TABLE OF CONTENTS

INTRODUCTION

This disk comprises the Developer's Tools for the DoubleTalk PC and DoubleTalk LT voice synthesizers.

The information presented here, along with the accompanying support files, are intended to augment the DoubleTalk PC/LT User's Manual that comes with DoubleTalk - not as a replacement for it. For example, the command set for controlling the text-to-speech synthesizer is presented in the User's Manual, and hence is not duplicated here.

If you cannot find the answer to a nagging problem or question you have about DoubleTalk, please give our tech support people a call. They'd be happy to help you solve your unsolved mysteries.

OVERVIEW

DoubleTalk PC is an internal voice synthesizer for the IBM PC/XT/AT, PS2/25/30 and compatible computers, while DoubleTalk LT is an external, serial-driven version which can be used with virtually any platform. Both contain four voice synthesizers, enabling them to support virtually all of the common voice technologies in use today.

The text-to-speech (TTS) synthesizer is capable of producing unlimited, medium quality speech from plain English or Spanish ASCII text. The linear predictive coding (LPC) synthesizer is compatible with Texas Instruments' TSP5220 voice chip and most LPC data recorded for other LPC synthesizers. It features extremely low data rates for digitized speech - only 200 bytes per second (40:1 compression). The PCM/ADPCM and CVSD synthesizers enable DoubleTalk to play back digitized speech and sounds with very high sound quality. DoubleTalk also includes programmable tone generators (TGN), which can be used to produce audible prompts, music, signaling tones, or even dial a telephone, from your programs.

DoubleTalk is software compatible with RC Systems' V8600 and V8601 OEM voice boards, which share DoubleTalk's software command set. Programs written for one synthesizer will work with the other (except for LPC and CVSD-based speech, which the V860X synthesizers do not support).

Unlike most synthesizers on the market, DoubleTalk does not require any of the host computer's resources to operate (e.g., memory, IRQ's or CPU time). A built in 10 MHz 16-bit processor, 512K ROM and 8K RAM comprise a complete, self-contained text-to-speech conversion system. An application program need only write the ASCII text to be spoken to DoubleTalk, just as if it were a printer. Built in exception dictionary support allows the pronunciation of any word or character to be changed by the programmer or end-user. The rich command set allows the voice output to be tailored to suit any user's taste.

All DoubleTalks contain a unique, program-readable "silicon serial number." This enables a program to lock itself to a specific

DoubleTalk, providing a very secure form of software copy protection. A user can make as many copies of the program as he wishes, but each will still run on only one computer.

DISK CONTENTS

This disk contains the following programs and files:

```
TOOLS.DOC    - This file
DTPRN.COM    - Text-to-speech printer emulator
INT4DAPI.COM - IBM Speech Adapter BIOS emulator
COMPILE.COM  - Exception rule compiler
DTQB.LIB     - Basic/DoubleTalk support library (QB 4.5)
DTQBX.LIB    - Basic/DoubleTalk support library (PDS 7.x)
DTC.LIB      - C/DoubleTalk support library
DTDECLAR.BI  - Function/sub declaration file for DTQB(X).LIB
DTC.H        - Function prototypes for DTC.LIB
DTQB_EX.BAS  - Basic/DoubleTalk library demo program
DTC_EX.C     - C/DoubleTalk library demo program
LOADER.BAS   - Example Basic exception dictionary loader
TXTPHM.BAS   - Basic program referred to in DTQB_EX.BAS and DTC_EX.C
DT_INTGT.ASM - Assembly language interrogation routine (MASM)
DT_LPC.ASM   - Example assembly language LPC driver (MASM)
DT_PCM.ASM   - Example assembly language PCM driver (MASM)
DT_TTS.ASM   - Example assembly language TTS driver (MASM)
DT_XLT.ASM   - Assembly language phoneme translator (MASM)
WORDS2.LPC   - LPC data file used in DTQB_EX.BAS and DTC_EX.C
PORTAL.PCM   - ADPCM data file used in DTQB_EX.BAS and DTC_EX.C
GROOVY.TGN   - TGN data file used in DTQB_EX.BAS and DTC_EX.C
SPANISH.EXC  - Source for SPANISH.EXS (included on Utilities disk)
```

PRINTER EMULATOR

If you will only be using DoubleTalk's text-to-speech capabilities (and possibly the DTMF generator) in your application, the printer emulator driver, DTPRN.COM, is the easiest way to go. DTPRN is a small TSR program that enables DoubleTalk's TTS synthesizer to emulate any one of the computer's printer ports (LPT1-3) or communication ports (COM1-4). This enables most DOS and Windows 3.1 applications and programming languages to communicate with DoubleTalk by simply "printing" the text to be spoken to a printer port. DTPRN requires less than 1K of memory, and can be loaded into upper memory on machines supporting this feature using the DOS LOADHIGH command.

Installing DTPRN

DTPRN is invoked from the DOS command line by typing the command

```
DTPRN <port> [/N] [/Cx] [/R]
```

where <port> is one of the seven system ports listed above. For example, DTPRN can be installed as LPT2 by typing

```
DTPRN LPT2 (or LPT2:)
```

If no port specification is given, DTPRN will install itself as LPT3. During installation, DTPRN will locate and initialize DoubleTalk; if DoubleTalk cannot be located in the computer or on a serial port, DTPRN will abort the installation process and notify the user. Note that DTPRN (as well as all other software drivers supplied on the Utilities and Developer's Tools disks) looks for DoubleTalk LT first, before scanning the internal slots for DoubleTalk PC. This scheme enables an application program to work with the internal DoubleTalk PC when an LT is also present, by simply turning the LT's power off.

If an error occurs during the installation of DTPRN, such as the use of an invalid port name or DTPRN is already resident, an error code is returned in the AL register. This code can be processed in a batch program using the IF ERRORLEVEL command, so appropriate action can be taken. The following is a summary of the return codes returned by DTPRN:

```
0    DTPRN was successfully installed (no errors).
1    DTPRN is already resident.
2    An invalid port name was specified.
3    Too many or invalid parameter(s) (DTPRN still loads).
4    Incompatible version of DOS (must be 2.0 or later).
5    DoubleTalk could not be located.
```

During operation, DTPRN intercepts the appropriate BIOS software interrupt vector (14h for COM ports or 17h for LPT ports). Programs that bypass the BIOS hooks by accessing the port's hardware directly (as some terminal programs do) will not speak, since DoubleTalk will never receive the output characters.

Disabling DoubleTalk's Buffer

The optional command line switch /N is used to disable DoubleTalk's text buffer. This can be beneficial in applications where it is important that the voice stay synchronized with the text being read from the screen. (Note: If your application sends an exception dictionary to DoubleTalk via DTPRN, the buffer *must* be enabled while the dictionary is being output to DoubleTalk.)

Removing DTPRN From Memory

DTPRN can be removed from memory and the interrupt vector (discussed below) restored to its previous state by typing

```
DTPRN /R
```

Checking For DTPRN's Presence

An application program can check for the presence of DTPRN by examining the interrupt vectors. The word at 0:52h (int 14h) or 0:5Eh (int 17h) specifies the segment where DTPRN is located.

Address seg:103h contains the string "DTPRN Copyright (C) 1990 RC Systems, Inc." if DTPRN was installed successfully. The installed port name ("LPT1," "COM3," etc.) can be determined by examining the first four bytes at address seg:12Ch.

During initialization, DTPRN stores the I/O address of DoubleTalk's TTS synthesizer (normally 29Fh for DoubleTalk PC, or the base address of the COM port DoubleTalk LT is connected to) in the BIOS data area. This can be useful if your program needs to read or write to DoubleTalk directly, such as to determine if DoubleTalk is currently speaking (see the SYNC and SYNC2 flag descriptions in the "Hardware I/O" section, below). Exactly where the I/O address is stored in the BIOS data area depends on which port DTPRN is emulating, as the following table illustrates:

Port DTPRN installed as	Mem. address (abs. hex)
COM1	400
COM2	402
COM3	404
COM4	406
LPT1	408
LPT2	40A
LPT3	40C

For example, if DTPRN was installed as LPT1, location 40:8h will contain DoubleTalk's TTS I/O address (29Fh for DoubleTalk PC, if the jumper block hasn't been moved from the factory setting, or 3F8h for an LT connected to COM1). In the case of the internal PC card, an IN instruction (or its high-level language equivalent) to this address will return the TTS synthesizer's status flags. Likewise, text and commands can be output to either synthesizer with an OUT instruction or its equivalent.

Sending Text and Commands

Any combination of text and commands can be sent to DoubleTalk via DTPRN. Remember that DoubleTalk will not actually begin speaking until it receives at least one carriage return (0Dh) or null (00h), except when in Character mode. Only the text up to that point will be spoken.

A special feature of DTPRN is the way in which it handles DoubleTalk commands. DoubleTalk itself accepts only a control character (^A by default) for commands, which can be somewhat difficult, if not impossible, for a user to enter in some printing applications, such as a word processor. For this reason, DTPRN has been written to also accept an asterisk (*) as the command character, besides the standard ^A character. A so-called "asterisk command" must be followed by an alphanumeric character, '+', '-', or '@' to be considered a valid command by DTPRN. Examples of valid asterisk commands are:

```
*8s   *3F   *+10P
```

If the text being read contains asterisk characters, you may find DoubleTalk's voice changing unexpectedly, if the characters following an asterisk evaluate to a valid command. For this reason, DTPRN allows you to change its command character to any other printing character, with the optional /Cx command line switch. For example, /C& changes the command character from '*' to '&'. To disable DTPRN's command recognition altogether, use /C by itself. For example,

```
DTPRN LPT1 /N /C
```

installs DTPRN as LPT1, with the buffer and command recognition both disabled. Note that even when DTPRN's command character has been changed (or disabled), commands can still be issued to DoubleTalk using DoubleTalk's command character (^A).

Stopping Speech Production

Some application programs, such as screen readers for the blind, may require that the speech be stopped before everything in the input buffer has been spoken. This can be done quite easily by simply writing a ^X (18h) to DoubleTalk's TTS port, which immediately causes speech production to stop and the entire input buffer to be cleared. The synthesizer can also be stopped by using the appropriate BIOS Initialize function call (AH = 0 for int 14h or AH = 1 for int 17h). The user may also stop the speech by pressing both left and right Shift keys simultaneously.

DoubleTalk's current settings are *not* be affected by any of the above methods. (Note: It is recommended that the ^X character be output to DoubleTalk's TTS port directly using an OUT instruction, without performing handshaking of any kind. This allows the speech to be stopped even if the input buffer is full.)

BASIC LIBRARY

The Basic library routines supplied on this disk provide support for using DoubleTalk with Microsoft QuickBASIC and Professional Basic 7.x. In fact, the SmartTalk demo program, supplied on the Utilities disk, was written in QuickBASIC 4.5 and linked with this library. Fully documented Basic source code demonstrating each library function is included in the file DTQB_EX.BAS.

The routines included in the Basic (and C) libraries were written to be as granular as possible. This means that a Basic program using, for example, only the DTINIT and SAY functions will pull in only the modules associated with initializing DoubleTalk and using the TTS synthesizer from the library during linking. Unneeded modules, such as those associated with LPC and PCM support, will not be included in the final executable file, saving both disk space and memory.

Two functionally-identical versions of the Basic library are included: DTQB.LIB for use with QB 4.5, and DTQBX.LIB for Basic PDS 7.x (the only difference in the two libraries deals with far string

support). The following is a list of the functions provided in these libraries:

```
DTINIT       - Locates and initializes DoubleTalk
ROMVER$      - Returns DoubleTalk's ROM version number
SERNUM       - Returns DoubleTalk's serial number
DTSTS        - Returns the status of the TTS and LPC synthesizers
INTGT        - Returns the current settings of the TTS synthesizer
SAY          - Speaks a text string through the TTS synthesizer
XLT$         - Converts a text string to a phoneme string
TTSBUFON     - Enables the TTS synthesizer's input buffer
TTSBUFOFF    - Disables the TTS synthesizer's input buffer
LPC          - Speaks one or more words from an LPC data file
LPCBUFON     - Enables the LPC synthesizer's input buffer
LPCBUFOFF    - Disables the LPC synthesizer's input buffer
LPCSPD       - Controls the LPC synthesizer's output speed
LPCRPT       - Repeats the last LPC word or phrase
PCM          - Plays one or more sounds from a PCM/ADPCM data file
PCMRATE      - Controls the PCM playback speed
TONES        - Plays back a TGN data file (proprietary format)
TONES1       - Plays back a TGN data file (standard format)
```

C LIBRARY

The C function library provides a set of C functions similar to the Basic library. Fully documented C source code demonstrating each function's use is included in the file DTC_EX.C. The following is a list of the functions provided in the DTC.LIB library:

```
DT_Init      - Locates and initializes DoubleTalk
DT_RomVer    - Returns DoubleTalk's ROM version number
DT_SerNum    - Returns DoubleTalk's serial number
DT_Sts       - Returns the status of the TTS and LPC synthesizers
DT_Intgt     - Returns the current settings of the TTS synthesizer
DT_Say       - Speaks a text string through the TTS synthesizer
DT_Xlt       - Converts a text string to a phoneme string
DT_Lpc       - Speaks one or more words from an LPC data file
DT_LpcRpt    - Repeats the last LPC word or phrase
DT_Pcm       - Plays one or more sounds from a PCM/ADPCM data file
DT_Tgn       - Plays back a TGN data file (proprietary format)
DT_Tgn1      - Plays back a TGN data file (standard format)
```

DATA FILE FORMATS

The LPC, PCM and TGN functions in the Basic and C libraries utilize a common file format for their data files. If you intend to use your own data files with either library, you must be sure to adhere to these formats. Note that the SmartTalk program (included on the Utilities disk) utilizes the same formats, and may be used for testing your data files.

LPC File Format

The first word (two bytes) of an LPC data file indicates how many entries (words and/or phrases) are contained within the file, and information about the LPC data format. The lower 14 bits (0-13) contain the number of entries; therefore, up to 16,383 words and phrases can be stored in a single LPC data file. Bit 15 (FMT) indicates the type of LPC data stored in the file, where 0 = 5220 format, and 1 = D6 format. Bit 14 is reserved, and should be cleared to 0.

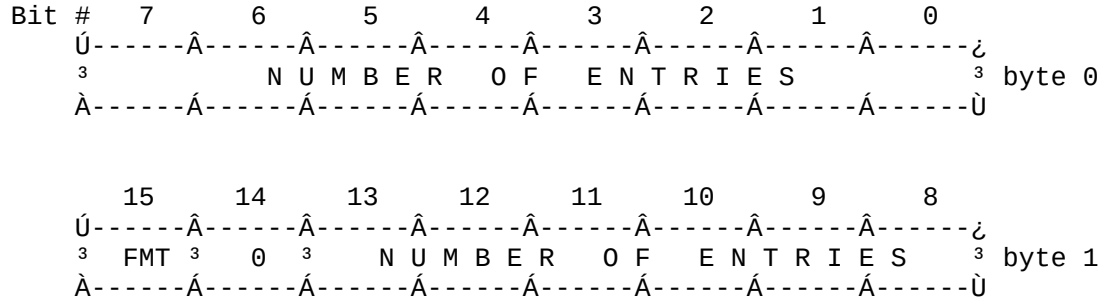

Figure 1. LPC Data File Header

Following the header word is the actual LPC data, in a length/data format, as shown below. This format was chosen because it is very easy to append new entries to the file (a table of pointers doesn't have to be maintained, as with other formats), and each entry may be up to 64K bytes in length.

```
bytes 0-1:     file header
bytes 2-3:     length of first entry
     .         first entry's data
     .               .
     .               .
     .         length of entry n
     .         entry n's data
```

The *.LPC files supplied on the Utilities and Developer's Tools disks use this format, and can be examined using DEBUG.

PCM File Format

The PCM data file format is similar to the LPC file format. The "number of entries" variable is eight bits in length, instead of 14. This limits the number of sounds that can be contained within a PCM data file to 255, but considering the inherent size of PCM sounds, this is actually quite a large number. The length variable for each entry is three bytes instead of two, allowing individual sound entries to be up to 16MB in length.

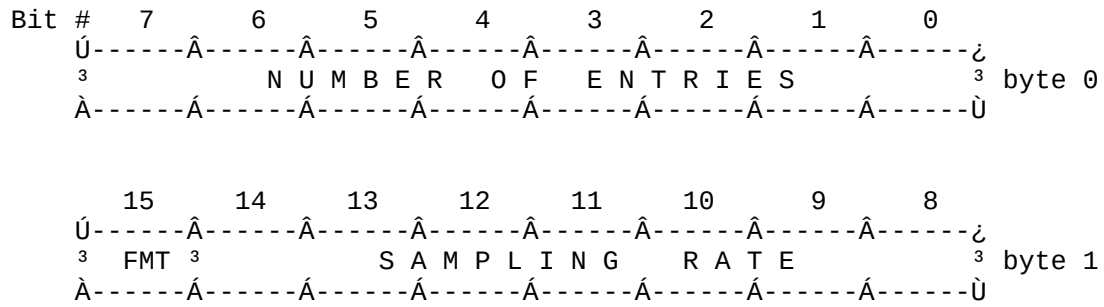

Figure 2. PCM Data File Header

The second byte of a PCM data file contains two types of information about the file: the rate at which the data was originally sampled, and the PCM data type. The low-order seven bits contain a number between 0 and 99 decimal, corresponding to the sampling rate variable in the PCM Mode command (see "PCM Mode" section, below). Note that the value stored here does not affect the actual playback rate in the libraries; it is for informational use only (the library PCM functions explicitly set the playback rate).

The high-order bit (FMT) should be 0 if the file contains linear PCM data, otherwise it should be set to 1 for ADPCM (compressed) data. This flag is very important, as it is responsible for invoking the correct decoding algorithm in the libraries. The PCM file format is summarized below.

```
bytes 0-1:     file header
bytes 2-4:     length of first entry
      .        first entry's data
      .              .
      .              .
      .        length of entry n
      .        entry n's data
```

The *.PCM files supplied on the Utilities and Developer's Tools disks use this format, and can be examined using DEBUG.

Sound conversion utility

CONVERT.COM is a DOS-based utility program used for converting eight-bit VOC (Soundblaster) and WAV (Windows) sound files to the PCM format used in the DoubleTalk libraries. Integrated into the program are ADPCM compression algorithms, used for reducing the storage requirements of sound files by a factor of two or more. The output of the utility is a file containing the converted PCM/ADPCM data, accompanied by a three-byte data length header (the format used by the Basic and C libraries).

The utility is run by typing the following on the DOS command line:

```
CONVERT <filename> [/C [/Sn]]
```

where <filename> is the name of the file to be converted. If the extension is omitted from the file name, "wav" is assumed by default. The optional C switch is used to invoke the ADPCM compression algorithms (otherwise, the output file will be linear PCM). Sn is an optional silence-encoding threshold parameter used in conjunction with the C switch, where n can be any value between 0 and 9 (default = 2).

By default, the output from the utility is a file which has the same name as the input file, but with the extension "pcm."

Silence threshold

The silence threshold parameter, Sn, sets the maximum level that is to be considered silence by the ADPCM encoding algorithms. The larger n is made, the higher the compression ratio will be. However, excessively high values of n can cause the decoded ADPCM output to become somewhat "grainy" sounding. The point at which this becomes objectionable depends on many factors, such as desired output quality vs. storage requirements, recording level, background noise, sound source, etc. Experimentation with this parameter is the best way to find the optimum setting. (Note that you may use a different silence threshold value for different sound files; the decoding algorithm can dynamically adapt to changing thresholds.) A silence threshold of 2 will be used if the threshold is not specified.

Creating library files

Multiple PCM sound files can be concatenated together for use with the Basic and C libraries with the DOS COPY command. For example, to combine the files file1, file2, and file3 into a single file file4, type

```
COPY /b hdr+file1+file2+file3 file4
```

The hdr file is a two-byte header "template," supplied on this disk, used for reserving room in the target file for the header required by the libraries.

TGN File Format

There are actually two TGN file formats, the simplest of which will be discussed here. This format is supported by the Basic library TONES1 and C library DT_Tgn1 functions. The more complex, proprietary format is supported by the TONES and DT_Tgn functions, and is used primarily for playing full-length songs. SmartTalk does not use or support the use of the simpler format discussed here.

The first two bytes of a TGN file indicate how many Voice frames (*not* bytes) are contained within the file (see "Musical Tone Generator," below, for a discussion of tone generator Voice frames). This value must be an integer between 0 and 16,383. The actual Voice frame data immediately follows these two bytes. A TGN data file contains no amplitude or tempo information; these values are passed by the calling program to the TGN functions as parameters.

As an example, suppose a short tune consisted of 12 voice frames. The first word in the TGN file would be set to a value of 12 (because there are 12 Voice frames), followed by 48 bytes (12 frames x 4 bytes/frame) of data. The GROOVY.TGN file supplied on this disk uses this format, and can be examined using DEBUG.

ASSEMBLY LANGUAGE API

The INT4DAPI.COM program provides a set of DoubleTalk function calls, similar to DOS' int 21h service calls. It also emulates the BIOS int 4Dh driver built into the IBM Speech Adapter. This implementation is actually a subset/superset of the Speech Adapter's, since DoubleTalk does not support CVSD recording (only playback). It does, however, support the foreground and background LPC playback and CVSD playback (32 kbps) functions, as well as some new functions including PCM/ADPCM playback. The driver was originally written so that programs utilizing these voice technologies could work with DoubleTalk, but you may also find it useful for your own programming. INT4DAPI can be run in high memory using the DOS LOADHIGH command.

The methods for detecting the presence of INT4DAPI, detecting installation errors, and removing the driver from memory are similar to that of the DTPRN driver, described elsewhere. The following is a summary of the return codes returned by INT4DAPI:

0	INT4DAPI was successfully installed (no errors).
1	INT4DAPI is already resident.
2	Reserved.
3	Too many or invalid parameter(s) (INT4DAPI still loads).
4	Incompatible version of DOS (must be 2.0 or later).
5	DoubleTalk could not be located.

Function Calls

This section describes the functions supported by INT4DAPI. Those functions which are extensions or additions to the original IBM Speech Adapter BIOS functions are denoted by an asterisk (*).

All INT4DAPI functions are invoked by loading the AH register with the desired function number, AL with the subfunction number (if applicable), and calling software interrupt 4Dh (int 4Dh). On return, the AL register will contain one of the following codes:

0	No errors encountered
1	Undefined command
2	Speech in progress

Function 00h	Reset

The Reset function initializes the LPC functions for 5220 data and normal playback speed; no other function is performed.

```
Ú-------------------------------------------------------------------¿
³ Function 01h     CVSD                                            ³
À-------------------------------------------------------------------Ù

   AL = 01h: Playback using speed table
        DS:SI   = segment:offset of CVSD data buffer
           CX   = length in bytes
           BL   = speed table (ignored)

   AL = 03h: Playback using user-defined speed
        DS:SI   = segment:offset of CVSD data buffer
           CX   = length in bytes
           BX   = user speed divisor (ignored)


Ú-------------------------------------------------------------------¿
³ Function 02h     LPC (Background)                                ³
À-------------------------------------------------------------------Ù

   AL = 00h: LPC status

   AL = 02h: LPC playback
        DS:SI   = segment:offset of LPC data buffer
           CX   = length in bytes

   AL = 03h: LPC format/speed *
           BL   = format code:
              0 = 5220 data/normal (default)
              1 = 5220 data/fast
              2 = D6 data/normal
              3 = D6 data/fast

The LPC Background mode is simulated by utilizing the LPC data
buffer within DoubleTalk (no interrupt processing takes place,
however, as with the Speech Adapter). The 4096 byte limit of the
data buffer in the Speech Adapter does not apply to INT4DAPI.
Subfunction 3 sets the data format and playback speed for all
subsequent subfunction 2 calls.


Ú-------------------------------------------------------------------¿
³ Function 03h     LPC (Foreground)                                ³
À-------------------------------------------------------------------Ù

   AL = 00h: LPC status

   AL = 02h: LPC playback
        DS:SI   = segment:offset of LPC data buffer
           CX   = length in bytes

   AL = 03h: LPC format/speed *
           BL   = format code:
              0 = 5220 data/normal (default)
              1 = 5220 data/fast
              2 = D6 data/normal
              3 = D6 data/fast
```

The LPC Foreground mode is simulated by disabling the LPC data buffer within DoubleTalk. The 4096 byte limit of the data buffer in the Speech Adapter does not apply to INT4DAPI. Subfunction 3 sets the data format and playback speed for all subsequent subfunction 2 calls.

```
Ú-------------------------------------------------------------------¿
³ Function 04h     PCM                                            * ³
À-------------------------------------------------------------------Ù

   AL = 00h: PCM playback
        DS:SI   = segment:offset of PCM data buffer
           CX   = length in bytes
           BL   = sampling rate

   AL = 01h: ADPCM playback
        DS:SI   = segment:offset of ADPCM data buffer
           CX   = length in bytes
           BL   = sampling rate
```

This function provides support for both PCM and ADPCM speech. Note that the sampling rate parameter passed in register BL may take on any value between 0 and 99d, corresponding to the 100 sampling rates supported by DoubleTalk (e.g., BL = 78d programs a playback rate of 8 kHz). See the section "PCM Mode" below for more information.

```
Ú-------------------------------------------------------------------¿
³ Function 05h     TTS                                            * ³
À-------------------------------------------------------------------Ù

   AL = 00h: TTS status
           AH   = TTS status flags

   AL = 01h: Speak
        DS:SI   = segment:offset of TTS data buffer
           CX   = length in bytes
           AH   = TTS status flags

   AL = 02h: Silence output
           AH   = TTS status flags
```

All TTS functions return the TTS synthesizer's status flags in register AH (see "I/O Ports, TTS Port," below, for a complete description of these flags). Any combination of text and commands may be included in the TTS data buffer for subfunction 1. The tone generators can also be used with this function by simply embedding the appropriate command in the buffer.

HARDWARE I/O

There may be instances when you prefer to communicate directly with DoubleTalk, or there's just no other way to get the job done (such as using DoubleTalk's indexing feature). This section gives you the necessary information to enable you to write your own DoubleTalk-compatible drivers.

DoubleTalk PC

All DoubleTalk PC functions are carried out by merely reading and writing to two I/O ports, located on the DoubleTalk card. In fact, DoubleTalk PC is virtually transparent to the host system in that it requires no memory-based text-to-speech software (unlike most other internal synthesizers), requires no CPU overhead, IRQs nor DMA.

I/O ports

From a programmer's point of view, DoubleTalk PC is simply two adjacent, eight bit I/O ports residing in the system memory map. One port, called the LPC port, is used for operating the LPC synthesizer and reading index markers. The second port, called the TTS port, is used for all other functions: text-to-speech, tone generation, PCM output, etc.

The six-position jumper block on the DoubleTalk card determines where the two I/O ports are mapped into the system. The jumper sets the base address, which is always the LPC port address. The TTS port address is always one higher (base+1) than the LPC port. The section "Detecting DoubleTalk" shows how to determine what these addresses are.

CAUTION Whenever DoubleTalk PC updates the data at either port, there is an inherent "settling" time for the bits to stabilize. Although this is on the order of only a few nanoseconds, it becomes significant with today's faster machines. If you are monitoring a port for a change in value, and also need the value (such as a new index marker coming in on the LPC port), you should read the port a second time (after the change was detected), in order to ensure reading a valid value.

LPC port

The LPC port is normally used only with the LPC synthesizer (except when reading index markers from the TTS synthesizer; see "Indexing," below). It accepts standard LPC-10 speech data, in both 5220 and D6 data formats.

Before sending any LPC data, a Speak command byte must be issued to the LPC synthesizer. The Speak command determines which LPC format

decoding table will be used by the hardware (5220 or D6), as well as the speech rate. The four Speak commands are:

Cmd	Format	Speed
60h	5220	normal
64h	5220	fast
20h	D6	normal
24h	D6	fast

Reading the LPC port yields the following status information:

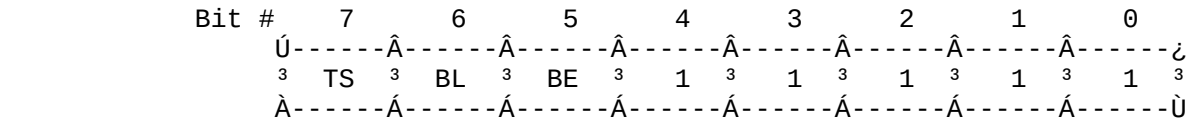

Figure 3. LPC Port Status Flags

Bit 7 TS - Talk Status. When set to 1, indicates the LPC synthesizer is producing speech.

Bit 6 BL - Buffer Low. When set to 1, indicates that the hardware LPC data buffer has less than 30 bytes remaining. (Total internal buffer size = 4096 bytes.)

Bit 5 BE - Buffer Empty. When set to 1, indicates that the LPC data buffer ran out of data (error condition if TS is also 1).

Bits 4-0 Reserved

The TS bit can be used to effectively disable the internal hardware LPC buffer by waiting for it to drop to 0 before returning to the application program.

The LPC repeat function is invoked by issuing the Speak command by itself, without any LPC data. The command used does not necessarily have to be the same as that used originally, although the data format (5220 or D6) must be the same. For example, a program may use the fast speed (64h) normally, and upon the user's request repeat the last word or phrase at normal speed (60h).

See the DT_LPC.ASM file for a working example of how to send LPC data to the LPC synthesizer.

TTS port

The TTS port handles all I/O between an application program and the TTS, PCM and CVSD synthesizers and tone generators. Regardless of the operating mode being used, data is written to the TTS port in

exactly the same manner, be it text, PCM audio samples, tone generator values, etc. DoubleTalk knows how to interpret the data based on a "mode command" you give it before sending the data (the default mode is always TTS). TTS commands that control TTS voice attributes, such as speed and pitch, are also written in the same way, allowing them to be embedded in text for dynamic control of the voice output.

Reading the TTS port returns several status flags which indicate when DoubleTalk is able to accept another data byte, indicate when it is talking, as well as the status of the TTS input buffer. The bit definitions are described below.

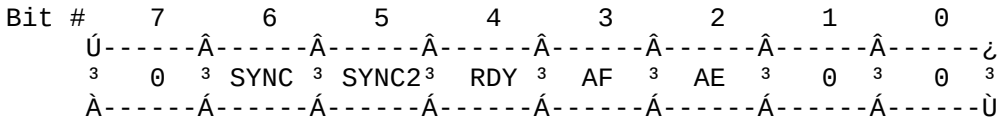

Figure 4. TTS Port Status Flags

Bit 7 Reserved

Bit 6 SYNC - Sync. When set to 1, this bit indicates that DoubleTalk is producing output (other than in LPC mode, which has its own talk status bit). SYNC drops to 0 immediately after output has ceased.

Bit 5 SYNC2 - Early Sync. Same function as SYNC, except drops to 0 up to 0.4 second earlier when using the TTS synthesizer (only). This enables a program to send another phrase to the TTS synthesizer just before the current one has completed, improving the response time.

Bit 4 RDY - Ready. When set to 1, indicates the TTS port is ready to accept a byte of data.

Bit 3 AF - Almost Full. When set to 1, indicates that less than 300 bytes are available in the TTS input buffer. AF is always 0 in the PCM, TGN and CVSD modes.

Bit 2 AE - Almost Empty. When set to 1, indicates that less than 300 bytes are remaining in DoubleTalk's input (TTS or PCM) buffer. AE is always 1 in the TGN and CVSD modes.

Bits 1-0 Reserved

Note that the SYNC flag can be used to effectively disable the internal text/tone/PCM data buffers by waiting for it to drop to 0 before sending any more data to the synthesizer. See the DT_TTS.ASM file for a working example of how to send data through the TTS port.

To avoid losing data, your program should test the RDY flag before each byte is output. The DT_Tts routine included in the DT_TTS.ASM

file shows the recommended method of doing this. The only exception to this rule is when stopping the TTS synthesizer with the ^X character - in this case, the RDY flag should be ignored in order to yield the fastest response time possible, as well as avoid potentially long delays should the input buffer become full.

DoubleTalk LT

DoubleTalk LT operates from a serial port at 9600 baud, 8 data bits, 1 stop bit, and no parity (9600,N,8,1). All DoubleTalk functions must be carried out through this port; there are no LPC or TTS ports to perform low-level I/O through.

DoubleTalk LT indicates when it is ready for data using the DTR hardware handshake protocol. Text, commands, and PCM/TGN data are sent to DoubleTalk by simply writing them to the serial port's transmit register. The source code in the DT_TTS.ASM file provide examples of how to communicate with DoubleTalk LT at the register level.

Using DoubleTalk LT's LPC synthesizer is quite different than DoubleTalk PC's. This is because the LT emulates several of the external Echo synthesizers (for compatibility with the Apple II and Macintosh platforms, for which the LT was originally designed). LPC driver source code, compatible with both the PC and LT, can be found in the file DT_LPC.ASM.

PCM mode with the LT is a bit convoluted, because at a sampling rate of 8 kHz (64 kbps), it would require more than eight seconds at 9600 baud to transmit each second's-worth of speech! The data transfer rate problem is overcome by automatically kicking the baud rate up to 115,200 baud while in the PCM/ADPCM modes.

If you need to support the LPC and/or PCM modes of both DoubleTalks, you may save yourself a lot of time by using the INT4DAPI driver, described elsewhere. This driver takes care of all the details of finding DoubleTalk, getting data to it, ADPCM decoding, etc. The Basic and C libraries, DTQB.LIB, DTQBX.LIB, and DTC.LIB, take care of the housekeeping for you as well in the Basic and C environments. (If you *really* must see PCM source code, see the file DT_PCM.ASM, on this disk.)

Reading the LT's SYNC flag

DoubleTalk PC has a convenient means of detecting when it is talking (see "I/O Ports, TTS Port," above, for a description of its two SYNC flags). DoubleTalk LT conveys this information in two ways, one using a COM port status bit (the "hardware" method), the other utilizing a sort of "software handshake" protocol. The Protocol Options command, described below, determines which method will be used.

The hardware sync method is usually the easiest to implement, and is therefore the recommended method to use. The LT's sync information is carried on the serial port's RI (Ring Indicator) line, and will

read zero (0) when SYNC is *true* (active). The following code fragment illustrates this:

```
     mov    dx,TTS_Port       ; LT's COM port base address
     add    dx,6              ; Modem Status Register
     in     al,dx             ; read status
     test   al,40h            ; test RI (SYNC)
     jz     Talking           ; zf = 1 if SYNC = 1
     jnz    Silent            ; zf = 0 if SYNC = 0
```

Protocol Options command

This command controls several internal operating parameters in DoubleTalk. The command is ^AnG, where n is an ASCII decimal number between 0 and 63. The number is calculated by ORing together the individual control bits, shown below. For example, the command 9G enables software synchronization with Sync2 timing. There are no restrictions in the way parameters can be combined; for example, you can enable software and/or hardware synchronization, or disable both of them. Note that bits 4 and 5 are the only bits that have any meaning to DoubleTalk PC. 2G is the default setting.

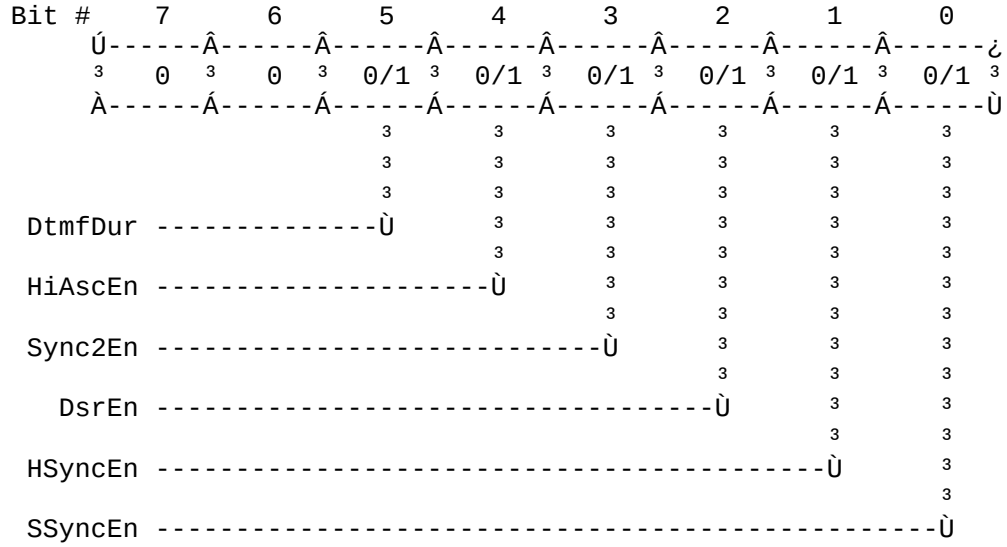

Figure 5. Protocol Options Control Bits

The control bits are defined as follows:

Bits 7-6 — Reserved (write "0" to ensure future compatibility).

Bit 5 — DtmfDur - DTMF Duration. Selects DTMF (Touch-Tone) generator burst duration (n* command). When set to 1, tone bursts will be 500 ms duration instead of 100 ms. Default: 100 ms (0).

Bit 4 — HiAscEn - High ASCII Enable. When set to 1, enables extended ASCII character set (128d - 255d) recognition. Generally

used only in conjunction with an exception dictionary for a foreign language. When disabled, the high-order bit of all ASCII text is masked. Default: Disabled (0).

Bit 3 Sync2En - Early Sync Enable. Enables Sync2 timing when set to 1. If enabled, software and hardware sync endings occur approximately 0.4 second early. If disabled, sync endings coincide with actual ending of speech production. This control bit corresponds to choosing between the SYNC and SYNC2 flags found in DoubleTalk PC (see TTS port flag definitions, above). Default: Disabled (0).

Bit 2 DsrEn - DSR Enable. Enables DSR handshaking when set to 1. If enabled, DoubleTalk will not transmit data to the host if the DSR serial status line is false, except for index markers, where DSR is ignored. Default: Disabled (0).

Bit 1 HSyncEn - Hardware Sync Enable. Enables hardware synchronization. If enabled, DoubleTalk carries sync status on the RI (Ring Indicator) serial status line. End timing is affected by Sync2En. Default: Enabled (1).

Bit 0 SSyncEn - Software Sync Enable. Enables software synchronization. If enabled, DoubleTalk transmits ASCII "B" when speech begins, and "E" when speech ends. End timing is affected by Sync2En. Default: Disabled (0).

Detecting DoubleTalk

Because it is generally unknown which DoubleTalk any particular system has, if any, and which port addresses or COM port are used, the first thing a DoubleTalk driver must do is determine if DoubleTalk is present, and where.

The DoubleTalk drivers and libraries supplied on this disk all incorporate a common routine for determining if DoubleTalk is present, and where it is located. The source code for this routine can be found in the DT_TTS.ASM file (DT_Init). This routine determines what the LPC and TTS I/O port addresses are if DoubleTalk PC was detected, or the base address of the COM port if DoubleTalk LT was detected. The routine also initializes DoubleTalk to the factory default settings.

CAUTION The method used to detect DoubleTalk PC assumes that the TTS and LPC ports will have certain specific values when they are read, forming a "signature" that is unique to DoubleTalk PC. If DoubleTalk happens to be talking at the time DT_Init looks for it, it will not be able to locate DoubleTalk, since one or both of the ports will have status bits that are in the "wrong" state. For similar reasons, DoubleTalk LT must also be idle in order for it to be detected.

Indexing

Index markers are non-speaking "bookmarks" that are used to monitor the progress of speech. This way, a program can tell where DoubleTalk is speaking within a passage of text.

The command to send an index marker to either DoubleTalk is ^AnI, where n is an ASCII number between 0 and 99. Thus, up to 100 unique markers may be active at any given time. If your program uses incrementing markers between each word, a simple way to implement this is to use DoubleTalk's "relative parameter" capability, by issuing the command ^A+1I for each marker. When the count reaches 99, it will automatically wrap around to 0 and start over again.

When DoubleTalk has spoken the text up to a marker, it transmits the marker number back to the program. Note that this value is a *binary* number between 0 and 99, *not* an ASCII number as was used in the command to place the marker. This allows the marker to be transmitted as a one-byte value.

The actual method of transmission and how markers are received, is where the two DoubleTalks part ways.

DoubleTalk PC

DoubleTalk PC transmits index markers through its LPC port. As long as the LPC synthesizer is idle, the value at the port will be 7Fh. If an index marker comes in, however, the port's value will change to a value between 0 and 63h (0-99d), and will not change until another marker comes (which will overwrite the previous one).

If you use only one marker value in your program, the value FFh should be written to the LPC port each time a new marker is read. This "clears" the marker (to 7Fh), enabling your program to detect when a new marker comes in. (If you use more than one marker value, such as two alternating values, this step would not be necessary.) Clearing the TTS synthesizer with any of the following TTS commands (through the TTS port) will also clear the marker from the LPC port:

```
^X (Clear)
^A@ (Reinitialize)
```

Clearing the marker in this manner also resets the relative index value, ensuring that relative indexing always begins at 0.

It is also good practice before quitting your application, that you issue one of these commands (preferably ^A@) to force the LPC status back to its normal (idle) state. Otherwise, the next application may not be able to find DoubleTalk (since most "find" routines expect the LPC port status value to be 7Fh).

DoubleTalk LT

DoubleTalk LT transmits index markers via its serial port. Normally, an interrupt handler in your program that processes incoming

characters from the COM port will handle the markers. Since the interrupt handler is invoked only when a new index marker comes in, it is not necessary to "clear" the marker, as with DoubleTalk PC.

CAUTION The Echo-emulation feature of DoubleTalk LT recognizes 40h ('@') as a request for the LPC synthesizer's status. This causes a one-byte value to be emitted from the serial port whenever '@' is transmitted to DoubleTalk LT. Take precautions to ensure that your program does not interpret this as a spurious marker - this can be done either by filtering any '@' characters from the text, or by checking that each received marker is within limits (0 to 99). The LPC status byte is generally a much larger number (usually 251), far outside the range of valid markers.

Not all DoubleTalks support indexing!

Early DoubleTalks *PCs* did not support indexing (all LT models do). To prevent your application from hanging up by waiting for index markers that may never come, it should first determine if the DoubleTalk card it is using supports indexing. The easiest way to determine this is to send it an index marker followed by a return or null character. If nothing comes back (i.e., the LPC port doesn't change in value from 7Fh), the card doesn't support indexing. Earlier version DoubleTalk PCs can be updated to support indexing with a simple user-installable ROM upgrade.

Interrogating DoubleTalk

The ^A? command built into both DoubleTalks enables a program to read the current settings of DoubleTalk anytime. The values returned from the Interrogate command are, in order, as follows:

Serial number	word; 0-7Fh:0-7Fh
ROM version	variable length string terminated by CR
Mode	byte; C/D/T; 0=Character; 1=Phoneme; 2=Text
Punc level	byte; nB; 0-7
Formant freq	byte; nF; 0-9
Pitch	byte; nP; 0-99
Speed	byte; nS; 0-9
Volume	byte; nV; 0-9
Tone	byte; nX; 0-2
Expression	byte; nE; 0-9
Exc dict loaded	byte; L; 1=exception dictionary loaded; 0 otherwise
Exc dict status	byte; U; 1=exception dictionary enabled; 0 otherwise
Free RAM	byte; L; # pages (truncated) remaining for text buffer - function of dictionary size and RAM chip installed (8K/32K)
Articulation	byte; nA; 0-9

Reverb	byte; nR; 0-9	
End of block	byte; 7Fh value indicating end of parameter block	

The number of parameters could change in the future, so it is recommended that the End of Block byte (7Fh) be used for determining when all of the data has been read from DoubleTalk.

The file DT_INTGT.ASM shows how to interrogate DoubleTalk from assembly language programs. The supplied Basic and C libraries also have their own equivalent interrogation functions built in.

MUSICAL TONE GENERATOR

DoubleTalk contains a three-voice musical tone generator, which can be used for creating music and sound effects. This section explains how to program the generator. DoubleTalk also contains a sinusoidal generator, which is covered in the next section.

The musical tone generator is activated with the ^AJ command. Once activated, all data output to DoubleTalk is directed to the tone generator. (Note: DoubleTalk assumes that tone generator data will immediately follow the ^AJ command; therefore, do not terminate the command with a carriage return or null.)

The tone generator is controlled by four, four-byte data and command frames, known as Initialize, Voice, Play, and Quit. With these, the programmer can control the volume, duration, and frequency of each of the three voices.

Initialize Command

The Initialize command sets up the tone generator's amplitude and tempo (speed). The host must issue this command to initialize the tone generator before sending any Voice frames. The Initialize command may, however, be issued anytime afterward to change the volume or tempo on the fly.

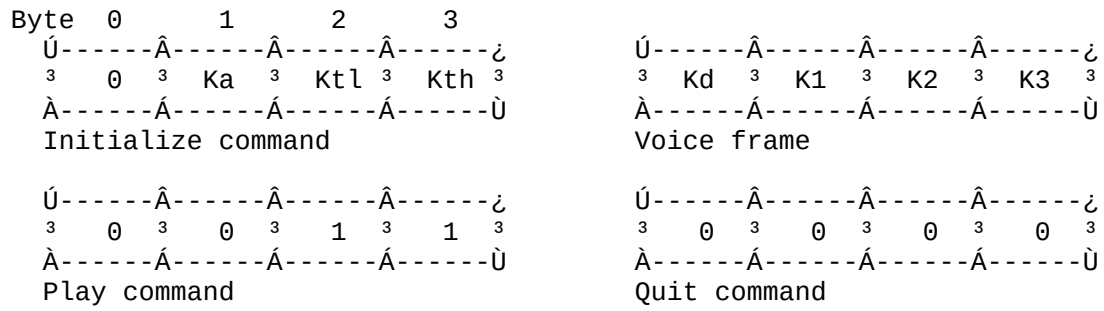

Figure 6. Tone Generator Command Formats

Initialize command format

The Initialize command consists of a byte of zero and three parameters (Figure 6). The parameters are defined as follows:

Ka	Voice amplitude (1-255)
Ktl	Tempo, low byte (0-255)
Kth	Tempo, high byte (0-255)

The overall range of the tempo Kt (Ktl and Kth) is 1-65,535 (1-FFFFh); the larger the value, the slower the overall speed of play. The amplitude and tempo affect all three voices, and stay in effect until another Initialize command is issued. If the command is issued between Voice frames to change the volume or tempo on the fly, only Voice frames following the command will be affected.

Voice Frame

Voice frames contain the duration and frequency (pitch) information for each voice. All Voice frames are stored in a 4K buffer within DoubleTalk, but are not played until the Play command is issued. If the number of Voice frames exceeds 4K bytes in length, DoubleTalk will automatically begin playing the data.

Voice frame format

Voice frames (Figure 6) consist of three frequency time constants (K1-K3) and duration byte (Kd), which specifies how long the three voices are to be played. The relationship between the time constant Kn and the output frequency fn is:

$$fn = 16{,}768/Kn$$

where fn is in Hertz and Kn = 4-255. Setting Kn to zero will silence voice n during the frame.

The Kd parameter may be programmed to any value between 1 and 255. The larger Kd is made, the longer the voices will play during the frame.

Table 1 greatly simplifies the task of finding Kn for a particular musical note. The tone generator can cover a four-octave range, from C two octaves below Middle C (Kn = 255), to D two octaves above Middle C (Kn = 14). Kn values less than 14 are not recommended.For example, the Voice frame

```
DATA 24,64,0,0
```

will play Middle C using voice 1 (K1 = 64). Since K2 and K3 are zero, voices 2 and 3 will be silent during the frame. The duration of the note is a function of both the tempo Kt and duration Kd, which in this case is 24.

As another example,

```
DATA 48,64,51,43
```

plays a C-E-G chord, for a duration twice as long as the previous example.

Note	Kn	Note	Kn
C	255 (FFh)	D	57 (39h)
C#	241 (F1h)	D#	54 (36h)
D	228 (E4h)	E	51 (33h)
D#	215 (D7h)	F	48 (30h)
E	203 (CBh)	F#	45 (2Dh)
F	192 (C0h)	G	43 (2Bh)
F#	181 (B5h)	G#	40 (28h)
G	171 (ABh)	A	38 (26h)
G#	161 (A1h)	A#	36 (24h)
A	152 (98h)	B	34 (22h)
A#	144 (90h)	C	32 (20h)
B	136 (88h)	C#	30 (1Eh)
C	128 (80h)	D	28 (1Ch)
C#	121 (79h)	D#	27 (1Bh)
D	114 (72h)	E	25 (19h)
D#	107 (6Bh)	F	24 (18h)
E	101 (65h)	F#	23 (17h)
F	96 (60h)	G	21 (15h)
F#	90 (5Ah)	G#	20 (14h)
G	85 (55h)	A	19 (13h)
G#	81 (51h)	A#	18 (12h)
A	76 (4Ch)	B	17 (11h)
A#	72 (48h)	C	16 (10h)
B	68 (44h)	C#	15 (0Fh)
C-Mid	64 (40h)	D	14 (0Eh)
C#	60 (3Ch)		

Table 1. Musical Pitch/Kn Values

Choosing note durations and tempo

Table 2 lists suggested Kd values for each of the standard musical note durations. This convention permits shorter (1/64th note) and intermediate note values to be played, while maintaining the same degree of accuracy. This is important when, for example, a thirty-second note is to be played staccato, or a note is dotted (multiplying its length by 1.5).

Using the suggested values, it turns out that most musical scores sound best when played at a tempo of 255 or faster (i.e., Kth = 0). Of course, the "right" tempo is the one that sounds the best.

Note Duration	Kd
Whole	192 (C0h)
Half	96 (60h)
Quarter	48 (30h)
Eighth	24 (18h)
Sixteenth	12 (0Ch)
Thirty-second	6 (06h)

Table 2. Duration/Kd Values

Play Command

The Play command causes the voice data in the buffer to begin playing. Additional Initialize commands and Voice frames may be sent to DoubleTalk while the tone generator is operating. DoubleTalk's SYNC flag is set during this time, enabling the host to synchronize to the playing of the tone data. SYNC returns to zero after all of the data has been played.

Quit Command

The Quit command marks the end of the tone data in the input buffer. DoubleTalk will play the contents of the buffer up to the Quit command, then return to the text-to-speech mode that was in effect when the tone generator was activated. Once the Quit command has been issued, DoubleTalk will not accept any more data until the entire buffer has been played.

Example Tune

Listing 1 is a simple Basic program which reads tone generator data from a list of DATA statements, and LPRINTs each value to DoubleTalk. The program assumes that the DTPRN printer emulator has been installed as port LPT1 (see "Printer Emulator," above).

```
100  REM  tone generator demo
110  LPRINT CHR$(1);"J";: REM  activate tone generator
120  READ B0,B1,B2,B3: REM  read a frame (4 bytes)
130  LPRINT CHR$(B0);CHR$(B1);CHR$(B2);CHR$(B3);
140  IF B0+B1+B2+B3 > 0 THEN 120: REM  loop until Quit
150  END
160 '
170 '
180  REM  data tables:
190 '
200  REM  Init (volume = 255, tempo = 86)
210  DATA  0,255,86,0
220 '
230  REM  Voice data
240  DATA  46,48,64,192, 2,0,64,192, 48,48,0,0, 48,40,0,0, 48,36,0,0
250  DATA  94,24,34,0, 2,24,0,0, 24,0,36,0, 24,0,40,0, 48,0,48,0
260  DATA  48,40,0,192, 46,36,0,0, 2,0,0,0, 48,36,0,0, 48,24,34,0
```

```
270  DATA  46,24,34,0, 2,0,34,0, 46,24,34,0, 2,24,0,0, 24,0,36,0
280  DATA  24,0,40,0, 48,0,48,0
290 '
300  REM  Play, Quit
310  DATA  0,0,1,1, 0,0,0,0
```

Listing 1. Example Tone Generator Program

The astute reader may have noticed some "non-standard" note durations (according to Table 2) in the DATA statements, such as the first two Voice frames in line 240. According to the original music, some voices were not to be played as long as the others during the beat. The F-C-F notes in the first frame are held for 46 counts, while the low F and C in the second frame are held for two additional counts. Adding the duration (first and fifth) bytes together, the low F and C do indeed add up to 48 counts (46 + 2), which is the duration of a quarter note.

SINUSOIDAL TONE GENERATOR

The musical tone generator is capable of producing three tones simultaneously, and works well in applications which require neither precise frequencies nor a "pure" (clean) output. The output is a pulse train rich in harmonic energy, which tends to sound more interesting than pure sinusoids in music applications.

DoubleTalk's sinusoidal tone generator enables the simultaneous generation of two sinusoidal waveforms. Applications for this generator range from generating simple tones to telephone call-progress tones (such as a dial tone or busy signal). The frequency range is 0 to 2746 Hz, with a resolution of 4 to 11 Hz.

The sinusoidal tone generator is activated with the command ^AnJ, where n is an ASCII number between 0 and 99. Note the similarity to the musical tone generator command, ^AJ, which uses no parameter. The parameter n programs the internal sampling rate, much like the buffered PCM mode command does; in fact, the sampling rate fs has the same relationship to n as the PCM mode command:

$$fs = 617/(155 - n)$$

Immediately following the command are three *binary* (not ASCII) parameter bytes:

^AnJ Kd K1 K2

where Kd determines the tone duration, and K1 and K2 set the output frequencies of generators 1 and 2, respectively. The tone duration and frequencies are not only functions of these parameters, but of n as well. The output amplitude is a function of the TTS synthesizer's Volume command (^AnV). The command and parameter values are buffered within DoubleTalk, and can be intermixed with text for the TTS synthesizer without restriction.

The tone duration Td is calculated as follows:

Td = Kd x 256/fs (sec)

where 0 ó Kd ó 255. Substituting the relationship fs = 617/(155 - n) into the above equation,

Td = Kd x (155 - n)/2410 (sec)

Setting Kd = 1 yields the shortest duration; Kd = 0 (treated as 256) the longest. Depending on the value of n, Td can range from 23 ms to 16.5 sec.

The tone frequencies F1 and F2 are computed as follows:

Fn = Kn x fs/1024 (Hz)

where 0 ó Kn ó 255. Substituting the relationship fs = 617/(155 - n) into this equation,

Fn = Kn x 603/(155 - n) (Hz)

Depending on the value of n, Fn can range from 0 Hz to 2746 Hz. If only one tone is to be generated, the other tone frequency may be set to 0 (Kn = 0), or equal in frequency. Note, however, that due to the additive nature of the tone generators, the output amplitude from both generators running at the same frequency will be twice that of just one generator running. Both K1 and K2 may be set to 0 to generate silence.

Note that the frequency step size and frequency range are strictly functions of n. In general, the larger n is, the larger the step size and range will be. The parameter Kn can be thought of as a multiplier, which when multiplied by the step size, yields the output frequency. For example, setting n = 95 (corresponding to an internal sampling rate of 10.28 kHz) results in a frequency step size of 603/(155 - 95) Hz, or 10 Hz. Thus, the output frequency range spans 0 Hz to 255 x 10 Hz, or 2550 Hz, in 10 Hz steps.

As an example, suppose an application needed to generate the tone pair 440/350 Hz (a dial tone) for say, 2.5 seconds. We will arbitrarily choose n = 95. The tone duration parameter Kd is calculated as follows:

Kd = 2410 x Td/(155 - n)

substituting Td = 2.5 (sec) and n = 95,

Kd = 2410 x 2.5/(155 - 95) = 100

K1 (440 Hz) is computed follows:

K1 = F1 x (155 - n)/603

= 440 x (155 - 95)/603 = 44

In like manner, K2 (350 Hz) is computed to be 35.

In order to embed the command in a text file, the computed values are converted to their ASCII equivalents (100 = 'd', 44 = ',', and 35 = '#'). The complete command becomes

```
^A95Jd,#
```

which can be embedded within normal text for the TTS synthesizer.

EXCEPTION DICTIONARIES

Exception dictionaries make it possible to alter the way the TTS synthesizer speaks any string of characters. This is useful for correcting mispronounced words, or even speaking foreign languages. This section describes how to create exception dictionaries for DoubleTalk.

The Text and Character modes of the TTS synthesizer rely on a set of ROM-based English pronunciation rules for converting text sent from the host into speech. These rules determine which sounds, or phonemes, each character will receive. The position of each letter in a word, as well as its context, is considered by each rule. DoubleTalk analyzes text in its input buffer by applying these rules to each word or character, depending on the translation mode in use. Exception dictionaries augment this process by defining exceptions (or even replacing) the ROM-based rules.

Exception dictionaries can be created and edited with word processors or editors that store documents as standard text (ASCII) files. However, the text file must be compiled into the internal format used by DoubleTalk before it can be loaded, using the COMPILE.COM program included on the Developer's Tools disk.

Exception Syntax

Exceptions have the general form

```
L(F)R=P
```

which means "the text fragment F, occurring with left context L and right context R, gets the pronunciation P." All three parts of the exception to the left of the equality sign must be satisfied before the text fragment will receive the pronunciation given by the right side of the exception. Exceptions are always terminated by a carriage return character.

The text fragment defines the characters that are to be translated by the exception, and may consist of any combination of letters, numbers, and symbols. The text fragment must always be contained within parentheses.

Characters to the left of the text fragment specify the left context (what must come before the text fragment in the input string), and characters to the right define the right context. Both contexts are optional, so an exception can contain neither, either, or both contexts. There are also 13 special symbols, or context tokens, that

can be used in an exception's context definitions. The tokens and their meanings are given in Table 3.

Symbol	Definition
#	A vowel: a, e, i, o, u, y
+	A front vowel: e, i, y
^	A consonant: b, c, d, f, g, h, j, k, l, m, n, p, q, r, s, t, v, w, x, z
*	One or more consonants
:	Zero or more consonants
?	A voiced consonant: b, d, g, j, l, m, n, r, v, w, z
@	One of: d, j, l, n, r, s, t, z, ch, sh, th
!	One of: b, c, d, f, g, p, t
%	A suffix: e(s), ed(ly), er(s), ely, ing(s), ingly, ement(s), eless, eness, able(s) (must also be followed by a non-alphabetic character)
&	A sibilant: c, g, j, s, x, z, ch, sh
$	Any nonalphabetic character (includes numbers, spaces, etc.)
\	A digit (0-9)
|	One or more digits

Table 3. Context Tokens

The right side of an exception (P) specifies the pronunciation that the text fragment is to receive, consisting of zero or more valid phonemes (listed in Table 5 of the DoubleTalk PC/LT User's Manual). If no pronunciation is given (no phonemes), no sound will be given to the text fragment, i.e., the text fragment will be silent.

A dictionary file may also contain comments, but they must be on lines by themselves (i.e., they cannot be on the same line as an exception). Comment lines begin with a semicolon character (;), so that the compiler will know to skip over it.

An example of an exception is

```
C(O)N=AA
```

which states that o after c and before n gets the pronunciation AA, the o-sound in cot. For example, the o in conference, economy, and icon would be pronounced according to this exception.

Another example is

```
$R(H)=
```

This exception states that h after initial r is silent, as in the word rhyme (the $ context token represents any non-alphabetic character, such as a space between words; see Table 3).

Punctuation, numbers, and most other characters can be redefined with exceptions as well:

```
(5)=S I NG K O            (Spanish five)
(CHR$)=K EH R IX K T ER   (Basic function)
```

Note that although these examples do not contain any context definitions, parentheses are still used around the text fragments.

The Translation Algorithm

DoubleTalk's text-to-speech algorithm works much like the human brain does when reading printed text. The algorithm scans input text from left to right and, for each character scanned, sequentially searches a list of pronunciation rules until it finds one that matches the character in the correct position and context. When a matching rule is found, the algorithm passes over the input characters bracketed in the rule (the text fragment), and assigns the pronunciation given by the right side of the rule to it. Scanning then resumes with the next character of text.

As an illustration of how the translation algorithm works, let's see how it would translate the word receive, using the ROM-based Text mode pronunciation rules.

The algorithm begins scanning at the letter r and searches the R pronunciation rules for a match. The first rule that matches is $(RE)^#=R IH, because the r in receive is an initial r and is followed by an e, a consonant (c), and a vowel (e). Consequently, the text fragment re receives the pronunciation R IH, and the scan moves past re to the next character: reCeive. (E is not the next scan character because it occurred inside the parentheses with the r; the fragment re as a whole gets the pronunciation R IH.)

The first match among the C rules is (C)+=S, because c is followed by an e, i, or y. C thus receives the pronunciation S, and processing continues with the second e: recEive.

(EI)=IY is the first rule to match the second e, so ei receives the sound IY. Scanning resumes at the character receiVe, which matches only the default V rule, (V)=V.

The final e matches the rule #:(E)$=, which applies when e is final and proceeds zero or more consonants and a vowel. Consequently, e receives no sound and scanning continues with the following word or punctuation, if any. Thus, the entire phoneme string for the word receive is R IH S IY V, which is the correct transcription.

Rule precedence

Since DoubleTalk uses its translation rules in a sequential manner, the position of each exception relative to any others must be carefully considered. For example, consider the following pair of exceptions:

```
(O)+=OW
(O)=UW
```

The first exception states that o followed by e, i, or y is to be pronounced OW, the o-sound in boat. The second exception does not place any restriction on what must come before or after o, so o in any context will match and receive the UW pronunciation. Note that if the exceptions were reversed, the (O)+ exception would never be reached since the (O) exception will always match o in any context. In general, tightly-defined exceptions (those containing many context restrictions) should precede loosely-defined exceptions (those with little or no context definitions).

```
(RAT)=R AE T
(RATING)=R EY T IH NG
(R)=R
```

This is an example of how *not* to organize exceptions. The exception (RATING) will never be used because (RAT) will always match first. According to these exceptions, the word rating would be pronounced "rat-ing."

It can be beneficial to also group exceptions by the first character of the text fragments, that is, all of the A exceptions in one group, all the B exceptions in a second group, and so on. This gives an overall cleaner appearance, and can prove to be helpful if the need arises to troubleshoot any problems in your dictionary.

Text not matched by the dictionary

It is possible that some input text will not match any of the exceptions, depending on the nature of the exception dictionary. For example, if a dictionary was written to handle unusual words, only those words would be defined by the exceptions. In this case, no other words would be handled by the exceptions. On the other hand, if the dictionary defined the pronunciation for another language, it would probably be comprehensive enough to handle most all types of input. In any case, if an exception is not found for a particular character, the English pronunciation will be given to that character according to the built in (ROM) English pronunciation rules.

Generally, the automatic switchover to the ROM rules is desirable if the dictionary is used to correct mispronounced words, since in essence it is defining exceptions to the ROM rules. If the automatic switchover is not desired, however, there are two ways to prevent it from occurring. One way is to end each group of exceptions with an unconditional exception that matches any context. To ensure the letter "a" will always be matched, for example, end the A exception group with the exception (A)=pronunciation. This technique works well to ensure matches only for specific characters, such as certain letters or numbers.

If the exception dictionary is to replace the ROM rules entirely, end the dictionary with the following exception:

```
()=
```

This special exception causes unmatched characters to simply be ignored, rather than receive the pronunciation defined by the ROM rules.

Effect on punctuation

Punctuation defined in the exception dictionary has priority over the Punctuation Filter command. Any punctuation defined in the dictionary will be used, regardless of the Punctuation Filter setting. However, if the dollar sign ($) character is defined in the text fragment of any exception, currency strings will not be read as dollars and cents.

Character mode exceptions

The information presented thus far applies to both the Text and Character modes of DoubleTalk. Character mode exceptions, however, can be defined independently of the Text mode exceptions, or be included as a subset of them to avoid duplicating similar exceptions.

The beginning of the Character mode exceptions is defined by including the character C between the last Text exception and the first Character exception. No exceptions prior to this marker will be used when DoubleTalk is in Character mode. For example:

```
.    (Text mode exceptions)
.
()=  (optional; if Character mode exceptions are
     not to be used in Text mode)

C    (Character mode exceptions marker)
.
.    (Character mode exceptions)
.

()=  (optional; used if ROM rules are not to be
     used in no-match situations)
```

Applications

The following examples were chosen to give you some ideas of how the exception dictionary can be used.

Correcting mispronounced words

The most obvious of all applications; mispronounced words can be corrected by writing exceptions for them.

```
(SEARCH)=S ER CH
$(OK)$=OW K EY
```

The first exception corrects the pronunciation of all words containing search (search, searched, research, etc.). As this exception exemplifies, it is only necessary to define the problem word in its root form, and only the part of the word that is mispronounced (ear in this case). The second exception corrects the word ok, but because of the left and right contexts, will not cause other words (joke, look, etc.) to be incorrectly translated.

No cussing, please!

The speaking of specific characters or words can be suppressed by writing "null" exceptions for them, that is, exceptions in which no pronunciation is given.

```
(????)= (YOU fill in the blanks!)
```

When 0 is not zero

When we read addresses or lists of numbers, we often substitute the word "oh" for the digit 0. For example, we might say 1020 North Eastlake as "one oh two oh North Eastlake." The digit 0 can be redefined in this manner with the following exception:

```
(0)=OW
```

Arithmetic operators

Some characters may have more than one name; for example, / may be read as "slash" or "divided by," depending on the context. Such characters can be redefined if their default names don't fit the application. For example, the arithmetic operators (/, *, ^, etc.) can be defined for mathematical applications with the following exceptions:

```
(/)=D IH V AY D IH D   B AY
(*)=M AH L T IH P L AY D   B AY
(^)=R EY Z D   T UW
 .
 .
etc.
```

Acronyms and abbreviations

Acronyms and abbreviations can be defined so the words they represent will be spoken:

```
$(KW)$=K IH L OW W AA T
$(DR)$=D AA K T ER
$(TV)$=T EH L AX V IH ZH IX N
```

Heteronyms

Heteronyms are words that have similar spellings but are pronounced differently, depending on the context, such as read ("reed" and "red") and wind ("the wind blew" and "wind the clock"). Exceptions can be used to fix up these ambiguities, by including non-printing (Control) characters in the text fragment of the exception.

Suppose a line of text required the word "close" to be pronounced as it is in "a close call," instead of as in "close the window." The following exception changes the way the s will sound:

```
(^DCLOSE)=K L OW S
```

Note the Control-D character (^D) in the text fragment. Although a non-printing character, the translation algorithms treat it as they would any printing character. Thus, the string "^D close" will be pronounced with the s receiving the "s" sound, wherever it appears in the text stream. Plain "close" (without the Control-D) will be unaffected-the s will still receive the "z" sound. It does not matter where you place the Control character in the word, as long as you use it the same way in your application's text. You may use any non-printing character (except LF and CR) in this manner.

Foreign languages

Dictionaries can be created that enable DoubleTalk to speak in foreign languages. It's not as difficult as it may first seem-all that is required is a high school level foreign language textbook and a bit of patience. If you don't have a book for the language you're interested in, check your local library. Most libraries have foreign language dictionaries that include pronunciation guides, which make it easy to transcribe the pronunciation rules into exception form. The Spanish exception file, SPANISH.EXC, was written using this method.

Language translation

Exception dictionaries even allow DoubleTalk to read foreign language text in English! The following exceptions demonstrate how this can be done with three Spanish/English words.

```
(GRANDE)=L AA R J
(BIEN)=F AY N
(USTED)=YY UW
```

The sense of translation can also be reversed:

```
(LARGE)=G R A N D EI
(FINE)=B I EI N
(YOU)=U S T EI DH
```

Tips

Make sure that your exceptions aren't so broad in nature that they do more harm than good. Exceptions intended to fix broad classes of words, such as word endings, are particularly notorious for ruining otherwise correctly pronounced words.

Take care in how your exceptions are organized. Remember, an exception's position relative to others is just as important as the content of the exception itself.

Exception anomalies

On rare occasions, an exception may not work as expected. This occurs when the ROM-based pronunciation rules get control before the exception does. The following example illustrates how this can happen.

Suppose an exception redefines the o in the word "process" to have the long "oh" sound, the way it is pronounced in many parts of Canada. Since the word is otherwise pronounced correctly, the exception redefines only the "o:"

```
PR(O)CESS=OW
```

But much to our horror, DoubleTalk simply refuses to take on the new Canadian accent.

It so happens DoubleTalk has a rule in its ROM which looks something like this:

```
$(PRO)=P R AA
```

This rule translates a group of three characters, instead of only one (as most of the ROM rules do). Because the text fragment PRO is translated as a group, the o is processed along with the initial "pr," and consequently the exception never gets a shot at the o.

If you suspect that this may be happening with one of your exceptions, include more of the left-hand side of the word in the text fragment (in the example above, (PRO)=P R OW would work).

EXCEPTION COMPILER ----------

The COMPILE.COM program is used to compile exception dictionaries for DoubleTalk's TTS synthesizer. Since the input (source) to the compiler is, and must be, a plain ASCII text file, exception dictionaries can be written and edited with any word processor or text editor, such as DOS 5.0's EDIT program.

You can compile exception dictionaries in either of the following ways:

1. Type all the information on the command line, using the following syntax:

   COMPILE sourcefile [objectfile] [options]

2. Or type:

   COMPILE

and respond to the following prompts:

```
Source file [.EXC]:
Object file [source.EXS]:
```

The source file is the ASCII text file containing your exception dictionary. The object file is the name of the compiled exception file that will be created. If you do not supply a name for the object file, the compiler will use the source file name, and add the extension .EXS automatically.

Options for the COMPILE command consist of either a forward-slash (/) or dash (-) character followed by a letter. Options can be specified only in the single command line syntax, and if given, must appear after the file name(s).

File Types

The compiler supports three different output (object) formats, described in this section. The default object file name extensions are shown in the following table:

Option Switch	Default Extension	File Type
none	.EXS	Standard
/A	.EXA	Auto-load
/C	.EXH	Chained

where File Type is defined as:

.EXS Standard object file. Contains the compiled exceptions, terminated by the EOF character (1Ch).

.EXA Auto-load object file. The structure of this file type is as follows:

```
0Dh 1Eh [ 01h "18G" 00h ] 01h "L" 00h
          .
          .
(compiled exceptions)
          .
          .
1Ch 00h 01h "U" 00h
```

The header contains all the commands necessary to begin downloading the exceptions to the synthesizer.

The trailing bytes immediately following the exception block halt the downloading process and enable the exceptions that were just loaded. This format makes it possible for a program to load exception files by simply sending the file to the synthesizer - all the details of invoking the Load mode and enabling the exceptions are taken care of automatically. For example, the DOS command

```
TYPE SPANISH.EXA > LPT3
```

will download and activate the Spanish exceptions (note that the DTPRN driver must have been installed as LPT3 for this example to work).

The optional characters in the brackets ([]) enable the HiAscEn bit, and are included only if the compiler detected extended ASCII characters in the source file. See "Protocol Options Command" for more information.

.EXH Chained object file. Same as the Standard file type, but without the EOF character at the end. This format is used only when the file is to be followed by another block of exceptions (such as a user-defined dictionary). It is the responsibility of the application program to ensure that the EOF character is sent to the synthesizer after the last block of exceptions has been loaded.

Input (source) files normally carry the .EXC extension. If you omit the extension for the source or object file name, the compiler will add the default extension(s) (.EXC and/or .EXS) for you automatically.

The following COMPILE commands are equivalent:

```
compile oddnames.exc,oddnames.EXS

compile oddnames oddnames

compile oddnames
Object file [ODDNAMES.EXS]: <Enter>

compile
Source file [.EXC]: oddnames
Object file [ODDNAMES.EXS]: <Enter>
```

as are these:

```
compile acronyms.exc /a
Object file [ACRONYMS.EXA]: test.EXA

compile acronyms,test /a
```

Compiling From the Command Line

The compiler also supports an immediate mode of operation, which compiles a single source exception on the command line. This is useful in situations where only a few exceptions are needed, and/or the compiled output is to be hard-coded into an application program.

```
COMPILE exception
```

Example:

```
compile s(a)mple=ae
```

Compiler output:

```
S(A)MPLEey;
```

Downloading Compiled Dictionaries

Downloading dictionaries which have been compiled is easy. If the dictionary was compiled with the /A switch, your application only needs to send the object file to the synthesizer as it would normal text. The LOADER.BAS program is an example of how to download a standard (.EXS) object file.

PCM MODE

In PCM mode, all data sent to DoubleTalk is written directly to DoubleTalk's digital-to-analog converter (DAC). This results in a very high data rate, but provides the capability of producing the highest quality speech, as well as sound effects that are not possible using the other modes. DoubleTalk also supports ADPCM, which reduces the effective data rate by a factor of one-half to one-third that of standard PCM. PCM driver source code, compatible with both the PC and LT, can be found in the file DT_PCM.ASM.

DoubleTalk PC supports two PCM modes: buffered and non-buffered. In buffered mode, PCM data is queued in an internal 4096 byte circular buffer, which is dialed out to the DAC at a programmed sampling rate. DoubleTalk PC's TTS port status flags are constantly updated in this mode, and I/O transfer is performed using the usual handshaking (RDY) method. The maximum sampling rate supported in buffered mode is 11 kHz. Both DoubleTalk PC and LT support this mode.

In non-buffered mode, PCM data is sent directly to the DAC in real time. The TTS port status flags are meaningless in this mode; the maximum sampling rate is better than 48 kHz. Only DoubleTalk PC supports non-buffered mode.

Buffered PCM Mode

The command ^An# places DoubleTalk in the Buffered PCM mode. All subsequent data sent to DoubleTalk is then routed to the DAC buffer.

Because the PCM data is buffered within DoubleTalk, the output sampling rate is independent of the data rate into DoubleTalk, as long as the input data rate is equal to or greater than the sampling rate.

The sampling rate can be programmed to virtually any rate between 4 and 11 kHz with the PCM Mode command. The relationship between the command parameter n and the sampling rate fs is

```
 n = 155 - 617/fs
fs = 617/(155 - n)
```

where fs is measured in kHz. The range of n is 0-99, hence fs can range from 4 to 11 kHz.

The procedure for sending PCM data to DoubleTalk is straightforward:

1. Program the desired volume level with the Volume (^AnV) command. A volume setting of 5 will cause the PCM data to be played back at its original volume level. This step is optional.

2. Issue the Buffered PCM Mode command, ^An#. The value of n will set the sampling rate within DoubleTalk.

3. Immediately begin transferring the PCM data to DoubleTalk. The same methods employed for sending ASCII data to the text-to-speech synthesizer should be used (see "Hardware I/O" section, above). PCM data must be sent to DoubleTalk as linear, eight bit signed data (-127 to +127, 0 = midscale). Note that if the AE status flag is 1, a block of 4096 - 300 = 3796 bytes may be transferred to DoubleTalk without concern of overflowing the DAC buffer. This can be beneficial in applications that need to do background tasks concurrently with PCM-based sounds.

4. After the last byte of PCM data has been sent to DoubleTalk, send the value 80h (-128d). This signals DoubleTalk to terminate PCM mode and return to the text-to-speech mode of operation. Note that up to 4096 bytes of PCM data may still remain in the DAC buffer, so DoubleTalk may continue producing sound for as long as 1 second (4 kHz sampling rate) after the last byte of data has been sent. DoubleTalk's SYNC flag will not be cleared until all of the speech data has been output to the DAC, at which time DoubleTalk will again accept data for the text-to-speech synthesizer.

Non-Buffered PCM Mode

The command ^A# places DoubleTalk PC in non-buffered PCM mode. Note that this is similar to the Buffered PCM Mode command, but the sampling rate parameter n is omitted. As in buffered mode, PCM data is expected to be linear, eight bit signed data, and the output level can be adjusted with the Volume command. It is up to the application program to write the PCM data to DoubleTalk at the desired sampling rate (e.g., for a desired sampling rate of 44.1 kHz, each sample should be spaced 22.7 uS apart). PCM mode is

terminated immediately when a value of 80h (-128d) is written to the TTS port.

TTS SYNTHESIZER PROGRAMMING TIPS

This section contains additional information about DoubleTalk's TTS synthesizer which is not covered in the User's Manual. You will learn how to ensure DoubleTalk remains as responsive as possible, a few programming tricks, and how to support user exception dictionaries.

Response Time Considerations

Considering the computations DoubleTalk has to go through to translate even the simplest word, it can still be considered to be quite responsive - an important requirement for visually impaired users of screen-reader programs. Understanding how DoubleTalk's text-to-speech algorithms work can help ensure that your programs will be as responsive with DoubleTalk as possible.

When the TTS synthesizer receives text from an application program, it queues up the characters in an internal 2.5K buffer (less if an exception dictionary is loaded). Assuming the synthesizer is not already talking, the TTS algorithms wait until a carriage return or null is received (defining what we shall call a "synthesizer phrase boundary") before translation into speech begins. (This is not true of Character mode - translation begins immediately upon receipt of the first character.)

Speaking from a "cold start" is the worst-case scenario for the TTS synthesizer, because it must translate a relatively large chunk of text before it can begin speaking. Once it has begun talking, however, it will continue to accept and store more phrases and preprocess them "ahead" of the speech, so the next phrase is already translated once the speech has caught up to it. It is the initial-phrase latency time that concerns blind users, because they like the speech to begin as soon as possible upon giving the command.

Normally-written text generally contains one or more punctuation marks, such as commas and/or periods. In terms of human speech, these marks indicate where a natural pause is to be placed. DoubleTalk takes advantage of this fact by translating text in its buffer in punctuation-delimited segments, as defined by the punctuation boundaries (which may or may not occur at synthesizer phrase boundaries). For this reason, it is important that all punctuation be sent with the text. The latency time of an initial phrase with the punctuation left in it will be much less than that of the same phrase with the punctuation stripped out of it.

Most screen-reader programs do not send the actual punctuation to the synthesizer, but send the spelling instead ("period," "comma," etc.). If your program falls into this category, try adding the actual punctuation to the end of the word ("period." "comma,"). This will make the speech more responsive, as well as give the added

benefit of more natural sounding intonation as the text is read by the synthesizer (punctuation directly affects the inflection contours).

If your program waits until the current phrase has been completely spoken before sending the next phrase, try using the SYNC2 status flag instead of SYNC (see "Hardware I/O, TTS Port," above). This will help increase the responsiveness by as much as 50%, depending on the speed and other factors.

Creating Pauses in Speech

After all this talk about avoiding pauses, why would one want to *create* them? One example is for creating the natural pauses a human reader places between paragraphs of a passage. Another would be for creating emphasis at a particular point in a sentence, to help enhance its meaning.

There are at least two ways to add pauses in your text. The easiest way, for short passages, is to simply string together commas and/or periods where each pause is to take place. Commas create a medium pause; periods create a long pause. Commas and periods may be combined to create even longer pauses. If you don't want the downward inflection associated with a period to occur, insert at least one space between it and the word preceding it:

```
Now let me think...       (inflects downward)
Now let me think ...      (no inflection)
```

The second method of creating pauses in the speech makes use of the synthesizer's exception dictionary. Say, for example, that you wanted a long pause to be generated between indented paragraphs of text in a text reader program. It is not practical to insert additional punctuation marks in the text the program will be reading, since the text content is generally not known beforehand. But let's say we loaded the following exception into the dictionary:

```
(   )=..      (3 spaces)
```

This exception causes the speech to pause (actually two long pauses comprised of the two "." phonemes) whenever three or more consecutive spaces appear in the text, namely the indent of each paragraph. Note also that this method will not affect the inflection in any way, as did the first method. An additional benefit is that this method will work for virtually any text input.

Forcing Character Pronunciation

It is not necessary to change from Text mode to Character mode to have text spelled out instead of spoken as words. By following each letter with a phrase delimiter (carriage return or null character), the text will be spelled letter by letter.

A potential problem with this method, however, is with the letter "A" - unless it is followed by another vowel, it will be pronounced

as "ah" instead of "ay." (This is due to the internal Text mode pronunciation rules.) To avoid this problem, use the spelling "AY." Of course, Character mode will avoid this problem altogether, and is more responsive than Text mode.

Supporting User Dictionaries

If you want to support user-defined exception dictionaries in your program, consider using the following strategy:

During your program's initialization phase, look in the current disk directory for a specific, documented dictionary file name, such as USER.EXH. If it exists, load it into the synthesizer, but only *after* DoubleTalk has been initialized. (All RC Systems drivers initialize DoubleTalk when they start up, including wiping out any exception dictionary; this includes DTPRN, INT4DAPI, DTC's Dt_Init and DTQB's DTINIT functions.) This enables your customers to use their dictionaries conveniently in the context of your program.

If you are going to use your own dictionary as well, load it first, before loading the user dictionary. Be sure to use the "chained" file type, so the user dictionary will load properly. Refer to the section "Exception Compiler" above for more information about the dictionary file formats.

To summarize, these are the steps your program should follow:

1. Load and initialize any drivers your program may use for interfacing with DoubleTalk.

2. Load and output the application program's exception dictionary, if used. It should be of the .EXH (chained) file format.

3. Load and output the user dictionary, if it exists. It should also be of the .EXH file format.

4. Output the EOF character (1Ch) to the synthesizer to signal the last of the exceptions have been loaded.

Of course, if only one dictionary is to be used (either the application's or user-defined, but not both), the .EXA or .EXS file formats may be used to simplify the loading process.

APPENDIX A
LPC Speech Encoding Services

The following companies are known to offer LPC speech development services. Their appearance here is not an endorsement by RC Systems.

Creative Education Institute
David Rousch
P.O. Box 7306
Waco, TX 76714
(817) 751-1188

Laureate Learning Systems
Burnie Fox
110 E. Spring St.
Winooski, VT 05404
(802) 655-4755

Texas Instruments
Regional Technology Center
17891 Cartwright Dr.
Irvine, CA 92714
(714) 660-8292

APPENDIX B
Additional Information

Application Note #1 - "Interfacing to Other Devices"

Describes methods of interfacing DoubleTalk PC to devices such as radio transmitters and the telephone network. Available from RC Systems Technical Support Services.