ = FNphone("Welcome to DeCTalk.")  
  \ q$ = FNphone("It is now " + time$(0%)  
  \ + " on " + date$(0%) + ").")  
  \ q$ = FNphone("Enter a number, the star key means")  
  \ q$ = FNphone("dollar sign, while the number-sign")  
  \ q$ = FNphone("key means decimal point.")  
  \ if (not FNphone("20")) then  
    \ ! turn the keypad on  
    \ q$ = FNlog("error enabling keypad")  
    \ goto 2080  
  \ ! Error exit
2020 if (not FNptest%(R3.PH.OFFH00K%)) then & &
q% = FNlog%("enable keypad, state:" + num1%(R3)) & &
\ goto 2080 & Error exit & &
2030 while TRUE% & &
\ timer% = 10% & For all numbers & &
\ work$ = """ & Input buffer & &
& while TRUE% & &
\ 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 2080 if c$ = 'T' & Timeout & &
\ c$ = "$" if c$ = "'" & 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 is" + FNexpand$(work$) + ":") & &
\ next & Read all numbers & &
2060 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 *
* FNExpand$( text$ ) *
*
* 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 *
* "12.3", the output will be " 1 2 point 3". Note *
* the leading blank.
*
* def* FNExpand$(text$)
\ q$ = """ & Output work &
\ for q% = 1% to len(text$) & For each byte &
\ q$ = mid(text$, q%, 1%) & Locate it &
\ q$ = "point" if q$ = ":" & Fix the &
\ q$ = "minus" if q$ = ":" & special &
\ q$ = "dollar sign" if q$ = ":" & cases &
\ q$ = q$ + ":" + q$ & 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%(kb$) Initialize DECTalk on kb:
! FNanswer% Finish last call, answer next
! FNhangup% Hangup the call
! FNkey%(timeout%) Read a character with timeout
  Returns the character, or
  E Error (from RSTS)
  H Phone hung up
  T Timeout
  X Bad Escape sequence
! FNtimeout%(sec%) Set specified timeout, 0 = none
! FNtest%(R2%, R3%) Test current reply, true if ok
  R3% is -1 to ignore it.
  FNtest%() checks character, intermediates, and finals.
! FNptest%(R3%) Test phone reply (R2% checked)
! FNsnd%(text$) 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$, R2%, R3%) Send DCS seq. test reply.
  text is "P2;P3...", return TRUE if ok.
  Note: FNmessage%() ignores
  R3.PH.TIMEOUT replies.
! FNphone%(text$) 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
!* The application program generally doesn't call the following routines.

!* FNsave%(char)        Save type-ahead character, return TRUE if saved.
!* FNdc$s%(text$)       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:

!* R1%, 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 (parity bit 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.PHONEx           R2% phone reply
!* R3.PH.ONHOKx        R3% (phone hung up)
!* R3.PH.OFFHOKx       R3% (phone is alive)
!* R3.PH.TIMEOUTx      R3% (keypad timeout)
!* DT.anything         reserved for local buffers
!* SEQ.anything        reserved for sequence parser
!* q[anything]         general temporaries
def FInit%(kb$)
!
! F N i n i t % ( k b $ )
!
! Initialize the DECTalk device
! Return TRUE% if ok, FALSE% if error
!

**
!* Open the terminal in "binary" mode.
!* Then initialize all constants.
!* open kb$ for input as file 0%, 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
\ DC5% = ascii('P') - 64% + 128% ! C1 control
\ ST% = ascii('\') - 64% + 128% ! characters
\ R2.PHONES = 70%
\ R3.PH.OHHOOK% = 0%
\ R3.PH.OFFHOOK% = 1%
\ R3.PH.TIMEOUT% = 2%

q% = FnSend%(chr$(ascii('Q') - 64% + 128%)) ! CTRL-Q
q% = FcGet%(2) while (q% > 0%) ! Drain text
q% = FNdcs%("82") ! No local-host
q% = FNsix%("c") ! "Who are you"
q% = FfReadEscapeSeq%(5%) ! Read escape sequence
if DT.char% <> CSI% ! Check
or DT.final$ <> 'c' ! for
or DT.private$ <> '?' ! DECTalk
or R1% <> 19% then ! reply
q% = FfIIn%("initialization")
FInit% = FALSE%
! Return failure
! goto 10190 ! from FInit%( )

q% = FSend%(ESC$ + "!p") ! Soft Terminal Reset
q% = FNdcs%("80:1") ! SET MODE SQUARE
\ DT.timeout% = 0% ! No timeouts now
\ FInit% = TRUE%
! Return TRUE

fnend
10200 def* FNanswer%  
  !  
  ! FNanswer%  
  !  
  ! 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%  
  \ q% = FNget%(2%) while (q% > 0%)  
  \ goto 10290 if (not FNphone%(""))  
  \ if (R3% = R3.PH.OFFHOOK%) then  
  \ goto 10290 if (not FNhangup%)  
  ! Assume error  
  ! Drain text  
  ! poll status  
  ! if alive,  
  ! hangup phone  
10220 if (R3% <> R3.PH.ONHOOK%) then  
  \ q% = FNfunny%("hangup/poll")  
  \ goto 10290  
  ! still alive?  
  ! Urk.  
  ! exit this  
10230 goto 10290 if (not FNphone%("10;1"))  
  \ if (R3% <> R3.PH.ONHOOK%) then  
  \ q% = FNfunny%("enable answer")  
  \ goto 10290  
  ! answer 1 ring  
  ! ok?  
  ! Urk.  
  ! exit this  
10240 q% = FNfromdectalk%(0%)  
  \ if (q% <> DCS%) then  
  \ q% = FNfunny%("waiting for ring")  
  \ goto 10290  
  ! wait for ring  
  ! ok?  
  ! oops.  
  ! exit this  
10250 if (not FNtest%(R3.PH.OFFHOOK%)) then  
  \ q% = FNfunny%("expecting offhook")  
  \ goto 10290  
10260 DT.timeout% = 0%  
  \ DT.pending$ = ""  
  \ FNanswer% = TRUE%  
  ! No timeouts now  
  ! Nothing pending now  
  ! ok.  
10290 fnend  
10300 def* FNtimeout%(seconds%)  
  !  
  ! FNtimeout%(seconds%)  
  !  
  ! 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$) > 0%)  
  \ if (seconds% > 0%) then  
  \ q% = FNsync%  
  \ \ seconds% = 0% if (len(DT.pending$) > 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 (seconds% = DT.timeout%) ! Don't resend &
  "timeouts set "; seconds% if (DT.log%) &
  num1$(seconds%) &

10330 DT.timeout% = seconds% ! save timeout state &
  if (not FNptes%(R3.PH.OFFHOOK)) then &
  FNfunny$("timeout") &

10390 fnend &

10400 def* FNsync% &
  ! FNsync% &
  ! 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 moon 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%(VT$); ! Flush speech &
  q% = FNdocs%("11") ! Send sync &
  if (not FNmessage%("22", 32, -1)) then &
  FNfunny%("sync") &

10490 fnend &

10500 def* FNhangup% &
  ! FNhangup% &
  ! 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 &
  ! wait until &
  ! it's hung up &
  goto 10590 if (not FNphone%("")) &
  ! loop forever &
  FNhangup% = TRUE% ! OK now. &

10590 fnend &

10600 def* FNphone%(text$) &
  ! FNphone%(text$) &
  ! 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$ <> """) ! If extra parameters &
  then text$ = "60;" + text$ ! tack them on, else &
  else text$ = "60" ! just do status report &
10620  FNphone% = FNmessage%(text$, R2.PHONE%, -1%)  &
10690  fnend  &
10700  def* FNsave%(char%)  &
      FNsave%(char%)  &
      ! If the char% came from a user data entry, save it in &
      the DT.pending$ 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 (len(DT.pending$) > 0%)  \ Save 'T'
      \    \ char% = ascii('T')  &
10720  if (Instr(0%, "0123456789#ABCDT", chr$(char%)) = 0%)  &
      \ then FNsave% = FALSE%
      \ else DT.pending$ = DT.pending$ + chr$(char%)  &
10790  fnend  &
10800  def* FNkey%(timeout%)  &
      FNkey%(timeout%)  &
      ! Read a keypad character (in there is one in the &
      ! type-ahead buffer, or read a character or escape &
      ! sequence from DECTalk. The timeout% parameter is &
      ! non-zero to enable timeouts. &
      ! Note that the timeout parameter, if non-zero, will be &
      ! extended to compensate for RSTS/E output buffering. &
      ! FNkey% ignores user timeout if timeout was disabled. &
10810  q% = FNtimeout%(timeout%)  ! Set/clear timeouts  &
      \ if (len(DT.pending$) > 0%) then
      \    FNkey% = ascii(DT.pending$)
      \    \ DT.pending$ = right(DT.pending$, 2%)
      \    \ goto 10890  &
10820  timeout% = (timeout% * 4%) + 60% if timeout% > 0%  &
      \ q% = FNfromdectalk%(timeout%)  &
      \ q% = ascii('T')  if FNptest%(R3.PH.TIMEOUT%)  &
      \ DT.timeout% = 0%  if (q% = ascii('T'))
      \ q% = ascii('H')  if FNptest%(R3.PH.ONHOOK%)  &
      \ q% = ascii('E')  if (q% <= 0%)  ! O.S. error
      \ FNkey% = q%  &
10890  fnend  &
12000  def* FNmessage%(text$, t2%, t3%) &
      ! FNmessage$(text$, t2%, t3%) &
      ! 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 keypad 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% = FNdscs%(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%) &
      ! FNfromdectalk%(timeout%) &
      ! Read an escape sequence or keypad 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 12180 ! 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 &
12180  FNfromdectalk% = DT.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% &

12290 return &

12300 def* FNtoss% &
  \ ! &
  \ ! FNtoss% &
  \ ! 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%) &
  \ ! FNgetsequence%(timeout%) &
  \ ! Read the next character or the next ANSI standard &
  \ ! Escape Sequence. &
  \ ! Initialize by setting SEQ.state% to zero. Returns: &
  \ ! SEQ.state% &
  \ ! 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 CO 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/ESC &
  \ ! SEQ.state% zero when sequence ends. &
  \ ! SEQ.parm% number of parameters &
  \ ! SEQ.p%(1%) each parameter as read &
  \ ! SEQ.inter% intermediates &
  \ ! SEQ.private% private introducer, 'X' if error seen &
13010 \ DT.c% = fnget%(timeout%)  ! Get a character &
13020 \ 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 &
13030 \ 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% = CANX%)  ! or CTRL-U &
\ or (DT.c% = SUBX%) 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%) &
\ \ \ ! Mark "param" &
13070 \ goto 13010 ! Read another char. &
! We know the character is in the range '0'..'9' or
! '; (separator) or ':' (illegal separator)

SEQ.parm% = 1% if (SEQ.parm% = 0%) ! \ print #2%, "parm or sep: "; fnvisible$(dt.c%)
\ if (SEQ.inter$ < "") then ! No param's after
  ! intermediates.
  ! seq.private$ = "X"
  ! Mark it invalid.
  \ print #2%, "parm or separator after inter"

13090 if (DT.c% = ascPi('9')) then ! Parameter digit
  ! Make it a number
  SEQ.p% = (SEQ.p% * 10%) + (DT.c% - ascPi('0'))
  ! \ print #2%, "digit, param :="; seq.p%(SEQ.parm%)
  \ goto 13010 ! Go read another byte

13100 if (DT.c% = ascPi(';')) then ! parameter separator
  \ 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%)
! DT.c% = SEQ.state%
! 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$)
! ! FNsend%(text$)
! ! Send a string of text to DECTalk. Note, the text
! length must be less than the DECTalk terminal buffer
! size.
field #9%, len(text$) as q$
\set 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%, ''
\fend

\def* FNcs$1%(text$) = FNsend%(ESC$ + "[" + text$)
  FNcs$1% ( text$ )
  \Send a Control Sequence to DECTalk.

\def* FNdc$cs%(text$) =
  FNsend%(ESC$ + "P0;" + text$ + "z" + ESC$ + "\"
  FNdc$cs% ( text$ )
  \Send a DECTalk Device Control Sequence.
  \Note that the DECTalk P1 parameter, final, and
  \string terminator are automatically included.

\def* FNsp$eak%(text$) = FNsend%(text$ + CRLF$)
  FNline$e% ( text$ )
  \Send a line of text to DECTalk, followed by <CR><LF>

\def* FNget%(timeout$)
  FNget%( timeout$ )
  \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

while (DT.incount% >= DT.inend%)  ! None saved?
  \goto 15080 if (notFNread%(timeout$))

next
15030  field #8%, DT.incount% as q$, 1% as q$ ! get char &
    \ DT.incount% = DT.incount% + 1% ! step index &
    \ q% = asci1(q$) and 127% ! drop parity &
    \ goto 15010 if (q% = 0% or q% = 127%) ! ignore nulls &
    \ FNget% = q% ! return char &
    \ goto 15090 ! exit &
15080  FNget% = 0% ! got timeout &
15090  fnend &
15100  def* FNread%(timeout%) &
    ! FNread%(timeout%) &
    ! ! Read a record from DECTalk. &
    ! timeout% = 0% means none &
    ! timeout% > 0% wait timeout% seconds &
    ! timeout% = -1% return immediately if none &
    ! return FALSE% on timeout, fatal exit on other errors &
    ! return TRUE% on success. &
15110  goto 15120 if DT.incount% < DT.inend% ! Still stuff &
    \ on error goto 15150 ! grab error &
    \ q$ = sys(chr$(3%) + chr$(9%)) ! no echo &
    \ + sys(chr$(4%) + chr$(9%)) ! odt mode &
    \ wait timeout% if timeout% > 0% ! timeout &
    \ get #8% if timeout% >= 0% ! read buffer &
    \ get #8%, 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 #8%, DT.inend% as q$ &
    \ change q$ to q% &
    \ print #2%, FNvisible$(q%(q%)); &
    \ for q% = 1% to q%(0%) &
    \ print #2%, "" &
15120  FNread% = TRUE% &
    \ goto 15190 &
15150  resume 15180 &
    \ if (err = 15% and timeout% > 0%) &
    \ or (err = 13% and timeout% < 0%) &
    \ goto 19000 &
15180  FNread% = FALSE% &
15190  fnend &
16000  def* FNtest%(t2%, t3%) =
   (DT.char% = DCS%)
   ! Make
   and (DT.final$ = '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%
   !
   FNtest%(t2%, t3%)
   !
   ! 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%.
   !
16100  def* FNpptest%(t3%) =
   ! Test phone reply
   FNtest%(R2.PHONE%, t3%)
   !
   ! FNptest%(t3%)
   !
   ! Return TRUE% if the current reply R2% parameter
   ! is R2.PHONE% and the T3% matches R3%
   !
17000  def* FNfunny%(text$)
   !
   ! FNfunny%(text$)
   !
   ! Log an error message and dump the current reply.
   !
17010  error.count% = error.count% + 1
   \ if (DT.log%) then
   print #2 if (ccpos(2%) <> 0%)
   \ print #2”, "Illegal reply at ": text$; "."
   \ FNfunny% = FNdump%("")
17090 fnend
17100  def* FNdump%(text$)
   !
   ! FNdump%(text$)
   !
   ! Dump the current reply.
17110 if (DT.log%) then
  print #2%, "Last sequence read";
  \ print #2%, " at "; text$; 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%, ";";
      \ ((q% + 1%) < DT.param)
      \ next q%
    \ print #2%, DT.final$;
    \ print #2%, "<ST>"; if (DT.char% = DCS%)
  \ print #2%

17190 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% = 10%) then FNvisible$ = CRLF$
else if (c% = 11%) 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$
fnend

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
  !
  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%)\n
fnend

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
  !
  print #1%, prompt$; "<"; default$; ""> ? ";
  \ input line #1%, q$
  \ FNprompt$, q$ = cvt$$,(q$, 397%)
  \ FNprompt$, = cvt$$,(default$, 397%) if len(q$) = 0%

fnend

! F a t a l E r r o r T r a p

error% = err
\ error.line% = erl
\ resume 19100
! save error number
! and error line
! and take fatal exit

print
\ print 'Fatal Error '';
\ print error message
\ cvt$$,(right(sys(chr$(6%)+chr$(9%))
\ + chr$(error%)), 3%), 4%);
\ at line'; error.line% ! force new line
! print error message
! and line number

stop

goto 32767

end
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>
<thead>
<tr>
<th>Mnemonic</th>
<th>Escape Sequence</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA Primary</td>
<td>ESC [ 0 c</td>
<td>027 091 048 099</td>
</tr>
<tr>
<td></td>
<td>Request DECTalk to identify itself. See Table A-2 for reply.</td>
<td></td>
</tr>
<tr>
<td>DECAC1</td>
<td>ESC SP 7</td>
<td>027 032 055</td>
</tr>
<tr>
<td></td>
<td>Select 8-bit C1 control character reception (accept the high-order bit).</td>
<td></td>
</tr>
<tr>
<td>DECSTR</td>
<td>ESC [ ! p</td>
<td>027 091 033 112</td>
</tr>
<tr>
<td></td>
<td>Reset to power-up state.</td>
<td></td>
</tr>
<tr>
<td>DECTC1</td>
<td>ESC SP 6</td>
<td>027 032 054</td>
</tr>
<tr>
<td></td>
<td>Select 7-bit C1 control character reception (truncate the high-order bit).</td>
<td></td>
</tr>
<tr>
<td>DECTST</td>
<td>ESC [ 5 ; Pn y</td>
<td>027 091 053 059 *** 121</td>
</tr>
<tr>
<td></td>
<td>Initiate local self-tests. See Table A-6 for Pn parameters.</td>
<td></td>
</tr>
<tr>
<td>DSR Brief</td>
<td>ESC [ 5 n</td>
<td>027 091 053 110</td>
</tr>
<tr>
<td></td>
<td>Give a brief status report. See Table A-2 for replies.</td>
<td></td>
</tr>
<tr>
<td>DSR Extended</td>
<td>ESC [ n</td>
<td>027 091 110</td>
</tr>
<tr>
<td></td>
<td>Give an extended device status report. See Table A-2 for replies.</td>
<td></td>
</tr>
<tr>
<td>DT_DICT</td>
<td>ESC P 0 ; 4 0 z name sub ESC \ 027 080 048 059 052 048 122 ... 027 092</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Load user dictionary. See Table A-2 for replies.</td>
<td></td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Escape Sequence</td>
<td>Decimal Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>DT_INDEX</td>
<td>ESC P 0 ; 2 0 ; P3 z ESC \ 027 080 048 059 050 048 059 *** 122 027 092</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Insert index flag in text. P3 range is 0 to 32767 (sent as ASCII characters).</td>
<td></td>
</tr>
<tr>
<td>DT_INDEX_QUERY</td>
<td>ESC P 0 ; 2 2 z ESC \ 027 080 048 059 050 050 122 027 092</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Request DECTalk to return last index marker spoken. See Table A-2 for reply.</td>
<td></td>
</tr>
<tr>
<td>DT_INDEX_REPLY</td>
<td>ESC P 0 ; 2 1 ; P3 z ESC \ 027 080 048 059 050 049 059 *** 122 027 092</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Insert index flag in text and inform host. P3 range is same as for DT_INDEX. See Table A-2 for reply.</td>
<td></td>
</tr>
<tr>
<td>DT_LOG</td>
<td>ESC P 0 ; 8 1 ; P3 z ESC \ 027 080 048 059 056 049 059 *** 122 027 092</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set trace and debugging log functions. See Table A-7 for P3 parameters.</td>
<td></td>
</tr>
<tr>
<td>DT_MASK</td>
<td>ESC P 0 ; 8 3 ; P3 z ESC \ 027 080 048 059 056 051 059 *** 122 027 092</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Controls how DECTalk sends escape sequences and keypad characters to the host. See Table A-8 for P3 parameters.</td>
<td></td>
</tr>
<tr>
<td>DT_MODE</td>
<td>ESC P 0 ; 8 0 ; P3 z ESC \ 027 080 048 059 056 048 059 *** 122 027 092</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set DECTalk mode. See Table A-3 for P3 parameters.</td>
<td></td>
</tr>
<tr>
<td>DT_PHONE</td>
<td>ESC P 0 ; 6 0 ; Pn ; Pn z text ESC \ 027 080 048 059 054 048 059 *** 059 *** 122 ... 027 092</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Control attached telephone and telephone keypad interface. See Table A-5 for Pn parameters.</td>
<td></td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Escape Sequence</td>
<td>Decimal Value</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>DT_PHOTEXT</td>
<td>\texttt{ESC P 0 ; 0 z text ESC \ 027 080 048 059 048 122 *** 027 092}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Speak phonemic text.</td>
<td></td>
</tr>
<tr>
<td>DT_SPEAK</td>
<td>\texttt{ESC P 0 ; 1 2 ; P3 z ESC \ 027 080 048 059 049 050 059 *** 122 027 092}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enable (P3=1) or disable (P3=0) speaking.</td>
<td></td>
</tr>
<tr>
<td>DT_STOP</td>
<td>\texttt{ESC P 0 ; 1 0 z ESC \ 027 080 048 059 049 048 122 027 092}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stop speaking and dump any pending unspoken text.</td>
<td></td>
</tr>
<tr>
<td>DT_SYNC</td>
<td>\texttt{ESC P 0 ; 1 1 z ESC \ 027 080 048 059 049 049 122 027 092}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Finish speaking current text before processing next command.</td>
<td></td>
</tr>
<tr>
<td>DT_TERMINAL</td>
<td>\texttt{ESC P 0 ; 8 2 ; P3 z ESC \ 027 080 048 059 056 050 059 *** 122 027 092}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set local terminal mode. See Table A-4 for P3 parameters.</td>
<td></td>
</tr>
<tr>
<td>DECID</td>
<td>\texttt{ESC Z \ 027 090}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Old identify terminal request. Not recommended.</td>
<td></td>
</tr>
<tr>
<td>RIS</td>
<td>\texttt{ESC c \ 027 099}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset to power-up state.</td>
<td></td>
</tr>
<tr>
<td>S7C1T</td>
<td>\texttt{ESC SP F \ 027 040 070}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select 7-bit C1 control character transmission.</td>
<td></td>
</tr>
</tbody>
</table>
### Table A-1  Escape Commands (Cont)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Escape Sequence</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S8C1T</td>
<td>ESC SP G</td>
<td>027 040 071</td>
</tr>
</tbody>
</table>

Select 8-bit C1 control character transmission.

### Table A-2  DECTalk Status Replies

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Escape Sequence</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_DIPICT Reply</td>
<td>ESC P 0 ; 5 0 ; P3 z ESC \</td>
<td>027 080 048 059 052 048 059 *** 122 027 092</td>
</tr>
</tbody>
</table>

P3 parameters are as follows.

- 0  Word entered correctly.
- 048
- 1  No room in dictionary.
- 049
- 2  Entry too long.
- 050

| DA Primary Reply  | ESC [ ? 1 9 c  | 027 091 063 049 057 099 |

Reply from DECTalk to DA primary sequence.

| DSR Brief Replies | ESC [ 0 n     | 027 091 048 110 |

No malfunctions.

| ESC [ 3 n       | 027 091 051 110 |

Malfunction occurred.
Table A-2  DECTalk Status Replies (Cont)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Escape Sequence</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSR Extended Replies</td>
<td>ESC [ 0 n ESC [ ? 2 1 n</td>
<td>027 091 048 110 027 091 063 050 049 110</td>
</tr>
<tr>
<td></td>
<td>No malfunctions, first reply.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESC [ 0 n ESC [ ? 2 0 n</td>
<td>027 091 048 110 027 091 063 050 048 110</td>
</tr>
<tr>
<td></td>
<td>No malfunctions, second or later reply.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESC [ 3 n ESC [ ? Pn ; ... Pn n</td>
<td>027 091 051 110 027 091 063 *** 059 *** 110</td>
</tr>
<tr>
<td></td>
<td>Malfunction occurred. Pn parameter values are as follows.</td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>Communication failure.</td>
<td></td>
</tr>
<tr>
<td>050 050</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 3</td>
<td>Input buffer overflow.</td>
<td></td>
</tr>
<tr>
<td>050 051</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 4</td>
<td>Last NVR operation failed.</td>
<td></td>
</tr>
<tr>
<td>050 052</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 5</td>
<td>Error in phonemic transcription.</td>
<td></td>
</tr>
<tr>
<td>050 053</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 6</td>
<td>Error in DECTalk private control sequence.</td>
<td></td>
</tr>
<tr>
<td>050 054</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 7</td>
<td>Last DECTST failed.</td>
<td></td>
</tr>
<tr>
<td>050 055</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT_INDEX_QUERY Reply</td>
<td>ESC P 0 ; 3 2 ; P3 z ESC \</td>
<td>027 080 048 059 051 050 059 *** 122 027 092</td>
</tr>
<tr>
<td></td>
<td>P3 is the ASCII value of the last index spoken.</td>
<td></td>
</tr>
<tr>
<td>DT_INDEX_REPLY Reply</td>
<td>ESC P 0 ; 3 1 ; P3 z ESC \</td>
<td>027 080 048 059 051 049 059 *** 122 027 092</td>
</tr>
<tr>
<td></td>
<td>Reply sent by DECTalk after speaking indexed text. P3 is the ASCII value of the last index spoken.</td>
<td></td>
</tr>
</tbody>
</table>
Table A-3  DT_MODE Parameters

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE_SQUARE</td>
<td>1</td>
<td>Set MODE SQUARE on.</td>
</tr>
<tr>
<td>MODE_ASKY</td>
<td>2</td>
<td>Use the single-character phonemic alphabet.</td>
</tr>
<tr>
<td>MODE_MINUS</td>
<td>4</td>
<td>Speak hyphen - as &quot;minus.&quot;</td>
</tr>
</tbody>
</table>

Table A-4  DT_TERMINAL Parameters

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERM_HOST</td>
<td>1</td>
<td>Send all local terminal characters to host line.</td>
</tr>
<tr>
<td>TERM_SPEAK</td>
<td>2</td>
<td>Speak all characters typed on local terminal.</td>
</tr>
<tr>
<td>TERM_EDITED</td>
<td>4</td>
<td>Line edit all local terminal input.</td>
</tr>
<tr>
<td>TERM_HARD</td>
<td>8</td>
<td>Do local terminal editing in hardcopy format.</td>
</tr>
<tr>
<td>TERM_SETUP</td>
<td>16</td>
<td>Speak all characters when terminal is in set-up mode.</td>
</tr>
<tr>
<td>TERM_FILTER</td>
<td>32</td>
<td>Do not send DECTalk-specific escape sequences to the terminal.</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>ASCII Code</td>
<td>Function</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>PH_STATUS</td>
<td>0</td>
<td>Send a telephone status report.</td>
</tr>
<tr>
<td></td>
<td>048</td>
<td></td>
</tr>
<tr>
<td>PH_ANSWER</td>
<td>1</td>
<td>Enable telephone autoanswer.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Enable telephone autoanswer.</td>
</tr>
<tr>
<td></td>
<td>049</td>
<td>Enable telephone autoanswer.</td>
</tr>
<tr>
<td></td>
<td>048</td>
<td>Enable telephone autoanswer.</td>
</tr>
<tr>
<td>PH_HANGUP</td>
<td>1</td>
<td>Hang up the telephone and disable the keypad.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Hang up the telephone and disable the keypad.</td>
</tr>
<tr>
<td></td>
<td>049</td>
<td>Hang up the telephone and disable the keypad.</td>
</tr>
<tr>
<td></td>
<td>049</td>
<td>Hang up the telephone and disable the keypad.</td>
</tr>
<tr>
<td>PH_KEYPAD</td>
<td>2</td>
<td>Enable the telephone keypad.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Enable the telephone keypad.</td>
</tr>
<tr>
<td></td>
<td>050</td>
<td>Enable the telephone keypad.</td>
</tr>
<tr>
<td></td>
<td>048</td>
<td>Enable the telephone keypad.</td>
</tr>
<tr>
<td>PH_NOKEYPAD</td>
<td>2</td>
<td>Disable the telephone keypad.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Disable the telephone keypad.</td>
</tr>
<tr>
<td></td>
<td>050</td>
<td>Disable the telephone keypad.</td>
</tr>
<tr>
<td></td>
<td>049</td>
<td>Disable the telephone keypad.</td>
</tr>
<tr>
<td>PH_TIMEOUT</td>
<td>3</td>
<td>Send telephone status message after n seconds (Chapter 4).</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Send telephone status message after n seconds (Chapter 4).</td>
</tr>
<tr>
<td></td>
<td>051</td>
<td>Send telephone status message after n seconds (Chapter 4).</td>
</tr>
<tr>
<td></td>
<td>048</td>
<td>Send telephone status message after n seconds (Chapter 4).</td>
</tr>
<tr>
<td>PH_TONE.Dial</td>
<td>4</td>
<td>Dial the telephone by using Touch-Tone dialing.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Dial the telephone by using Touch-Tone dialing.</td>
</tr>
<tr>
<td></td>
<td>052</td>
<td>Dial the telephone by using Touch-Tone dialing.</td>
</tr>
<tr>
<td></td>
<td>048</td>
<td>Dial the telephone by using Touch-Tone dialing.</td>
</tr>
<tr>
<td>PH_PULSE.Dial</td>
<td>4</td>
<td>Dial the telephone by using pulse dialing.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Dial the telephone by using pulse dialing.</td>
</tr>
<tr>
<td></td>
<td>052</td>
<td>Dial the telephone by using pulse dialing.</td>
</tr>
<tr>
<td></td>
<td>049</td>
<td>Dial the telephone by using pulse dialing.</td>
</tr>
</tbody>
</table>
### Table A-6  DECTST Parameters

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>ASCII Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST_POWER</td>
<td>1, 049</td>
<td>Rerun power-up tests.</td>
</tr>
<tr>
<td>TEST_HDATA</td>
<td>2, 050</td>
<td>Run host port data loopback test.</td>
</tr>
<tr>
<td>TEST_HCONTROL</td>
<td>3, 051</td>
<td>Run host port control loopback test.</td>
</tr>
<tr>
<td>TEST_LDATA</td>
<td>4, 052</td>
<td>Run local port data loopback test.</td>
</tr>
<tr>
<td>TEST_SPEAK</td>
<td>5, 053</td>
<td>Speak a built-in message.</td>
</tr>
</tbody>
</table>

### Table A-7  DT_LOG Parameters

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_TEXT</td>
<td>1</td>
<td>Speak all ASCII text.</td>
</tr>
<tr>
<td>LOG_PHONEME</td>
<td>2</td>
<td>Log all spoken text in phonemic format.</td>
</tr>
<tr>
<td>LOG_RAWHOST</td>
<td>4</td>
<td>Log all text read from the host on the local terminal.</td>
</tr>
<tr>
<td>LOG_INHOST</td>
<td>8</td>
<td>Log all text read from the host on the local terminal.</td>
</tr>
<tr>
<td>LOG_OUTHOST</td>
<td>16</td>
<td>Log all text sent to the host on the local terminal.</td>
</tr>
<tr>
<td>LOG_ERROR</td>
<td>32</td>
<td>Log all error messages on the terminal.</td>
</tr>
<tr>
<td>LOG_TRACE</td>
<td>64</td>
<td>Log all escape sequences as readable text on the local terminal.</td>
</tr>
<tr>
<td>LOG_DEBUG</td>
<td>128</td>
<td>Reserved for DECTalk internal use.</td>
</tr>
</tbody>
</table>
### Table A-8  DT_MASK Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>512</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1024</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>11</td>
<td>#</td>
</tr>
<tr>
<td>4096</td>
<td>12</td>
<td>A</td>
</tr>
<tr>
<td>8192</td>
<td>13</td>
<td>B</td>
</tr>
<tr>
<td>16384</td>
<td>14</td>
<td>C</td>
</tr>
<tr>
<td>32768</td>
<td>15</td>
<td>D</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>2-Char. Symbol</th>
<th>1-Char. Symbol</th>
<th>Example</th>
<th>2-Char. Symbol</th>
<th>1-Char. Symbol</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vowels</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ey</td>
<td>e</td>
<td>bake</td>
<td>ah</td>
<td>*</td>
<td>but</td>
</tr>
<tr>
<td>aa</td>
<td>a</td>
<td>Bob</td>
<td>aw</td>
<td>W</td>
<td>bout</td>
</tr>
<tr>
<td>iy</td>
<td>i</td>
<td>beat</td>
<td>yu</td>
<td>Y</td>
<td>cute</td>
</tr>
<tr>
<td>eh</td>
<td>E</td>
<td>bet</td>
<td>rr</td>
<td>R</td>
<td>bird</td>
</tr>
<tr>
<td>ay</td>
<td>A</td>
<td>bite</td>
<td>ao</td>
<td>c</td>
<td>bought</td>
</tr>
<tr>
<td>ih</td>
<td>I</td>
<td>bit</td>
<td>ae</td>
<td>@</td>
<td>bat</td>
</tr>
<tr>
<td>oy</td>
<td>O</td>
<td>boy</td>
<td>uh</td>
<td>U</td>
<td>book</td>
</tr>
<tr>
<td>ow</td>
<td>o</td>
<td>boat</td>
<td>ix</td>
<td>l</td>
<td>kisses</td>
</tr>
<tr>
<td>uw</td>
<td>u</td>
<td>lute</td>
<td>ax</td>
<td>x</td>
<td>about</td>
</tr>
<tr>
<td>ir</td>
<td></td>
<td>beer</td>
<td>er</td>
<td></td>
<td>bear</td>
</tr>
<tr>
<td>ar</td>
<td></td>
<td>bar</td>
<td>or</td>
<td></td>
<td>bore</td>
</tr>
<tr>
<td>ur</td>
<td></td>
<td>poor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Consonants</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>p</td>
<td>pet</td>
<td>b</td>
<td>b</td>
<td>bet</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>test</td>
<td>d</td>
<td>d</td>
<td>debt</td>
</tr>
<tr>
<td>k</td>
<td>k</td>
<td>Ken</td>
<td>g</td>
<td>g</td>
<td>guess</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>fin</td>
<td>v</td>
<td>v</td>
<td>vest</td>
</tr>
<tr>
<td>th</td>
<td>T</td>
<td>thin</td>
<td>dh</td>
<td>D</td>
<td>this</td>
</tr>
<tr>
<td>s</td>
<td>s</td>
<td>sit</td>
<td>z</td>
<td>z</td>
<td>zoo</td>
</tr>
<tr>
<td>sh</td>
<td>S</td>
<td>shin</td>
<td>zh</td>
<td>Z</td>
<td>azure</td>
</tr>
<tr>
<td>ch</td>
<td>C</td>
<td>chin</td>
<td>jh</td>
<td>J</td>
<td>gin</td>
</tr>
<tr>
<td>w</td>
<td>w</td>
<td>wet</td>
<td>m</td>
<td>m</td>
<td>met</td>
</tr>
<tr>
<td>yx</td>
<td>y</td>
<td>yet</td>
<td>n</td>
<td>n</td>
<td>net</td>
</tr>
<tr>
<td>hx</td>
<td>h</td>
<td>head</td>
<td>nx</td>
<td>G</td>
<td>sing</td>
</tr>
<tr>
<td>l</td>
<td>l</td>
<td>let</td>
<td>en</td>
<td>N</td>
<td>button</td>
</tr>
<tr>
<td>r</td>
<td>r</td>
<td>red</td>
<td></td>
<td></td>
<td>silence</td>
</tr>
<tr>
<td>el</td>
<td>L</td>
<td>bottle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>em</td>
<td>M</td>
<td>ransom</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Allophones (Override DECTalk internal values.):**

- rx: r-oration
- lx: l-electric
- q: q-we eat
- dx: &-rider
- tx: Q-Latin
<table>
<thead>
<tr>
<th>2-Character Symbol</th>
<th>1-Character Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stress and Syntax Phonemic Symbols</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Primary lexical stress</td>
</tr>
<tr>
<td>.</td>
<td>'</td>
<td>Secondary lexical stress</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>Emphatic stress</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>Pitch rise</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>Pitch fall</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>Pitch rise/fall</td>
</tr>
<tr>
<td>Syntactic Structure Symbols</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>Syllable boundary (dash)</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>Morpheme boundary</td>
</tr>
<tr>
<td>#</td>
<td>#</td>
<td>Compound noun boundary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Word boundary (space)</td>
</tr>
<tr>
<td>(</td>
<td>(</td>
<td>Beginning of relative clause</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td>End relative clause, begin verb phrase</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>End of clause</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>End of sentence</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>End of question</td>
</tr>
<tr>
<td>!</td>
<td>!</td>
<td>End of exclamation</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>Paragraph introducer</td>
</tr>
</tbody>
</table>
### RELATED DOCUMENTATION
You can order the following DECTalk documents from Digital.

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECTalk DTC01 Owner's Manual (EK-DTC01-OM)</td>
<td>This manual gives an overview of DECTalk operations and a detailed description of DECTalk off-line (local) operations, phonemic codes, and spoken text conventions.</td>
</tr>
<tr>
<td>DECTalk DTC01 Programmer Reference Manual (EK-DTC01-RM)</td>
<td>This manual describes DECTalk-computer connections, DECTalk escape sequences, and programming methods for interfacing DECTalk with a host computer and telephone.</td>
</tr>
<tr>
<td>DECTalk DTC01 Programmer Reference Card (EK-DTC01-RC)</td>
<td>This card summarizes DECTalk phonemic codes, commands, and escape sequences.</td>
</tr>
<tr>
<td>DECTalk DTC01 Installation Manual (EK-DTC01-IN)</td>
<td>This manual explains how to install and operate DECTalk.</td>
</tr>
</tbody>
</table>
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

A

Abbreviations
  how indexing affects, 39
  in user dictionary, 43
ALGOL, 70
Allophones, 220
Alternate character sets,
  selecting, 27
ANSI standards, 22
Answering the phone, 48
Application programs
  BASIC-PLUS, 191
  C, 69
  DECTalk-specific codes, 73
  dialog, 15
  error codes, 72
  flags, 72
  guidelines, 15
  numeric encoding, 16
  SEQUENCE data structure
    in C, 78
  two-character encoding, 17

ASCII code tables
  7-bit, 23
  8-bit, 25
ASCII character sets,
  selecting, 20
ASCII escape sequences, 6
ASCII_G, 27
Audio delay, 38
Autoanswering telephone, 47

B

Backspace (BS) character, 11
BASIC-PLUS program, 191
Baud rate with XON/XOFF, 13
Buffer
  overflow, 13
  size, 13
  reset with DT_STOP, 38
C

C0 control characters, 26, 27
C1 control characters, 26, 27, 32
selecting, 21
C language, 70
C modules
  DECTLK.H, 83
  DEMO.C, 98
  DTANSW.C, 100
  DTCLOSE.C, 101
  DTCMD.C, 103
  DTDCHA.C, 105
  DTDCS.C, 106
  TDTIAL.C, 108
  DTDRAI.C, 111
  DTDUMP.C, 113
  DTEOL.C, 115
  DTGESC.C, 116
  DTGET.C, 123
  DTHANG.C, 125
  DTINIT.C, 126
  DTINK.E.C, 128
  DTIOGE.C, 130
  DTIOPU.C, 140
  DTISKE.C, 143
  DTISTI.C, 144
  DTISVA.C, 145
  DTKEEP.C, 146
  DTMSG.C, 147
  DTOFFH.C, 149
  DTONHO.C, 150
  DTOPEN.C, 151
  DTPEEK.C, 157
  DTPESC.C, 163
  DTPHON.C, 167
  DTPTES.C, 168
  DTPUT.C, 169
  DTREAD.C, 170
  DTRESE.C, 172
  DTSAVE.C, 173
  DTSPLICE.C, 175
  DTST.C, 177
  DTSYNC.C, 178
  DTTALK.C, 179
  DTTEST.C, 180
  DTTIME.C, 181
  DTTONE.C, 183
  DTTRAP.C, 185
  DTVISI.C, 188
  HELLO.C, 190

C program
  data structure, 78
  DECTalk commands, 73
  DECTalk replies, 75
  DECTalk-specific
    parameters, 73
  error codes, 72
  flags, 72
  logging command
    parameters, 77
  module list, 80-82 (see also C
    modules)
  self-test parameters, 76
  structure, 70
  telephone control
    parameters, 74
  variables, 72

Characters
  backspace, 9
  control, 8, 24 (see also C0 and
    C1)
  graph, 24
  hierarchy of, 9

Character sets, 27
  7-bit ASCII, 23-24
  8-bit, 25-26
  DEC multinational, 30-32
  DEC supplemental graphics, 32
  mapping, 20
  selecting alternate, 27
  selecting ASCII, 20
  speaking, 20
Clause boundary, 10, 36
   DT_SYNC as, 38
COBOL, 69, 70
Code table
    7-bit, 23
    8-bit, 25
Coding standards, 22
Command
    enter phonemic text, 37
    index query, 41
    index reply, 40
    index text, 40
    load dictionary, 43
    local log, 58
    local terminal, 62
    speak, 39
    stop speaking, 38
    synchronize, 38
    telephone management, 46
Commands
    arguments or parameters, 11
    DECTalk, 73
    ending sequences, 5
    invalid commands, 5
    telephone, 45
    voice, 35
Communication
    DECTalk-computer, 10
    DECTalk guidelines, 2
    setting up DECTalk, 12
    telephone, 45
Computer. See Host computer
Control character logging, 9
Control characters, 8, 24, 26
Controlling DECTalk, 2
Controlling DECTalk speech, 35
CR (carriage return), 8
CTRL-K, 9
CTRL-Q, 13
CTRL-S, 13

D
   DA primary, 51
   Data loss, 13
   Data paths, 10
      logging and debugging, 60
   Data synchronization, 13, 38
   Debugging, 58-61
   DECAC1, 21
   DECID, 52
   DECNVR (nonvolatile memory
      reset), 58
   DECESTR (soft reset), 39, 57
   DECTalk speech
      sentences and paragraphs, 36
   DECTC1, 21
   DECTST, 54
   DEL (delete), 8
   Delete user dictionary, 53, 56
   Device attribute request, 51
   Device attributes, 51
   Device self-test, 54
   Device status failure codes, 56
   Device status report, 55, 57
      brief report, 55
      extended report, 55
   Device testing, 52
   Dialing phone numbers, 47, 49
   Dictionary, user, 2, 43
      status report, 44
      deleting, 53, 56
   Discarding host data, 13
   DSR (device status report), 55
   DT_DICT, 43
   DT_INDEX, 40, 41
   DT_INDEX QUERY, 38, 40, 41
   DT_INDEX_REPLY, 40, 41
   DT_LOG, 58
      parameters, 59
   DT_MASK, 64
      parameters, 65
DT.MODE, 32
  parameters, 33
DTPHONE, 39, 45, 46, 48
  parameters, 47
DT.PHOTEXT, 37
DT.STOP, 38, 39
DT.SYNC, 38, 39, 41
DT.TERMINAL, 62
  parameters, 63

G

G0-G3 character sets, 27-29
GL (graphics left), 27, 32
GR (graphics right), 27, 32
Graphic characters, 24
  processing, 33

H

Hang up telephone, 47, 48, 53, 57
Hardware tests, 52
Host computer, 2, 11, 13
  commands (see the specific topic)
  setup, 12
Host-DECtalk interaction, 10, 42
Host line format, 53
Host line speed, 53
Host port tests, 54
HOSTSYNC, 13
HT (horizontal tab), 8
Hyphen, pronouncing a, 33

I

Identify terminal command, 52
Index
  defining an index, 40
  last index seen query, 41
  replying when an index is
    spoken, 40
Index query command, 41
Index reply command, 40
Index text command, 40
Indexing text, 39
Input buffer, 13
ISO standards, 22

E

Empty parameters, 7
Enable or disable speaking, 39
English, 2, 20
  rules for text, 36
Enter phonemic text command, 37
Error flags, 55, 56
ESC (escape), 8
Escape sequence. See also
  Command
  ASCII characters, 6
  decimal value, 7
  description, 2, 4-5
  format, 6
  mnemonic, 6
  parameters, 7
  summary list, 210-213
  terminator, 5

F

Factory settings, 53, 58
Firmware version level, 9
FF (form feed), 8
Flush pending text, 53
Foreign letters, 20
INDEX  229

K

Keypad characters
  sending,  46
  parameters,  47
Keypad mask command,  64
  parameters,  65

L

LF (line feed),  8
Line editing on terminal,  63
Load dictionary command,  43
Local line format,  53
Local line speed,  53
Local log command,  58
  parameters,  59
Local log flags,  53
Local port tests,  54
Local terminal command,  62
  parameters,  63
Local terminal flags,  53
LOG__DEBUG,  59
LOG__ERROR,  61
LOG__INHOST,  61
LOG__OUTHOST,  61
LOG__PHONEME,  61
LOG__RAWHOST,  61
LOG__TEXT,  61
LOG__TRACE,  61
Long sentences,  36
Loopback tests,  54
LS (locking shift) commands,  29

M

Maintenance commands,  51-67
Mapping 7-bit and 8-bit
  sets,  20-21
Marking text,  39

Memory,  58
Mnemonics,  6
MODE__ASKY,  33
MODE__MINUS,  33
Modes
  7-bit or 8-bit,  21
  off-line,  4
  on-line,  4
  operating,  4
  setup,  4
MODE__SQUARE,  2, 12, 33, 37
Multinational character set,  27
  as default,  32

N

Names and definitions
  variable,  72
NUL,  8
NVR (nonvolatile memory),  58

O

Operating features,  53
Operating modes,  4
Owner's manual,  3

P

Parameters
  DECTalk-specific,  73
  logging command,  77
  self-test,  76
  telephone control,  74
  values,  7
Parameters in escape
  sequences,  6
Pascal,  69
PH__ANSWER,  46, 48
  two status replies to,  48
PH__HANGUP, 48
PH__KEYPAD, 48
PH__NOKEYPAD, 48
Phonemes, 37
Phonemic alphabet, 219-221
Phonemic commands, 35, 37
Phonemic spelling, 35
   in abbreviations, 43
   recognizing errors in, 43
Phonemic text
   interpreting, 33
   speaking, 37
   using comments in, 37
Phone status, 47
PH__PULSE__DIAL, 49
PH__STATUS, 46, 47
PH__TIMEOUT, 48
PH__TONE__DIAL, 47, 49
Power-up status, 51
Product identification, 51
Program language and
   structure, 70
Programming
   considerations, 13
   escape sequence format, 6
   escape sequences, 4
Programs
   application, 80
Pronunciation
   changing, 33
   of foreign letters, 14
   using phonemics, 37
Public telephone network, 12
   automatic hangups, 49
PUP (power up), 51

R
R3__PH__OFFHOOK, 47
R3__PH__ONHOOK, 47, 48, 49
R3__PH__TIMEOUT, 46, 47, 48
Received characters, 2
   7-bit and 8-bit environment, 32
Replies
   DECTalk, 71
Reset, 53-54
RIS (reset to initial state), 39, 52, 56
Rules
   for DECTalk sequences, 4
   for text, 36

S
S7C1T control sequence, 21
S8C1T, control sequence, 21
Selecting active character
   sets, 29
Self-test, 52, 54
Sequences, 5
   ending, 5
Setup
   escape sequences, 19-33
   commands, 4, 12
Setup mode, 4
   speaking in, 63
   using BREAK key to enter, 4
Seven-bit mode, 20
Shift commands, 29
SI (shift in), 8
SO (shift out), 8
Soft terminal reset, 57
Source programs, ordering, 69
SP (space), 8
Speak
    command, 39
    foreign letters, 20
    phonemic text, 37
Speech
    changing rate of, 37
    commands that restart, 39
    control, 35
    enable or disable, 39
    stopping, 38
    timeout, 36
Square bracket commands, 2
SS2 (single shift 2), 29
SS3 (single shift 3), 29
Standards
    coding, 22
Status
    power-up, 52
Status report, 55
Status reporting, 52
Stop speaking command, 38
Stress marks, 221
SUB (substitute), 8
Switch-hook flash, 49
Synchronization
    data, 13
    DT SYNC command, 38
    XON/XOFF example, 14
Telephone, 45-50
    See also Phone
    example, 50
    keypad, 47, 48
    management command, 46
    replies, 47
    status messages, 47
Telephone control parameters, 74
TERM EDITED, 63
TERM FILTER, 63, 64
TERM HARD, 63
TERM HOST, 63
TERM SETUP, 63
TERM SPEAK, 63
Terminal and DECTalk, 2
Terminal commands, 63
Terminal identification, 52
TEST DATA, 54
TEST HCONTROL, 54
TEST HDATA, 54
TEST POWER, 54
TEST SPEAK, 54
Text, 36
Timeout, 10, 36, 48-49
Touch-Tone keypad, 44, 46, 49
Tracing, 58

U
Underlined text, 9
User dictionary, 2, 43
    deleting entries, 53, 56
    status report, 44

V
VT (vertical tab), 8

X
XOFF, 12, 13, 61
XON, 12, 13, 61