Professional™ 300 series

Telephone Management System (TMS) Programmer's Manual

AA-AD34A-TH

Developer's Tool Kit

digital

software

Telephone Management System (TMS) Programmer's Manual

AA-AD34A-TH

November 1983

This document describes TMS, the Professional Telephone Management System, as implemented for the Professional Developer's Tool Kit. This is a user guide and reference manual for programmers developing telephone applications for Professional personal computers.

DEVELOPMENT SOFTWARE:	Professional Host Tool Kit V1.7 PRO/Tool Kit V1.7
TARGET OPERATING SYSTEM:	P/OS V1.7

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754

First Printing, November 1983

CONTENTS

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES

This manual describes TMS and is intended to be used as both a reference manual and user guide. It covers applications running on the Professional and also provides information about the hardware components of TMS.

INTENDED AUDIENCE

You should read this manual if you are developing a telephone communications application for the Professional 300 Series computer and need information about TMS, the Telephone Management System that runs on the Professional computer. TMS is one of the tools that can be used in developing communications applications for the Professional computer.

This document is intended for programmers who have had experience with systems programming and communications applications software. The reader also is expected to be familiar with the Professional Developer's Tool Kit, and either MACRO-11 or any of the other Tool Kit Languages.

It is recommended that you also read the _Tool Kit User's Guide_ and the _P/OS System Reference Manual_ for a more information on communications software development on the Professional computer.

STRUCTURE OF THIS DOCUMENT

The manual has six chapters and three appendixes. The contents are summarized in the following subsections.

Chapter 1 -- Introduction

Introduces TMS capabilities and the environment for telephone communications applications running on the Professional. Presents TMS hardware and software components, along with both standard and optional TMS features. Compares the Professional's Communications Services Library (COMLIB) capabilities and ease-of-programming to TMS' own I/O driver (XT) software efficiency. Describes typical TMS application functions and capabilities.

Chapter 2 -- System Description

Describes the major TMS system components and their interface with P/OS, the Professional's operating system. Describes the functions and capabilities of each component and the various programmable telephone capabilities inherent in TMS.

Chapter 3 -- Driver Calls to TMS

Summarizes TMS driver call groups and their functions. Describes the Queue Input/Output (QIO) function directives that send instructions to TMS and return TMS status and event reports. Provides MACRO-11 programming syntax rules and examples, as an introduction to the remainder of the chapter, an alphabetically ordered detailed description of the TMS calls.

Chapter 4 -- Communications Characteristics

Describes the TMS' four data modes, DTMF, serial, voice, or Codec. Provides detailed descriptions of the various programmable characteristics for TMS data lines.

Chapter 5 -- TMS Event Reporting

Describes TMS' reported reactions to changes in the state of the telephone lines and other hardware components. Defines events that TMS responds to and lists the driver calls used to program event handling.

Chapter 6 -- TMS Hardware Components

Provides a functional description of the individual TMS hardware components.

Appendix A -- Instruction Summary

Lists the TMS calls and their associated parameters.

Appendix B -- Sample Program

Pascal/MACRO-11 sample routines

Appendix C -- TMS error and status codes and their meanings

ASSOCIATED DOCUMENTS

- Telephone Management System Owner's Manual
- Installation Instructions Telephone Management System

Also, refer to the other Tool Kit Documentation Set manuals for more information on developing communications applications for the Professional.

CONVENTIONS USED IN THIS DOCUMENT

This document uses the following conventions.

Convention	Meaning
[optional]	Square brackets indicate optional fields in a call line. Do not include the brackets in the call line. If the field appears in lowercase type, you must substitute a legal parameter if you include the field. Do not confuse the square brackets in a call line format with the square brackets that you use to specify a file specification.
UPPERCASE	Any call line field in uppercase type indicates that you should type the word or letter exactly as shown.
lowercase	You must substitute a value for any call line field in lowercase type. Usually the lowercase word identifies the type of substitution required.
...	A horizontal ellipsis indicates that you can repeat the preceding item one or more times. For example: parameter [,parameter...]
. . .	A vertical ellipsis in a figure or example means that not all of the statements are shown.

CHAPTER 1

INTRODUCTION

The Professional Telephone Management System (TMS) integrates everyday telephone equipment with Digital's Professional computers. TMS offers voice communications management, automatic telephone dialing and answering, multi-speed modem capabilities, and message recording and playback.

TMS lends itself to a wide variety of software applications. The TMS hardware components are programmable under control of the TMS I/O Driver (XTDRV) software interface. The following list provides just a few of the potential application areas for TMS:

- Telemarketing (order entry, mailing lists, and the like)
- Automated store and forward
- Voice mail with forwarding
- Combined text and voice message handling
- Teleconferencing with simultaneous voice and text
- "Smart Directory" with automatic dialing
- Integrated executive workstation

Applications extend the capacities of most professional business areas, such as integrated dictation and word processing, telemarketing, teleconferencing, auto-dialed communications, automatic computer-to-computer data exchange, and voice message storage and retrieval. TMS' hardware and the Professional's XT driver interface software are designed to be the optimum tools for telecommunications application development.

TMS operates on two public switched telephone network or on private branch exchange lines. One of the telephone lines accommodates a telephone handset that can be used for normal telephone operation (non-computer-assisted). The handset must be provided by the user or by the local telephone supplier.

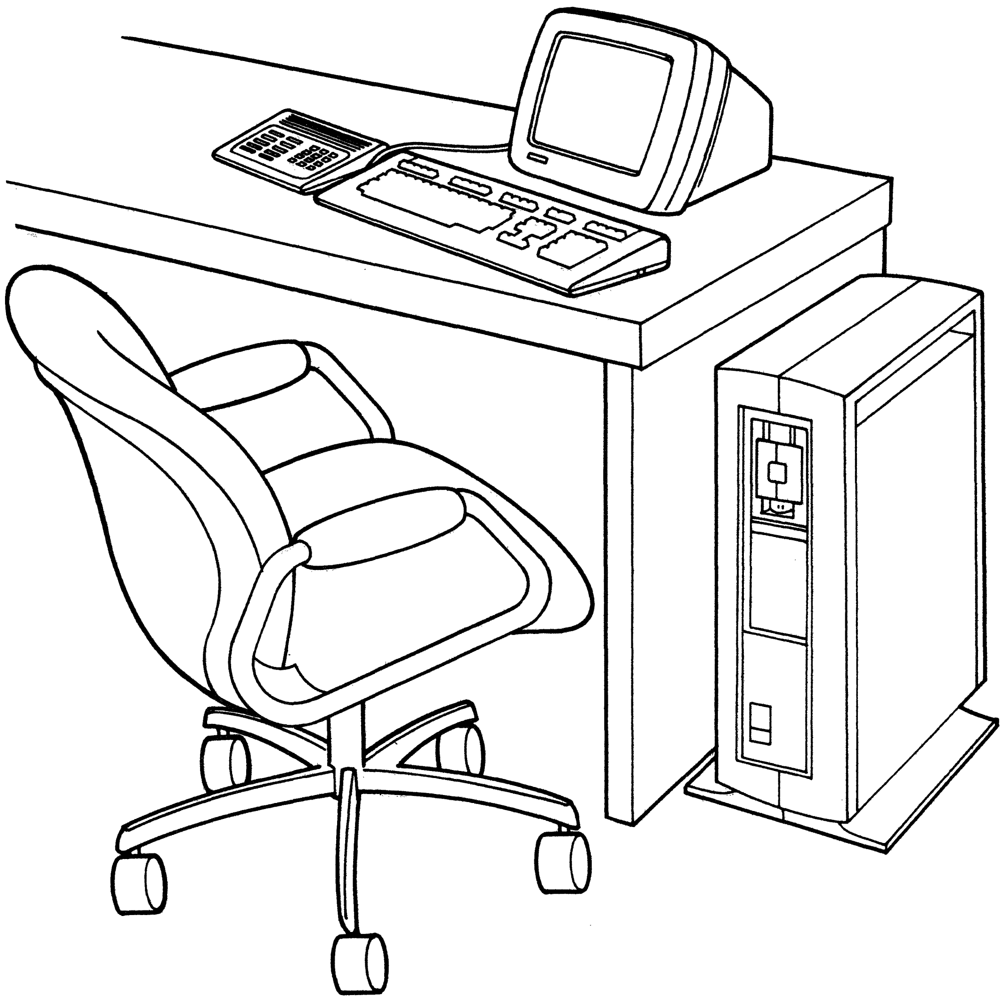

Figure 1-1: Professional Telephone Management System (TMS)

TMS also has an optional "no-hands" voice unit with keypad, speaker, and microphone. Figure 1-1 shows a Professional equipped with TMS and an optional voice unit.

TMS application programs running on the Professional can originate or answer calls. Applications can perform these operations semi-automatically, under explicit control of a Professional keyboard user. Alternatively, applications can perform fully automatically on a time-delayed basis and/or to transmit data in an unattended mode.

Design considerations ensure that TMS operations do not use Public Switched Telephone Network resources unnecessarily for ineffective calls. Various TMS operations are illustrated in the series of sample application descriptions provided at the end of this introductory chapter.

1.1 PROFESSIONAL SERIES COMMUNICATIONS SERVICES

Communications services allow you to perform communications operations on the Professional. There are three categories of communications services in the Professional environment, including TMS:

1. Base System Services included with the P/OS operating system. These services include an asynchronous driver (XKDRV) for the communications port, as well as a communications service library (COMLIB).

2. PRO/Communications Services, optional application software providing features beyond those of the base system. These services include utility programs (such as a terminal emulator), as well as additional communications routines.

3. Telephone Management System (TMS) Services, the optional package that includes TMS communications hardware and driver software for TMS' unique communications capabilities.

Your TMS application program can access the Professional communications services by calling routines you program with the functions described in Chapter 3. Alternatively, high-level programming languages can choose to call TMS using existing Professional communications library routines or to spawn tasks that execute Professional communications utility programs.

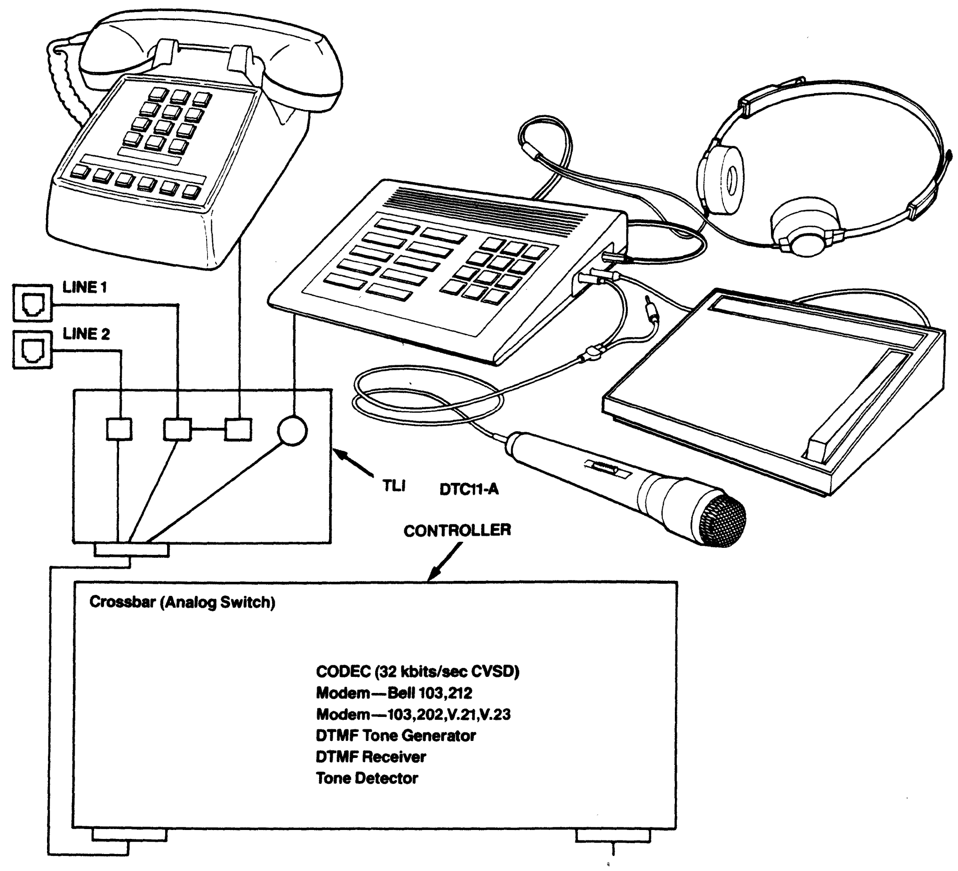

Figure 1-2: TMS Component Diagram

For example, your program can call a routine named CCATT to attach a telephone line; alternatively, the program can spawn a task that executes a terminal emulation utility. For applications written in MACRO-11 or with queue input/output (QIO) requests, a call to the TMS IO.ATT function performs the same line attach.

TMS communications functions are described in Chapter 3 of this manual. For details of base system P/OS communications services and the Professional's Communications Services Library (COMLIB), refer to the Tool Kit User's Guide. In this manual, Appendix C provides some sample MACRO-11 routines and a Pascal program that calls the various routines. These samples are provided as examples of one method for programming TMS for telephone communications applications.

1.2 TMS COMPONENTS

The Professional's TMS option is made up of three hardware components: (1) an internal controller board that plugs into the Professional's bus, (2) a telephone line interface unit (TLI) that provides standard plug jacks for connecting two different telephone lines, and (3) an external telephone set, and an optional cable-connected voice unit. TMS' two telephone lines operate independently, rather than connecting them together as a single unit involving two lines. Chapter 2 describes component functions.

The TMS internal controller board has on-board memory, its own 8031 micro-processor, serial data communications (modem) components, and ROM-based firmware under control of the TMS I/O Driver. Event reporting from TMS to the Professional can be enabled so that unsolicited and unscheduled events involving the TMS hardware trigger interrupts to P/OS via the Professional's data bus. Figure 1-2 is a diagram of TMS components. For specifics on TMS hardware components, refer to Chapter 2.

1.3 STANDARD TMS FEATURES

TMS capabilities offer the following calling methods and data modes:

- Voice calls involving a user present at the system.
- Data calls (originated/answered both semi-automatically and automatically) using the current asychronous modem capable of Bell 103 and Bell 212 operations or using tone data. (A version of TMS is planned with V.21, V.23 capabilities for international communications. Also Bell 202 capabilities are provided by the TMS hardware, but without software support.)

- Codec calls (digitized voice calls, which can be originated or answered both semi-automatically and automatically).

- Tone-data calls using dual-tone multifrequency (DTMF), or "Touch-Tone"* sounds. DTMF tones can be used to request that an application retrieve digitized voice messages, store new messages, or perform other data processing functions.

*Touch-Tone is a registered trademark of AT&T

Under software control, each mode of TMS call can be switched from current mode to any of the other modes. For example, a user who places a voice mode call to ask for computer services could switch to data mode for the transfer of data files. A DTMF mode caller could signal with a DTMF tone to switch TMS to codec mode and rerieve recorded messages. A codec call could switch to voice mode to allow person-to-person order taking following a recorded marketing message, and so forth.

1.3.1 Dialing Capabilities

TMS is capable of dialing calls using either the interrupted pulsing method of dialing (the TMS I/O driver default dial mode) or the CCITT standard MF 4 tones. The user specifies with a setup option the dialing mode for each line. This setup is executed automatically each time the Professional operating system is initialized and remains in effect until altered by the user or until the next initialization of the system.

For dialing using application software, the application program can specify a telephone number with a maximum length of 48 characters, including the special characters. (The TMS I/O driver provides functions for dialing TMS calls from the application program. Refer to the IO.CON description in Chapter 3 of this manual.)

When dialing via the telephone set or the keypad on the Voice Unit, there are no special considerations; the user dials in the normal manner and listens for intermediate tones as if using a regular telephone.

1.3.2 Voice Storage and Retrieval

The TMS voice digitizer translates analog voice signals into digital signals suitable for computer storage. The TMS storage requirement uses approximately 0.25 megabytes of storage per minute of spoken voice. Voice to be stored can be entered from a remote telephone dialed into TMS or can be entered using the optional TMS voice unit. Speech can be retrieved both remotely or with the voice unit.

An application program could forward voice messages from one Professional to another using a high-speed communications link to transmit the encoded form in serial data mode (a lengthy process unless a communications link is extremely fast, such as Ethernet). Alternatively, the application program could establish a telephone link in Codec mode to rebroadcast the message to a waiting receiver.

1.4 OPTIONAL FEATURES

The TMS voice unit makes a wide variety of telecommunications applications possible. It allows you to answer and originate calls using the buttons on the voice unit keypad. The voice unit serves as a conference-call phone and device for dialing and signalling applications.

The voice unit option also provides connections for a headphone, a hand-held microphone, or a standard dictation foot pedal. When making telephone calls, the voice unit normally operates on line 1; however, an application program could intercept the user's actions on the voice unit and direct the call to line 2 or could perform other actions, such as storing voice or retrieving pre-stored voice recordings without telephone line involvement.

A secondary firmware component resides in the optional TMS voice unit to control voice unit hardware and to communicate with firmware on the TMS controller unit. For details on TMS hardware, refer to Chapter 2 and Chapter 6.

1.5 OPERATING EFFICIENCY

TMS firmware has several features to ensure that it is operating on an effective call. These features prevent TMS from needlessly using the resources of the public telephone network.

When operating in voice mode, TMS requires that a user be present at the system to monitor the progress of the call. Unless a physical, external stimulus is received (lifting the telephone handset or depressing a key on the Voice Unit) any call originated or answered in voice mode is abandoned automatically after 40 seconds.

When using modems, the serial data mode requires that modem carrier be present. If, during the initiation of a call, carrier is not received within 40 seconds of completion of dialing or answering, the call is abandoned. The loss of carrier during a call also causes a call to be abandoned.

When using DTMF control signalling, the absence of any new DTMF signals from the distant end for two minutes and fifty seconds initiates a ten-second warning tone after which the call is abandoned.

Codec mode can be used to deliver a sound message or to record a message for a maximum of two minutes and fifty seconds (unless DTMF control signals are being received). If no DTMF signals are received within the timeout period, a ten-second warning tone is heard. Unless a DTMF tone is detected during the ten seconds, TMS abandons the call.

It is expected that this mode be used primarily for answering incoming calls and providing special applications. It is also possible to originate calls in this mode. It is recommended that applications originating calls begin a voice announcement immediately after dialing so that the recipient of the call is aware of what is happening.

1.6 EMERGENCY CONSIDERATIONS

If a telephone line in use by TMS is needed for an emergency, TMS has three methods of releasing the line quickly. If you lift the telephone handset on line 1, TMS automatically relinquishes control of the line.

EMERGENCY CONSIDERATIONS

TMS always relinquishes control of the line and returns the telephone handset to normal use whenever the power to the Professional computer is turned off. As an additional emergency consideration, a lighted switch on the rear of the Professional can be used to disable the use of TMS instantly.

NOTE

> When the switch is out and the light is off, TMS is disabled and does not have access to the telephone lines.

1.7 TMS OPTIONS

Table 1-1 lists the TMS components and order numbers.

Table 1-1: TMS Components and Order Numbers

Component	Order Number
TMS Standard Equipment	DTC11-A
1 TMS Controller Board,	
1 Telephone Line Interface,	
1 Modular jack phone cable	
TMS Optional Equipment	
Multi-line keyset adapter	DTC11-XK
Voice Unit	DTC11-B
Voice Unit Foot Pedal	DTC11-XF
Voice Unit Microphone	DTC11-XM
Voice Unit Headset	DTC11-XH

1.8 APPLICATION EXAMPLES

The following sections provide sample application situations and the way TMS functions in handling the various situations. Note that the TMS-specific functions are used for these samples; optionally, other Professional communications services can be used in place of the TMS functions. Refer to the Tool Kit User's Guide for information on the Professional's communications services.

1.8.1 Voice Calls for Telemarketing

Incoming voice calls:

When TMS is used for telemarketing applications that involve incoming calls (information service or order-taking situation), the operation takes place as follows:

1. You start up a telemarketing application program.

2. The application program chooses either to enable auto-answering or to issue explicit answer commands on detection of incoming ring(s).

3. A call can be answered initially in either voice or Codec mode. If the initial answer is in Codec mode, a special "greeting" announcement could be made.

4. The call is passed to you. TMS notifies you of the call with a "zip tone," heard by both parties.

5. The ON/OFF LED on the Voice Unit (if present) flashes (240 impulses per minute) during the first 40 seconds of the call. If you do not depress the ON/OFF button to respond to the call during this period, TMS abandons the call to free telephone resources.

6. The call terminates when any of the following occurs:

   a. You pick up the telephone set. In this case, TMS notifies the application program of two unsolicited events: Telset off-hook and off-line.

   b. The application program issues a hang-up command (QIO IO.HNG). Driver software sends a hang-up function command to TMS, then TMS unconditionally terminates the call and reports back with an off-line message, which the driver reports to the application as successful completion.

   c. You depress the ON/OFF hook button. TMS sends the button-depressed-code and line-released messages, followed by the off-line message, which the driver software reports to the application program (if unsolicited event notification is enabled).

Outgoing voice calls:

When TMS does outgoing calls for telemarketing, the process is exactly as for voice calls originated by application, described at the end of this section. The application provides a list of outgoing numbers and places calls sequentially under human supervision. Completion of one call causes the application to proceed to the next. (This application is not part of the TMS I/O Driver software.)

1.8.2 Serial Data Calls for Data Transfer

Manually originated data calls, with retries:

1. You start up a serial data application program and set parameters: the telephone number and whether to retry or not.

2. The program sets TMS characteristics to serial mode (the QIO IO.SMC function)and then specifies dialing the proper phone number (the QIO IO.CON function).

3. TMS takes line off-hook and outpulses the number. Pause commands could be present in the number and are handled as follows:

   a. The "," character causes an unconditional delay of 2 seconds.

   b. The "!" character causes a wait for dial tone. TMS proceeds with the next character in the dialing sequence either when dial tone is detected or after six seconds elapse.

   c. When two exclamation points appear together ("!!"), TMS waits 40 seconds for dial tone. This longer wait allows the user to wait for dial tones that take several wait cycles to occur, such as when accessing either a remote PBX or a dial-access common carrier.

4. After dialing, TMS sets a 30-second timer to timeout in the event of no carrier.

5. If the timeout occurs prior to carrier detection, TMS reports this to the application program as an error return from the IO.CON function.

6. If the program specifies a timeout of less than 30 seconds on the IO.CON and TMS does not return with carrier-detected in this time period, the driver software terminates the sequence and reports the status as IE.TMO.

7. In the event of an error, the application program determines whether to retry the connection, and when. Application programmers must be aware of any regulatory requirements in the locale where the programs are used, as to permitted frequency of retries. (Some regulatory or operating authorities might restrict the frequency of retries to prevent network blockage.)

8. If carrier is detected, serial communications is established, and data transmission can begin. During data transmission, an application has the full scope of serial features, such as large buffers, read timeouts, and so forth.

9. A call terminates when the application issues an IO.HNG function, which the driver software reports to the application if the application requests unsolicited event notification. If the application attempts to issue IO.RLB (read logical block) or IO.WLB (write logical block) after carrier is lost, these requests terminate with an error.

Automatically originated and terminated serial data calls:

Serial store and forward operates precisely as described above. The only difference is that the parameters (telephone number, and so forth) are supplied by the application program.

1.8.3 Voice Store and Forward

Store and Forward to Another TMS system: When calling another TMS system, the call should be originated in serial mode as described previously. Once a serial protocol, application dependent, is used to determine that the correct system is reached and that the remote system is prepared to receive a voice message, the following takes place:

1. One of the two systems (the application dependent protocol determines which, referred to here as system B) issues a "prepare to go voice" characteristic command, temporarily disabling loss of carrier disconnect. This system notifies the other system (system A), again by use of a serial protocol, that it is ready to switch to voice mode.

2. The application running on system A sets its mode to Codec (a QIO IO.SMC function). TMS on system A drops carrier, causing TMS on System B to also drop carrier, but not to disconnect. Each end now has a 30-second timer running to ensure that a new on-line mode is established. The TMS unit on system B reports loss of carrier, which the driver software relays to the application program. Then the application program sets the system B mode to Codec (an IO.SMC function).

3. At this point the two systems are ready to begin communication in Codec mode. By an application dependent method, the two systems will have already determined who will receive and who will transmit, so the applications can issue appropriate QIO commands to set direction and IO.ORG or IO.ANS QIOs to reestablish on-line mode.

4. TMS, when operating in Codec mode, uses a two minute fifty second timer to ensure effective call progress. Prior to the expiration of this timer, the two applications must communicate with each other using DTMF in order to maintain the call. The applications will have agreed prior to the establishment of Codec mode exactly how this exchange will take place. In the absence of an appropriate handshake taking place, TMS must terminate the call.

1.8.4 Manual Voice Calls at Your Desk

Dialing from the TMS voice unit:

1. With the Professional turned on, you press the voice unit's ON/OFF button. The LED adjacent to the ON/OFF button lights.

2. TMS enters on-line voice mode, conditional only on the telephone set being on-hook, and preempts any resources in use by any other process.

3. You dial a telephone (waiting for intermediate dial tones, if necessary). TMS allows you to dial at any speed and stores digits until they can be outpulsed. If the line is in rotary mode, depressing the "#" key temporarily switches the line to DTMF mode to permit the use of dial-access computers or common carriers.

4. You listen for ringing, busy, or network error messages.

5. If no effective call is established, you press the ON/OFF button to discontinue the call.

6. You talk to the desired party. Dialing as described above is permitted to allow the use of dial-access computers or common carriers.

7. TMS releases the line when you either depress the ON/OFF button or you pick up the telephone handset. The call can be continued using the telephone handset, but software control terminates. If the line is in temporary DTMF mode, it returns to rotary mode when it goes off-line.

8. When TMS releases the line, the LED adjacent to the ON/OFF button goes out.

Transferring a telephone handset call to the voice unit:

1. Telephone handset is already off-hook.

2. You depress ON/OFF. The LED next to the ON/OFF button lights.

3. TMS goes off-hook, but does not go on-line until it detects a telephone handset on-hook status. The time-out period is 30 seconds.

4. You place the telephone handset on-hook.

5. TMS enters on-line voice mode.

6. Talk to the called party using the TMS Voice Unit.

7. TMS releases the line when you either depress the ON/OFF button or when you pick up the telephone handset. For a line temporarily in DTMF mode, the dial mode returns to rotary mode upon going off-line.

8. When TMS releases the line, the LED adjacent to ON/OFF goes out.

1.8.5 Voice Call Originated by an Application Program

1. The application program sets the line to voice mode and specifies the telephone number to call (the TMS SF.SMC and IO.CON functions).

2. TMS takes the line off-hook, mutes the telephone handset if present, outpulses the desired number.

3. If the telephone handset is off-hook (voice mode only), TMS cancels the connect command with a preempt message to the application program as an error completion of the originate function (IO.ORG). If the application requests interrupt on unsolicted event notification, TMS reports "Telset-off-hook".

4. Upon completion of dialing, TMS immediately reports on-line status to the application program, connects speaker and microphone and begins flashing the LED adjacent to the ON/OFF switch (240 impulses per minute). TMS starts its 30-second timeout timer.

5. Upon receipt of on-line message, the driver software notifies the program by successful completion of the IO.ORG function.

6. You listen for ringing, busy, or network error messages and take the call by depressing the ON/OFF button to cancel the 30-second timer and stop the LED flashing. There is no notification when the ON/OFF button is depressed to cancel the timer. If you do not press the button within the timeout period, TMS abandons the call. An off-line message is sent as an unsolicited event to the application program by the driver software.

End of Chapter

CHAPTER 2

SYSTEM DESCRIPTION

This chapter describes the interface between TMS and the Professional computer, the functions of TMS system components, and their specific role in the management of telephone communications. Major functional components of the TMS system include:

- TMS I/O Driver Software Interface, the system-level software included with P/OS. The I/O driver controls an application's access to TMS hardware components.

- TMS Controller Unit, a printed-circuit board with: RAM/ROM memory for TMS commands, data, and shared access areas, modem facilities for serial data communications, a dual-tone multi-frequency (DTMF) receiver/transmitter for pushbutton telephone signal processing, the TMS 8-bit micro-processor, and a Codec for recording and playing back voice messages.

- Telephone Line Interface Unit connecting TMS directly to individual telephone lines.

- Voice Unit, a separate optional desk-top keypad unit that combines "no-hands" telephoning with message recording and playback capabilities.

2.1 TMS/PROFESSIONAL INTERFACE

Professional-to-TMS communication requires the TMS I/O driver, a standard RSX QIO software interface that is provided with P/OS. The driver provides a the link between application software and TMS firmware. Chapter 3 provides a detailed description of each of the TMS I/O driver functions. The TMS I/O driver encompasses all communication between P/OS and TMS. There are two types of interaction: (1) commands from the driver to TMS that control

TMS hardware activity and telephoning functions, and (2) event reports from TMS to P/OS reporting on TMS activity, unscheduled events, and hardware status changes. TMS firmware directs the operations of telephone-interface hardware as instructed by TMS I/O driver function commands. Driver requests are made by the application program running on the Professional. When the hardware reacts to these operations, TMS firmware detects any changes in the hardware state (incoming call sensed, on-board modem characteristics changed, and so forth) and reports these changes to the application program for further instructions.

For example, TMS could interrupt with a detection report about an incoming call on one of its phone lines; then, the application program could initiate an "answer the phone" function that the TMS I/O driver would interpret, causing TMS to answer the incoming call. With the phone answered and connection established, further application program/TMS interaction could determine the call type and provide the application program with sufficient information to allow a choice of appropriate actions.

2.2 TMS CONTROLLER UNIT FUNCTIONS

The TMS controller unit is an option board on the Professional's bus. Controller unit components enable TMS to interpret and execute requests for telephone services, to transmit and receive data, and to detect status changes in connected telephone lines. The major controller component functions described here include:

- Modem capabilities
- DTMF functions
- Codec capabilities

Additional information on the TMS controller components is provided in Chapter 6.

2.2.1 Modem Capabilities

The TMS controller includes on-board modem facilities that handle serial data and link the TLI's phone lines to TMS through the parallel input-output port. The standard TMS modem is the asynchronous 212A modem used in TMS. This low-to-medium-speed modem provides the capabilities for transmitting and receiving serial digital data over telephone lines. The modem provides: (1) bi-directional 103 frequency shift keying (FSK) at 75 to 300 baud and (2) high-speed differential phase shift keying (DPSK) capabilities at 1200 bps.

TMS includes the same low-speed capabilities as the 103 modem, as well as half-duplex operation compatible with 202 modems at speeds up to 1200 bps.

2.2.2 DTMF Facilities

TMS handles dual-tone multi-frequency (DTMF) signaling with an on-board DTMF receiver/transmitter. DTMF tones are used for incoming and outgoing functions (including both dialing and data transmission) and for tone deciphering of incoming telephone push-button signals. TMS has the capability of operating, not only the twelve tones on a normal telephone, but also can transmit an additional four tones, called A, B, C, and D, which make up a fourth column of tones as included on the DTMF Standard for future use. The values used in DTMF are provided with the description of the connect-line (IO.CON) function described in Chapter 3.

2.2.3 Autodialing

TMS provides a dialing capability, along with telephone number storage. The I/O driver functions that initiate a call and dial a telephone number are described in Chapter 3. Refer to the IO.CON function description.

2.2.4 Dial Modes

TMS is capable of telephoning in one of four possible dial modes. The choice of dial mode is dictated by the telephone equipment connected to the telephone line. For example, dialing over certain lines cannot exceed a 10-pulse-per-second rate without losing information. The reason for the rate limit lies in the type of line switches used with many telephone systems.

TMS' selectable dial modes are:

Pulse mode, 10 pps	An interruption rate of 10 pulses per second, usually required for telephone lines equipped with rotary dial phones.

DTMF (Dual-Tone Multi-Frequency)	Two groups of four signaling frequencies that provide up to 16 combinations of tone and enable transmission rates higher than pulse-mode lines. DTMF lines are present when the telephone supplier's line equipment supports tone-dialing service.
Pulse mode, 20 pps	A relatively fast rotary dial service with lines that permit an interruption rate of 20 pulses per second.
Off-hook Service	The off-hook dial mode can be used with telephone equipment connected to fixed, hard-wired, dedicated telephone lines or special service lines where the telephone equipment contains a pre-specified number, which eliminates the need for a user to dial. A dedicated "red-phone" connecting two parties with a single dedicated line with a telephone at each end is an example of off-hook service.

2.2.5 Speech Handling

TMS' on-board Codec provides for voice communication over either of TMS' telephone lines or from the optional TMS voice unit. The TMS codec is an integrated Continuous-Variable-Slope-Detect (CVSD) codec. Tone and speech synthesis is not supported.

Storage characteristics: Table 2-1 lists the amount of voice data that can be handled by TMS. Limits are imposed by the storage space available on your Professional.

Table 2-1: TMS Voice Storage Requirements

Conversation Length	Storage Space Required
4 Minutes	1 Megabyte
8 Minutes	2 Megabytes

There is a silence detection algorithm that can reduce storage requirements; the amount of storage space saved depends on the characteristics of the speech being recorded. Varying levels of telephone connections provided by local telephone suppliers can affect the performance of the TMS silence-detection mechanism. Silence detection and reduction can be enabled/disabled by the set-multiple-characteristics (SF.SMC) function. (See Chapter 3.)

2.2.6 Speech Buffers

Any program receiving voice from the TMS codec must be able to store data continuously without exhausting limited on-board buffering. A program transmitting voice data must be able to fill buffers continuously. A double-buffering algorithm should be used to ensure no loss of data.

2.3 TMS STATUS AND ERROR DETECTION

TMS monitors its own system status, telephone line events, and any error conditions. Each change in status or new event is noted. When TMS is enabled for interrupts, individual events are reported by the TMS on-board processor. (See Chapter 5 for details on event reporting from TMS.)

2.4 TELEPHONE LINE INTERFACE (TLI)

The TMS Telephone Line Interface (TLI) Unit provides connections for tying TMS directly into individual telephone lines and to the external voice unit. The TMS controller links the TLI to the Professional.

The TLI contains jacks which form an interface between physical telephone lines and the TMS controller. Each line has various capabilities and dedicated functions described in the following subsections. Figure 2-1 shows the back panel containing TMS telephone line jacks.

TMS TLI provides two jacks to telephone lines, one jack for a telephone set, a DIN connector for the TMS voice unit cable, and the TMS bypass switch. Two of the back panel jacks are reserved for Line 1 -- one for the external phone connection, and one for the user-provided telset. The bypass switch allows disabling of TMS in the event of an emergency requiring immediate use of the telset. (You cannot attach a telset to Line 2 through TMS.)

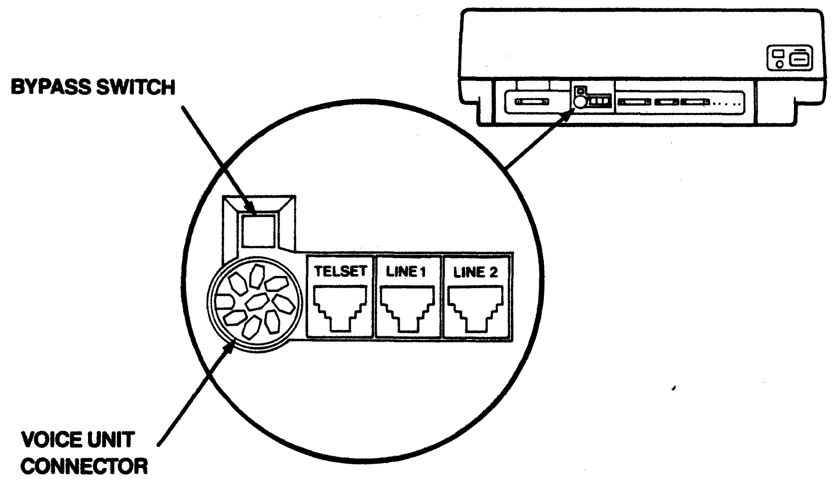

Figure 2-1: TMS Telephone Line Interface (TLI) Back Panel

2.4.1 TMS Telephone Line States

TMS firmware reports status changes for the three TMS lines. A line can be on-line or not, or can be on-hook or off-hook. These terms are defined in the following section.

On-line	Required resources are allocated successfully. All the conditions necessary for communication in a particular data mode are established.
On-hook	Telephone connection is idle (hung-up).
Off-hook	Telephone connection is active.

2.4.2 Special Line Features

Special line status changes also can be detected by TMS. Whenever TMS has interrupts enabled, it can notify the Professional of special features activity. Some of TMS' special features include:

Escape sequence checking	TMS can be programmed to listen for a special DTMF password and notify the Professional.

Zip tone	An informational tone provided by TMS when answering in voice mode to indicate to both the calling party and to a TMS user that a new call is present. (This feature is useful when doing telemarketing with auto-answer enabled.)

2.4.3 Telephone Connections

TMS can be connected to several different jacks, depending on the application requirement. Possible jacks are:

- RJ11C 6-Position Jack
- RJ13C 6-Position Jack
- RJ34X 8-position Jack
- RJ35X 8-Position Jack

For details, see FCC Rules and Regulations, Volume X, Part 68.502. A keyset adapter option merges the telephone handset and line 1 jacks for attaching a multi-line keyset rather than a single-line telephone handset.

2.5 OPTIONAL VOICE UNIT

The TMS optional voice unit is a separate desk-top keypad unit combining "no-hands" telephoning with message recording and playback capabilities. The voice unit contains a speaker, microphone, and a keypad for dialing and/or key signalling TMS functions. A serial cable connects the voice unit to a DIN connector on the TMS TLI back panel.

The voice unit houses logic enabling transmission and reception of signals to and from both TMS and the Professional. The keypad switches can be used for signalling; also, signals from TMS and/or the Professional can control the speaker and microphone, and can turn on and off voice unit LED indicators. Figure 2-2 shows the keypad and LED indicators (A), along with control switches (B) found on the voice unit. For programmable switch and indicator descriptions, see the auxiliary keyboard input section in Chapter 4.

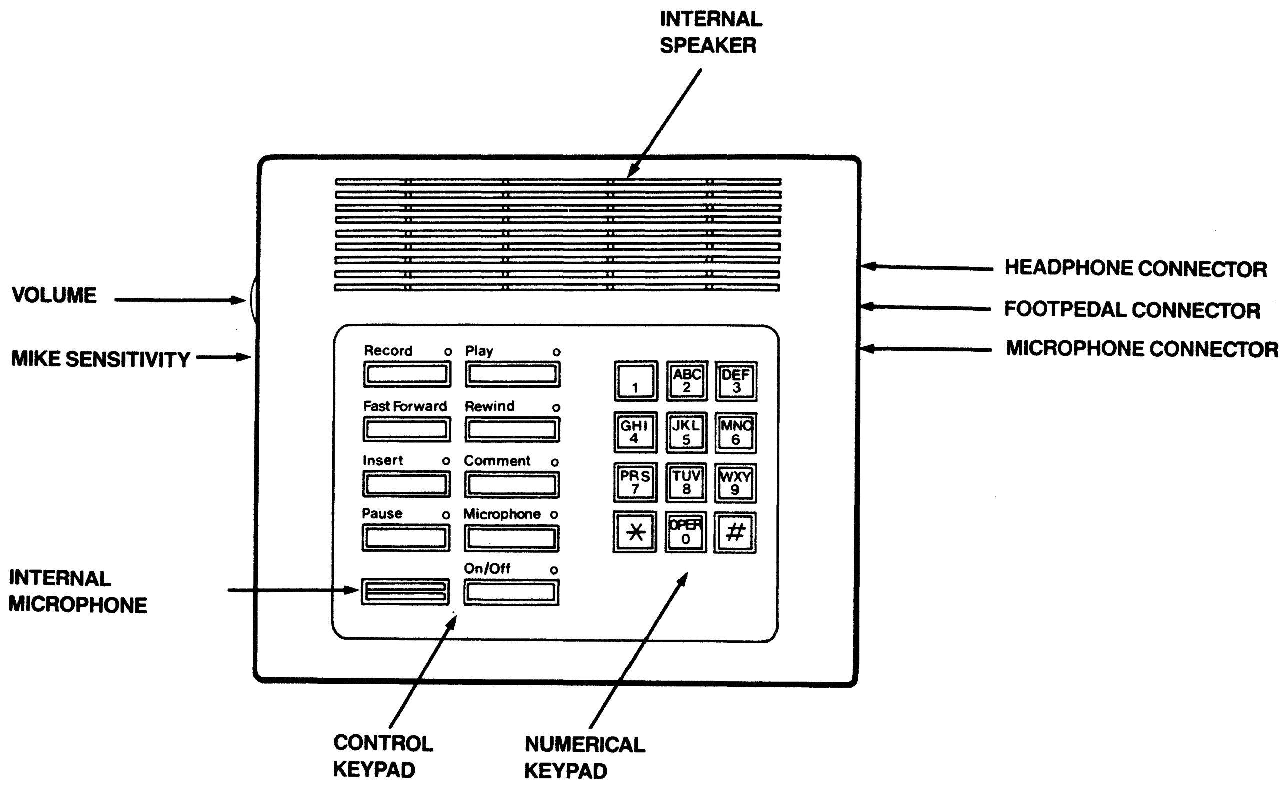

Figure 2-2: TMS Optional Voice Unit Controls/Indicators

End of Chapter

CHAPTER 3

DRIVER CALLS TO TMS

The TMS I/O driver software directs TMS operations under control of P/OS. The driver interprets and initiates calls from an application program to the TMS processor. See Chapter 2 for details on communication between the driver and TMS components.

This chapter contains descriptions of each of the driver calls to TMS. Table 3-1 groups the TMS I/O driver calls by general function for: initializing TMS, placing outgoing telephone calls, answering incoming calls, transmitting data, receiving data, and terminating TMS operations. The table briefly describes each call.

The TMS calls in this chapter are arranged in alphabetical order by function code name. Each call description begins with a definition of the call's function. The description provides the call format and parameter definitions. Notes describe operational effects, results, and special considerations for the call. Call-specific error messages are listed at the end of each description. The sequence and interaction of the various calls is illustrated by the application descriptions at the end of Chapter 1.

As an alternative to using the QIO functions described in this manual, TMS set-up and control operations can be handled by the Professional's communications services. Professional service routines allow you to control TMS operations from high level languages.

TMS-specific communications services provide library routines that control TMS line mode, enable/disable the voice unit keypad, handle switches to voice mode, and set the TMS DTMF sequence for a line. General Professional base system services and the PRO/Communications option provide callable routines for other TMS functions as well. See the Tool Kit User's Guide for descriptions of P/OS communications facilities, communications programming information, and the communications services library (COMLIB).

Table 3-1: TMS Driver Functions

Initialization	
IO.ATA	Attach a task to a line and report unsolicited events
IO.ATT	Attach a task to a line without event reporting
IO.LTI	Monitor an unattached line for unsolicited events
SF.GMC	Retrieve existing characteristics of a line
SF.SMC	Initialize or modify characteristics of a line
IO.BRK	Issue a break or long-space
Placing Calls	
IO.CON	Dial a specified phone number and connect a line to it
IO.ORG	Initiate a call on an existing connection
Answering Calls	
IO.ANS	Answer an incoming call
Transmitting Data	
IO.WAL	Write logical data block and pass all bits
IO.WLB	Write a logical data block
IO.WSD	Write special data to modify voice unit controls/indica
IO.WVB	Write a virtual data block
Receiving Data	
IO.RAL	Read a logical data block, pass all bits
IO.RLB	Read a logical data block and echo data
IO.RNE	Read a logical data block, do not echo data
IO.RVB	Read a virtual data block and echo data
Termination	
IO.DET	Detach a task from a line
IO.HLD	Place a line on hold
IO.HNG	Hang up a line
IO.KIL	Cancel any current or pending TMS functions
IO.UTI	Discontinue monitoring a line for unsolicited events

3.1 CALLING PROGRAM LANGUAGES

You can call TMS I/O driver functions from any of the Professional's Tool Kit program development languages. For generic call information, see the appropriate Tool Kit language manual and/or user's guide.

3.2 TMS I/O DRIVER CALL FORMAT

To request any of the TMS I/O driver services, use the Queue I/O (QIO) system directive. The QIO directive instructs P/OS to place an I/O request into the device queue for TMS. The system transfers the request to TMS according to the request priority and TMS availability.

P/OS clears the I/O status block when the request is queued and then sets the block to the final I/O status once the request completes. Optionally, you can select event flag notification and I/O completion asynchronous system traps (ASTs).

This section describes the MACRO-11 QIO call format for the TMS I/O driver. A summary description of the generic QIO parameters and some sample calls follow the QIO format. Each TMS call description in this chapter contains detailed definitions of TMS call-specific parameters. In the call descriptions, general QIO parameters are omitted and represented by an ellipsis (...). For calling information from high level Professional programming languages, see the _Tool Kit User's Guide_ and/or the appropriate language manual(s).

Each QIO directive requires a specific TMS I/O function code, the logical unit number for the TMS device, any selected optional QIO parameters, followed by any TMS-specific parameters. The MACRO-11 call format is:

```
QIO[W] fnc,lun,[efn],[pri],[isb],[ast],TMS-parameters
```

Required parameters include fnc (function code), lun (logical unit number), and most TMS parameters. Brackets ([]) indicate the call parameters that are optional. The event flag number is required with the synchronous wait form, QIOW. (For the system to parse MACRO-11 calls correctly, unused optional parameters must be represented by a comma delimiter in place of the argument.)

3.3 GENERAL QIO PARAMETERS

The following sections summarize those parameters expected for each MACRO-11 TMS I/O driver call. In the call format, the general QIO parameters precede any TMS-specific parameters.

Required QIO Parameters

fnc	I/O function code (IO.ATT, IO.ANS, IO.LTI, etc.)
lun	logical unit number for one of the TMS lines (a lun assigned in the program to XT1, XT2, or XT3, logical names representing TMS lines 1, 2, and 3, respectively).

Optional QIO Parameters

efn	event flag number (required with the QIO-and-wait form of the directive) (1-64/clear is queued; set is completed)
pri	priority (ignored)
isb	address of a two-word I/O status block
ast	address of AST service routine entry point
TMS	TMS-specific parameter list <P1,...P6>

(See function descriptions further on for parameter specifics.)

TMS returns error and status messages as described in chapter 5 of this manual. Chapter 5 also identifies the types of unexpected events that occur during TMS operation (incoming rings, loss of carrier during a call, and so forth.)

3.3.1 Sample Call to the TMS Driver

The following sample call illustrates the type of call statement used to direct the TMS I/O driver. Note that data definitions are not provided for the sample TMS call.

```
QIOW$S #IO.RLB!TF.TMO,#TLUN,#TEFN,,#IOSB,,<#BUFF,#BUFSIZ,#RDTMO>
```

This call statement performs a read function (read logical block) to receive information on the specified TMS telephone line (TLUN). TMS sets TEFN upon I/O completion, and returns call status to IOSB. The TMS-specific parameters BUFF and BUFSIZ define the input buffer provided for the received data. The RDTMO parameter specifies a timeout period in seconds for the read function, returning the total number of bytes read to the second word of IOSB. For details, refer to the read function description further on in this chapter.

3.4 CALL DESCRIPTIONS

The remainder of this chapter contains TMS call descriptions. The descriptions are presented in alphabetical order by function code name. A summary of the TMS I/O driver calls is provided in an appendix.

3.4.1 Answer a Line (IO.ANS)

The answer function establishes a connection in response to a ringing line or explicitly by answering an existing connection. This call directs TMS to take the line off-hook and to process the call according to the line`s current data mode.

The answered line switches to on-line status immediately for voice and Codec modes. For serial mode, on-line occurs after modem carrier is established; for DTMF mode, on-line occurs after sending a DTMF tone to the other party.

Format:

```
QIO[W]$S #IO.ANS,#lun,...,<TMS-specific-parameters>
```

lun	A logical unit assigned to XT1, XT2, or XT3
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

dbf	A data buffer required by P/OS (No data transferred)
size	A size value required by P/OS (e.g., #I)

Notes:

For line 1 (XT1), the answer command first checks current line status; if the telephone handset is off-hook, the answer command waits for an on-hook status for 40 seconds before timing out and cancelling itself. While waiting for a hung-up status, TMS takes the specific line off-hook. Once the telephone handset is on-hook, the answer command marks the line's status as on-line (off-hook). Then TMS determines the line's data mode to resolve resource allocation.

In voice, DTMF, and Codec modes, IO.ANS temporarily attaches the DTMF receiver (if available) to accept unsolicited DTMF characters. When DTMF escape-sequence checking is enabled, this function retains the DTMF receiver for 15-seconds waiting for a DTMF tone, then releases it in voice mode. In Codec mode there is no time-out; TMS keeps the DTMF receiver on the line.

If the line is already on-line, IO.ANS always returns success

Effects of the IO.ANS function vary, according to the line's data mode, as described in the following paragraphs:

DTMF, Voice, and Codec Modes	The answer function completes successfully if necessary resources are available. TMS reports any characters arriving (even if the characters are part of an escape-sequence) as unsolicited DTMF tone events. The escape-sequence, when received, is reported as an unsolicited event, escape-sequence received.
DTMF Mode	TMS attaches available DTMF resources and answers the line by applying the answer tone (a DTMF pound sign, #) for the period specified for DTMF tone length. TMS checks the escape-sequence until the first read command is issued. If the DTMF receiver is attached to another line, this command terminates at once with device-not-ready (IE.DNR) error.
Serial Mode	TMS attaches serial resources and performs the appropriate modem protocol. However, the answer does not complete until carrier is established. If carrier is not present, the call terminates with a carrier-not-received error message.
Voice Mode	TMS checks for availability of speaker and and microphone; if either is allocated to another line, the answer command terminates with a fatal error. If both are available, TMS first attaches the speaker to the outgoing line and applies a zip tone of 941 Hz for 200 milliseconds. Then TMS attaches both speaker and microphone. If the DTMF receiver is available, TMS also checks the escape-sequence for 15 seconds.

Codec Mode	TMS attaches and enables the codec. If the DTMF receiver is free, TMS attaches it. If enabled, TMS does escape-sequence checking and unsolicited DTMF tone reporting. (There is no time-out for the DTMF receiver checking in Codec data mode.)

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.DNR	Device not ready. Could not answer the specific line within the time-out period specified in the call, or in serial mode on a line connected to a modem without carrier present.
IE.CNR	Connection rejected.
IE.RSU	Shared resource in use, unable to answer.

3.4.2 Attach Line with ASTs (IO.ATA)

The attach-device/specify-unsolicited-input-AST function, a version of the simpler attach (IO.ATT) function, specifies asynchronous system traps (ASTs) to process unsolicited events coming in from TMS. Refer to Chapter 5 for additional information on events. When you attach a line with this function, no other program can interrupt control until your program terminates or releases the line with a detach (IO.DET).

Format:

```
QIO[W]$S #IO.ATA,#lun,...,<TMS-specific-parameters>
```

lun	A logical unit assigned to XT1, XT2, or XT3.
...	Optional general QIO parameter(s)

<TMS-specific- parameters>

ast	Address of an asynchronous system trap routine
par2	User-specified optional one-byte parameter passed to the AST anytime an event occurs. (For example, par2 could be 1 or 2 indicating the TMS line reporting the event.)

Notes:

The unsolicited event types reported are:

XTU.CD	Carrier detected (returned in serial mode and to report on-line transitions in other modes)
XTU.CL	Carrier lost in serial mode and disconnect for all modes
XTU.OF	XOFF received
XTU.ON	XON received
XTU.RI	Ringing detected (with auto-answer disabled)
XTU.UI	Unsolicited input (serial and DTMF modes)
XTU.DR	DTMF escape-sequence received
XTU.TU	Telephone handset taken off-hook (Line1 only)

XTU.TD	Telephone handset placed on-hook (Line1 only)

When the event type is XTU.UI indicating unsolicited input, the AST becomes "disarmed" until the task issues a read request. Once the read request completes, the AST is armed again for new unsolicited events. A read-with-zero-timeout can be used to catch all ASTs. (Refer to the read function description further on in this chapter.)

Unsolicited input from the voice unit's auxiliary keyboard and unsolicited DTMF input is indicated by the low byte returned. When the low-order byte in the top word in the stack is negative, it is the negated value of the key depressed on the keyboard or the negated value of the DTMF code. See Chapter 5 for keyboard values and refer to the IO.CON description in this chapter for DTMF values.

Note that DTMF unsolicited input always comes in on TMS lines 1 or 2, while unsolicited input for the voice unit keypad always arrives on line 3.

An IO.KIL does not detach an attached line.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted.
IE.DAA	Device already attached to calling task.

3.4.3 Attach Line (IO.ATT)

The attach function reserves the TMS line specified by the call's logical unit number parameter. Once you attach a line, no other program can interrupt control until your program releases the line with a detach (IO.DET) call or until your program terminates. Refer to the IO.ATA call for a line-attaching call that also enables ASTs.

Format:

```
QIO[W]$S #IO.ATT,#lun,...,
```

lun	A logical unit assigned to XT1, XT2, or XT3
...	Optional general QIO parameter(s)

Notes:

An IO.KIL does not detach an attached line.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted.
IE.DAA	Device already attached to calling task.

3.4.4 Issue a Break (IO.BRK)

The break function specifies sending either a long-space or break signal on the addressed line. Some systems use break and long space as special control characters. A long space is a 3.5-second space state on the line. A break is a 300-millisecond space state on the line.

Format:

```
QIO[W]$S #IO.BRK,#lun,...,<TMS-specific-parameter>
```

lun	A logical unit number assigned to line 1 or 2
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

type	0 = break 1 = long space (parameter1)

Notes:

If the specified TMS line is not in serial data mode, this function is ignored and returns immediate success.

With the $S form of the macro, the type parameter's 0 or 1 must be either preceded by a pound sign (#), contained in a register, or indirectly referenced.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted.

3.4.5 Connect Line (IO.CON)

The connect function directs TMS to dial the specified telephone number and connect the indicated TMS line to the dialed number Phone numbers are encoded as ASCII input that corresponds to dialing digits. See the "notes" part of this description for a table of ASCII/dialing values.

The connection takes place in the line's current data mode. An optional time-out parameter allows you to specify abandoning the call before the hardware time-out period elapses.

Format:

```
QIO[W]$S #IO.CON,#lun,...,<TMS-specific-parameters>
```

lun	A logical unit assigned to TMS lines XT1 or XT2
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

phone number	Address of the stored telephone number
size	Size of the phone number buffer
[time-out]	Optional wait period value in secs. (0-30)

Notes:

For line1, the telephone handset is temporarily disconnected during the dialing process.

TMS evaluates the optional time-out parameter in the following manner: the low byte specifies the time-out in 10-second multiples; any high byte value is added to this value as one-second multiples. If the result is greater than 255, the value is changed to 255. For example, time-out values of 1 or 2560 both specify a 10-second time-out; 2561 specifies 20 seconds; 256 is one second and 257 is 11 seconds. This resolution of value also applies to the IO.ORG and TMS read functions.

For the time-out parameter to have an effect, the I/O function must be IO.CON!TF.TMO.

TMS abandons dialing unless the tone detector is available to detect dial tone.

An automatic hardware time-out occurs when the line's data mode is DTMF and no DTMF tone is detected. Codec mode always completes immediately after dialing.

Telephone numbers are variable-length. The maximum size of the phone number buffer (after any translation) must be no larger than 48 bytes, including special characters used for access pauses and any optional start and end sequences. Table 3-2 lists all legal characters for TMS telephone numbers.

Table 3-2: Legal Characters for TMS Telephone Numbers

Character	Usage
0-9	Digits zero to nine.
!	The exclamation mark character causes an access pause to wait six seconds for an initial or intermediate dial tone(s).
!!	Two exclamation marks cause a longer access pause (40 seconds) for intermediate dial tone used when significant network switching time is needed (more than a few seconds).
,	The comma character provides a fixed 2-second delay.
#	The pound-sign used in DTMF mode, sends the tone for the symbol. In interrupted pulse mode, indicates a temporary change to DTMF for the rest of the number. (Used where mixed-mode dialing permitted; for example, where local telephone systems support only interrupted dialing, but calls are to be switched further through a remote DTMF access port reached via the public switched telephone network.
* A B C D	These character codes are valid only when dialing in DTMF mode, where they generate the defined tones for each of these symbols.

Table 3-3 lists the ASCII/DTMF dialing codes.

Table 3-3: DTMF/ASCII Dialing Codes

ASCII CHAR.	<OCTAL> VALUE	HEX VALUE	KEYPAD BUTTON
"0"	<60>	30	0
"1"	<61>	31	1
"2"	<62>	32	2
"3"	<63>	33	3
"4"	<64>	34	4
"5"	<65>	35	5
"6"	<66>	36	6
"7"	<67>	37	7
"8"	<70>	38	8
"9"	<71>	39	9
"*"	<52><72>	2A,3A	*
"#"	<43><73>	23,3B	#
	Auxiliary DTMF ASCII Codes		
"A"	<101><141><74>	41,61,3C	A
"B"	<102><142><75>	42,62,3D	B
"C"	<103><143><76>	43,63,3E	C
"D"	<104><144><77>	44,64,3F	D
","	<54>	2C	Pause 2 secs
"!"	<41>	21	Pause 3 secs, await dial tone
"E"	<105><145>	45,65	697 Hz
"F"	<106><146>	46,66	770 Hz
"G"	<107><147>	47,67	852 Hz
"H"	<110><150>	48,68	941 Hz
"I"	<111><151>	49,69	1209 Hz
"J"	<112><152>	4A,6A	1336 Hz
"K"	<113><153>	4B,6B	1477 Hz
"L"	<114><154>	4C,6C	1633 Hz

NOTE

Multiple octal/hex codes are interchangeable on ASCII values * through L when TMS transmits DTMF data; however, received DTMF values are in the range 60 to 77, as shown in this table. For example, for the DTMF ASCII for "A", TMS can transmit any of the values (<101>,<141>, or <74> in octal and 41, 61, or 3C in hexadecimal), but TMS receives the A only as <74> or 41.

For a hook-flash before dialing, the first character of the number must be the caret (^) to cause the hook-flash before dialing the number. When the number consists only of a hook-flash caret, TMS performs the hook-flash and does not dial anything further (not even the optional start and end sequences).

The effects of the IO.CON vary, according to the line's current data mode, as described in the following paragraphs.

DTMF Mode	TMS attaches DTMF resources. Connection is successful when TMS receives a DTMF tone, or after the hardware time-out period elapses, whichever occurs first.
	Serial Mode TMS attaches serial resources, and when dialing completes, begins conditioning the appropriate modem. The call terminates with a "device not ready" status if carrier is not present before either the user-specified or hardware time-out period elapses after dialing completes.
	TMS checks for availability of both speaker and microphone; if either is allocated to another line, the command terminates with a fatal error. With both available, TMS attaches speaker and microphone to the outgoing line.
Codec Mode	TMS attaches the Codec and the DTMF receiver. IO.CON completes immediately on completion of dialing.

For line 1 only, when in serial, Codec, or DTMF mode, if the telephone handset is off-hook at the end of the dialing, TMS waits 30 seconds for the set to go on-hook before timing out. In voice mode, with the telephone handset off-hook when dialing completes, the function terminates with a message, and the user can handle the call on the telephone handset in the usual manner.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted.
IE.ALC	Allocation failure when size of phone number (plus the start and end sequences) exceeds size of TMS buffer.

IE.CNR	Connection rejected because carrier is already present when IO.CON call issued.
IE.DNR	Device not ready. Could not connect the line within the default time-out period or in the time-out period specified; alternatively, could not connect in serial mode when connected to a modem without carrier.
IE.RSU	Shared resource in use, unable to connect
IE.BAD	Invalid phone number digit or call issued for line 3.

3.4.6 Detach Line Request (IO.DET)

The detach function relinquishes access to a specific TMS telephone line. Following the detach, the line can be accessed by any program.

Format:

```
QIO[W]$S #IO.DET,#lun,...,
```

lun	A logical unit number assigned to line 1, 2, or 3
...	Optional general QIO parameter(s)

Notes:

If a program that attached the line exits without issuing an IO.DET, the operating system executive automatically detaches the line for use by other programs.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted

3.4.7 Place a Line on Hold Request (IO.HLD)

The function performs the same operation as IO.HNG. However, if hold-control is enabled, TMS indicates to the local telephone equipment that the call is to be placed on hold rather than terminated. (From TMS' standpoint, the call is released fully.)

Format:

```
QIO[W]$S #IO.HLD,#lun,...,
```

lun	A logical unit assigned to XT1, XT2, or XT3
...	Optional general QIO parameter(s)

Notes:

None

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted

3.4.8 Hang Up a Line Request (IO.HNG)

The hang-up function releases a TMS line and any TMS resources currently in use by the line. For Line1 and Line2, line status changes to on-hook; for Line3, the function releases the Codec (if it is dedicated to the specified line).

Format:

```
QIO[W]$S #IO.HNG,#lun,...,
```

lun	Logical unit number assigned to line 1, 2, or 3
...	Optional general QIO parameter(s)

Notes:

Line status reverts to off-line. Line1 and Line2 go on-hook.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted

3.4.9 Cancel I/O Request (IO.KIL)

The kill function interrupts call processing, directs TMS to abort the currently executing instruction for that line and purges all of the line's pending I/O requests. The IO.KIL call overrides all other calls and executes immediately.

Format:

```
QIO[W]$S #IO.KIL,#lun,...,
```

lun	Logical unit number assigned to XT1, XT2, or XT3
...	Optional general QIO parameter(s)

Notes:

TMS responds to the IO.KIL call by immediately terminating the currently executing instruction and by purging all subsequent calls waiting for that line.

Pending TMS calls are purged to preserve control of subsequent TMS operations. Purged calls return a status code IE.ABO and activate an appropriate AST, if specified.

If the addressed TMS line is off-hook but not connected to another party, TMS sets the line on-hook (hangs up the line).

Due to the asynchronous nature of both the hardware and software, it is possible for an application to issue an IO.KIL as a connection is being established. The I/O status of the IO.CON, IO.ANS, or IO.ORG functions correctly indicate whether or not the connection occurred before the effect of the IO.KIL. If these functions indicate successful connection, the application must issue an IO.HNG to release the call.

Note that an IO.KIL completes immediately, however it can take a short time for other outstanding I/O requests to complete. The requests do not necessarily complete in the same order as issued.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted.

3.4.10 Link Task to Interrupts (IO.LTI)

The IO.LTI function arranges for notification of a specified task by any asynchronous system trap (AST) upon detection of an unsolicited event. This function is used to monitor a TMS line whenever another task is not specifically attached to the line. The IO.LTI function specifies a task to handle the unsolicited event. Refer to Chapter 5 for additional information on TMS event-handling.

Format:

```
QIO[W]$S #IO.LTI,#lun,...,<TMS-specific-parameters>
```

lun	A logical unit assigned to XT1, XT2, or XT3.
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

dbuf	Address of three-word buffer in the form: .WORD AST-address .RAD50 /first_half_of_task_name/ .RAD50 /second_half_of_task_name/
size	The size of the AST and taskname buffer (must be 6)
par3	User-specified optional one-byte parameter passed to the AST anytime an event occurs. (For example, par2 could be 1 or 2, indicating the TMS line reporting the event.)

Notes:

The unsolicited event types reported are:

XTU.CD	Carrier detected (returned in serial mode and to report on-line transitions in other modes)
XTU.CL	Carrier lost in serial mode and disconnect in all data modes)
XTU.OF	XOFF received
XTU.ON	XON received
XTU.RI	Ringing detected (with auto-answer disabled)

XTU.UI	Unsolicited input (serial mode only)
XTU.DR	DTMF escape-sequence received
XTU.TU	Telephone handset taken off-hook (Line 1 only)
XTU.TD	Telephone handset placed on-hook (Line 1 only)

When the event type is XTU.UI indicating unsolicited input, the AST becomes "disarmed" until the task issues a read request. Once the read request completes, the AST is armed again for new unsolicited events. A read-with-zero-timeout can be used to catch all ASTs.

Unsolicited input from the voice unit's auxiliary keyboard and unsolicited DTMF input is indicated by the low byte returned. When the low-order byte in the top word in the stack is negative, it is the negated value of the key depressed on the keyboard or the negated value of the DTMF code. See Chapter 5 for keyboard values and refer to the IO.CON description for DTMF codes.

Note that DTMF unsolicited input always arrives on lines 1 or 2, and voice unit keypad input always comes in on line 3.

An IO.KIL does not detach an attached line.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted
IE.ALC	Buffer size not 6

3.4.11 Originate Connection (IO.ORG)

The connect function directs TMS to connect to an existing party on the indicated line. Prior existence of connection is assumed.

The connection takes place in the line's current data mode. An optional time-out parameter allows you to specify abandoning the call before 40 seconds elapse.

Format:

QIO[W]$S #IO.ORG,#lun,...,<TMS-specific-parameters>

lun	A logical unit number assigned to XT1, XT2, or XT3.
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

dbuf	A data buffer (required by P/OS)
size	Size of the data buffer
[time-out]	Optional waiting period value in seconds (0-60)

Notes:

The line goes off-hook (even if already off-hook). If the line is already on-line, this function immediately returns success. The line switches to on-line status immediately for voice and Codec modes. For serial mode, on-line occurs after carrier is established. For DTMF mode, on-line occurs after a DTMF tone is received from the called party.

For the time-out period to be used, the call's function must be IO.ORG!TF.TMO. A zero value for the optional time-out parameter returns successfully for an already-connected line; when not connected, the function returns the IE.DNR message and does not attempt connection. (This allows the application program to check to see if a line is on-line.) A non-zero value for the time-out parameter results in normal connection.

TMS evaluates the optional time-out parameter in the following manner: the low byte specifies the time-out in 10-second multiples; any high byte value is added to this value as one-second multiples. If the result is greater than 255, the value is changed to 255. For example, time-out values of 1 or 2560 both specify a 10-second time-out; 2561 specifies 20 seconds; 256

is one second and 257 is 11 seconds. This resolution of value also applies to the IO.CON and TMS read functions.

For line 1, when the time-out parameter and TF.TMO modifier are not specified and the telephone handset is on-hook, TMS returns the IE.DNR message. With the telephone handset off-hook the connection occurs as soon as the telephone handset is placed on hook unless the 40-second hardware time-out period elapses before the on-hook state. This allows manual originate mode on line 1.

Additional effects of the IO.ORG, according to the line's current data mode, are described in the following paragraphs.

DTMF Mode	TMS requires that the DTMF receiver be available in order to originate the connection successfully.
Serial Mode	TMS attaches serial resources and begins to condition the appropriate modem to originate frequency bands.
Voice Mode	TMS checks for availability of both speaker and microphone; if either is allocated to another line, the call terminates with a fatal error. With both available, TMS attaches speaker and microphone to the outgoing line.
Codec Mode	TMS attaches and enables the codec and DTMF receiver.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted
IE.DNR	Device not ready. Could not connect the line within default time-out period; alternatively, could not connect in serial mode when connected to a modem without carrier present.
IE.RSU	Shared resource in use. Cannot establish connection.
IE.TMO	Could not connect within the specified time-out period.

3.4.12 Read Functions (IO.RAL, IO.RLB, IO.RNE, IO.RVE)

The read functions are Pass All Bits (IO.RAL), Read Logical Block (IO.RLB), Read With No Echo (IO.RNE), and Read Virtual Block (IO.RVE).

These functions direct TMS to read incoming data and to pass the data to the requesting task. The characteristics of the TMS line determine whether any bits are stripped or whether control characters are recognized. For details on line characteristics, refer to the SF.GMC and SF.SMC function descriptions further on in this chapter.

A read function call specifies a byte count maximum for a receive buffer. Once data equals the requested byte count, TMS passes the data to the driver. An optional time-out parameter allows timed reads. The read-all-bits function, IO.RAL, overrides the eight-bit character (8BC) characteristic specified by the SF.SMC function and returns all bits passed by the TMS hardware.

Format:

```
QIO[W]$S #IO.rdf,#lun,...,<TMS-specific-parameters>
```

rdf	One of the read functions (RAL, RLB, RNE, or RVE)
lun	A logical unit number assigned to XT1, XT2, or XT3.
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

dbuf	A receive buffer for incoming data
size	Size of the data buffer (1-60 bytes)
[time-out]	Optional waiting period value in seconds (0-60)

Notes:

In serial data mode, the read function does not pass CTRL/Q or CTRL/S to the requesting task unless you disable XON/XOFF support (the TC.BIN characteristic).

In serial and DTMF data modes, when the TF.TMO sub-function is specified with a time-out of 0 seconds (the default value), the read command completes immediately, once it completes transfer of as many characters as are already received and stored in the type-ahead buffer (up to the receive buffer maximum as specified by the size parameter.

TMS evaluates the optional time-out parameter as follows: the low byte specifies the time-out in 10-second multiples; any high byte value is added to this value as one-second multiples. If the result is greater than 255, the value is changed to 255. For example, time-out values of 1 or 2560 both specify a 10-second time-out; 2561 specifies 20 seconds; 256 is one second and 257 is 11 seconds. This resolution of value also applies to the IO.CON and IO.ORG functions.

For the time-out parameter to have an effect, the I/O function must be IO.CON!TF.TMO.

When a 1-60 second time-out is specified, the read completes as soon as either the time-out period elapses or the specified number of characters transfers. The number of bytes transferred is returned in the second word of the I/O status block.

DTMF mode incoming data translates to ASCII according to the chart in Table 3-3 earlier in this chapter. Prior to the first read command in DTMF mode, DTMF characters received can cause a DTMF event. Once the first read is issued, DTMF characters are handled exactly as in serial mode. That is, the XTU.UI event occurs and characters are placed in a type-ahead buffer

In Codec mode, buffers as large as possible (preferably a multiple of 1024 bytes) should be specified. Double buffering must be used so there always is an outstanding read request to avoid loss of voice input. In Codec mode, read and write functions cannot be performed simultaneously. DTMF data is reported as a DTMF event.

In DTMF mode, although simultaneous read/write functions are permitted, because of telephone line characteristics, results are unpredictable. In addition, after completing any write functions in DTMF mode, it might be necessary to wait for the echo delay period and clear the type-ahead buffer to remove transmitted characters that echoed causing TMS to receive them. This requirement is most likely necessary on satellite connections.

Status:

IS.TMO	Successful completion of a read. The input from TMS terminated by the user-specified time-out parameter. (The input buffer contains the bytes read.
IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted. (Word 2 contains the total number of bytes transferred before cancelling.)
IE.DNR	Device not ready. Returned if a read function is attempted in voice mode when not on-line and whenever a read function is attempted in Codec while a write is active.

3.4.13 Unlink Tasks from Interrupts (IO.UTI)

The IO.UTI function halts monitoring of unsolicited events on an unattached line. (Refer to the IO.LTI function description and to Chapter 5 for details of the event monitoring process.)

Format:

```
QIO[W]$S #IO.UTI,#lun,...
```

lun	A logical unit number assigned to XT1, XT2, or XT3.
...	Optional general QIO parameter(s)

Notes:

(See IO.LTI)

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted.

3.4.14 Write Functions (IO.WAL, IO.WLB, and IO.WVB)

The write functions are Pass All Bits (IO.WAL), Write Logical Block (IO.WLB), and Write Virtual Block (IO.WVB).

These functions direct TMS to transmit data on the specified line in the line's current data mode. All write functions pass all bits of the data on the line. With this function, all outgoing information including control characters pass from the task issuing the write function. The write-special-data function (IO.WSD), the voice unit indicator control function, is described further on in this section.

Format:

QIO[W]$S #IO.wrf,#lun,...,<TMS-specific-parameters>

wrf	One of the write functions (WAL, WLB, or WVB)
lun	A logical unit number assigned to XT1, XT2, or XT3.
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

dbuf	A data buffer for outgoing data
size	Size of the data buffer (0 to 60 bytes)

Notes:

The line's data mode must be DTMF, serial, or codec. The TMS transmit-data command cannot be used with voice mode.

The addressed line must be on-line (connected) before the tranmit-data command executes. If the line is off-line, execution terminates with a fatal-error status.

DTMF mode outgoing data translates to ASCII according to the chart in Table 3-3.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending

IE.ABO	Operation aborted. (Word 2 contains the total number of bytes transferred before cancelling.)
IE.DNR	Device not ready. Write function issued with the line off-line or in voice mode, or, if in Codec mode, with a read outstanding.

3.4.15 Write Special Data (IO.WSD)

The write-special-data function directs TMS to modify the voice unit indicators according to the configuration specified in the call's accompanying data buffer. Indicator numbers and states are encoded as numeric input, listed later in this section.

Format:

```
QIO[W]$S #IO.WSD,#lun,...,<TMS-specific-parameters>
```

lun	A logical unit assigned to any TMS line
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

icb	address of the stored indicator configuration buffer
size	Size of the indicator buffer (maximum 30 bytes)

This driver call to a TMS line enables and disables one or more specific voice-unit indicators. At least one data byte must accompany the call. The accompanying data bytes are encoded values indicating the state (ON or OFF) and the specific indicator number to be affected. On/Off requests for the microphone/mute indicator are ignored.

Notes:

Octal values 101 and 141 operate on all software-controllable LEDs by turning them off (101) or on (141) as a group. (The microphone mute indicator does not respond to this value.)

Indicator configuration buffers are variable in length. The maximum size of the buffer must not exceed 30 bytes. Table 3-4 lists the legal values for TMS voice unit indicator states.

Table 3-4: Voice Unit Indicator States

Value	Usage
1-6	Digits one through six turn off the corresponding indicator(s) (indicators 1 through 6)
7	Invalid. The microphone-mute indicator is not under software control.
10-11	Octal digits 10 and 11 turn off the corresponding voice unit indicators, 10 and/or 11.
41-46	Octal digits 41 through 46 turn on the corresponding indicator(s)
47	Invalid value. The microphone-mute indicator is not under software control.
50-51	Octal digits 50 and 51 turn on the corresponding voice unit indicators.
101	The octal value 101 turns off all voice unit indicators, with the single exception of the microphone-mute indicator.
141	The octal value 141 turns on all voice unit indicators, with the single exception of the microphone-mute indicator.

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted.
IE.ALC	Allocation failure when size of configuration buffer exceeds 30 bytes.
IE.DNR	Device not ready.
IE.BAD	7, 47, or an out-of-range value specified

3.4.16 Get Multiple Characteristics (SF.GMC)

The SF.GMC function retrieves the identifying characteristics of the specified TMS line. Use this function to assess the current characteristics of a communications line, and use the SF.SMC function to set or to modify current line characteristics.

The line characteristics that TMS reports are listed in Table 3-5. Originating and receiving characteristics must be coordinated for intelligible data communications to take place. Refer to Chapter 4 for additional information on data line protocols and characteristics.

The TMS protocol defaults in effect on start-up are listed in the following table. These predefined TMS I/O driver defaults can be overridden by start-up options available through the P/OS system. (Refer to the P/OS system documentation for start-up options.)

Format:

```
QIO[W]$S #SF.GMC,#lun,...,<TMS-specific-parameters>
```

lun	A logical unit assigned to XT1, XT2, or XT3.
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

cbuf	Starting address of the characteristics buffer, a variable-length buffer with each characteristic represented by a two-byte field holding the TMS line characteristic-name and characteristic value in the form: .BYTE characteristic-name .BYTE 0 Characteristic-names and their possible values are listed in Table 3-5.
size	The size (total number of bytes) of the SF.GMC characteristics buffer.

Table 3-5: Line Characteristics (SF.GMC)

Name	Valid Values	Meaning
TC.ARC	0-20	Number of rings (1 to 20) to wait before answering an incoming call. A value of 0 disables auto-answering and enables interrupts on ringing.
TC.BIN	0,1	Enable (1) or disable (0) binary mode. (With binary mode enabled, XON/XOFF recognition/transmission disabled.)
TC.CTS	0,1	Status of data output is suspended (1) or active (0). (Serial data mode only)
TC.EPA	0,1	Odd (0) or Even (1) parity sense (ignored unless TC.PAR is enabled (1).
TC.FSZ	5-9	Character frame size (width in data bits, plus parity bit if enabled). See Note 1, Table 3-6 for values.
TC.PAR	0,1	Enable (1) or disable (0) parity checking and generation. See Note 1, Table 3-6 for values.
TC.RSP	(Table 3-7)	Set speed (baud rate) for receiving incoming data in bits-per-second. Table 3-7 in Note 2 lists valid TC.RSP and TC.XSP values. (Note that for Release 1 of TMS, transmit speed and receive speed must be the same.)
TC.STB	1,2	Number of stop bits at the end of each character.
TC.TBF	0-255	For SF.GMC, number of unprocessed characters in the line's input buffer. A value of 255 indicates a minimum of 255 characters to be processed.
TC.TRN	(Invalid)	Invokes error when used with SF.GMC. (Write only. Refer to SF.SMC description.)
TC.XMM	0,1	Hold control (low-order bit only).

Name	Valid Values	Meaning
TC.XSP	(Table 3-7)	Set speed (baud rate) for transmitting outgoing data in bits-per-second. Table 3-7 (Note 2) lists valid TC.RSP and TC.XSP characteristic values. (Note that the receive and transmit speed must be the same.)
TC.8BC	0,1	Pass 7-bit (0) or 8-bit (1) characters on input. (A TC.BIN value of 1 and/or the use of the IO.RAL (read-all) function overrides the TC.8BC value.)
XT.MTP	(Table 3-8)	Specifies one of the modem types listed in Note 3, Table 3-8.
XT.DLM	(Table 3-9)	Specifies one of the valid dial modes listed in Note 4, Table 3-9.
XT.DMD	(Table 3-10)	Specifies one of the valid data modes listed in Note 5, Table 3-10.
XT.DIT	1,255	Sets the length of the pause between dialed DTMF signals. (TMS multiplies value by 10 milliseconds. See XT.DTT.)
XT.DTT	1,255	Sets the length of time to apply a dialed DTMF signal. (TMS multiplies value by 10 milliseconds. See XT.DIT.)
XT.GOV	(Invalid)	Invokes error when used with SF.GMC. (Write Only. Refer to SF.SMC details.)
XT.SDE	(Invalid)	Invokes error when used with SF.GMC. (Write Only. Refer to SF.SMC details.)
XT.TAK	0,1	Enables (1) or disables (0) the automatic interpretation of input from the keyboard on TMS' optional voice unit.
XT.TSP	0,1	TMS input silence processing state. The default, (1), indicates that the analog level detector is to be used to force TMS input to silence if it drops below the threshold. (Note that this is a global flag that applies to all TMS lines.

Name	Valid Values	Meaning
XT.TTO	0-255	Sets a time limit for a Codec line that is receiving silence. Limit is value multiplied by 100 milliseconds.

End of Table

Notes:

1. TC.FSZ and TC.PAR together determine the number of data bits returned to the task as shown in Table 3-6.

Table 3-6: Frame Size/Parity Values

TC.FSZ	TC.PAR	Number of Data Bits Returned
9	1	8
8	0	8
8	1	7
7	0	7
7	1	6
6	0	6
6	1	5
5	0	5

The following combinations are invalid:

1. TC.FSZ=9 with TC.PAR=0 (Driver sets TC.PAR)

2. TC.FSZ=5 with TC.PAR=1 (Driver clears TC.PAR)

2. The speed selected determines the type of modem to use and overrides the modem type characteristic. For speeds from 110-300 bps, TMS uses the USFSK modem, and for 1200 bps, TMS uses the USDPSK modem.

Table 3-7: Valid Receive/Transmit Speeds (TC.RSP & TC.XSP)

Value for TC.RSP and/or TC.XSP	Actual Receive Speed (Bits-per-second)
S.110	110
S.134	134.5
S.150	150
S.200	200
S.300	300
S.1200	1200

(75 and 600 bps are reserved for future use.)

3. The modem-type (XT.MTP) characteristic specifies whether there is a remote modem in use and, if so, its type. Valid values are listed in Table 3-8.

Table 3-8: Type of Modem Characteristic (XT.MTP)

XT.MTP Value	Modem Type
XTM.FS	USFSK - 0 to 300 baud Bell 103J
XTM.PS	DPSK - 1200 baud Bell 212

4. The dial-mode (XT.DLM) characteristic specifies the type of telephone equipment TMS is to emulate. Dial mode is dictated by telephone system line limitations. The valid dial modes are listed in Table 3-9.

Table 3-9: Dial Modes (XT.DLM)

XT.DLM Value	Equipment Type
XT.DIA	Dial-pulse, 10 pulses per second (Default)
XT.DTM	Dual-tone multifrequency
XT.D20	Dial-pulse, 20 pulses per second
XT.OHS	Off-Hook service (direct-wired or central office controlled)

5. The data-mode (XT.DMD) characteristic specifies the type of data communications enabled on the line and ensures that the required TMS hardware is appropriated for the call. Valid data modes are listed in Table 3-10.

Table 3-10: Data Modes (XT.DMD)

XT.DMD Value	Communications type
XT.DTD	Dual-tone Multifrequency (DTMF) data
XT.ENC	Encoded voice data (Codec) (Line 3 default)
XT.SER	Serial data (modem) (Line 2 default)
XT.VOI	Attended voice telephone (Line 1 default)

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted or message. If the value returned is IE.ABO, the high byte of the first I.O status word contains one of the following three values:

SE.NIH	Characteristic not implemented
SE.VAL	Illegal characteristic value
0	Operation Aborted

3.4.17 Set Multiple Characteristics (SF.SMC)

The SF.SMC function initializes and modifies the operating characteristics of a specified TMS line. Use this function to set and reset characteristics of a communications line, and use the SF.GMC function view current communications line characteristics.

TMS line characteristics are listed in Table 3-11. Originating and receiving line characteristics must be coordinated for intelligible data communications to take place. Refer to Chapter 4 for additional information on TMS data line protocols and characteristics.

The TMS protocol defaults in effect on start-up are listed in the following table. These predefined TMS I/O driver defaults can be overridden by start-up options available through the P/OS system. (Refer to the P/OS system documentation for start-up options.)

Format:

```
QIO[W]$S #SF.SMC,#lun,...,<TMS-specific-parameters>
```

lun	A logical unit assigned to XT1, XT2, or XT3.
...	Optional general QIO parameter(s)

<TMS-specific-parameters>

cbu — Starting address of the characteristics buffer, a variable-length buffer with each characteristic represented by a two-byte field holding the TMS line characteristic-name and characteristic value in the form:

```
.BYTE characteristic-name
.BYTE 0
```

Characteristic-names and their possible values are listed in Table 3-11.

siz — The size (total number of bytes) of the SF.SMC characteristics buffer

Table 3-11: Line Characteristics (SF.SMC)

Name	Valid Values	Meaning
TC.ARC	0-20	Number of rings (1 to 20) to wait before answering an incoming call.A value of 0 disables auto-answering and enables interrupts on ringing.
TC.BIN	0,1	Enable (1) or disable (0) binary mode. With binary mode enabled, XON/XOFF recognition/transmission disabled.
TC.CTS	0,1	Resume (0) or suspend (1) data output (serial mode only)
TC.EPA	0,1	Odd (0) or Even (1) parity sense (ignored unless TC.PAR is enabled (1).
TC.FSZ	5-9	Character frame size (width in data bits, plus parity bit if enabled) See Note 1, Table 3-12 for values.
TC.PAR	0,1	Enable (1) or disable (0) parity checking and generation. See Note 1, Table 3-12 for values.
TC.RSP	(Table 3-13)	Set speed (baud rate) for receiving incoming data in bits-per-second. Table 3-13 in Note 2 lists valid TC.RSP and TC.XSP values.
TC.STB	1,2	Number of stop bits at the end of each character.
TC.TBF	0-255	For SF.SMC flushes all characters in the line's type-ahead buffer, despite the value specified.
TC.TRN	(Note 3)	For SF.SMC, specifies translations for dialing or sets up standard sequences to be sent before and after telephone numbers. Refer to Note 3.
TC.XMM	0,1	Hold control (low-order bit only).

Name	Valid Values	Meaning
TC.XSP	(Table 3-13)	Set speed (baud rate) for transmitting outgoing data in bits-per-second. Table 3-13 (Note 2) lists valid TC.RSP and TC.XSP characteristic values.
TC.8BC	0,1	Pass 7-bit (0) or 8-bit (1) characters on input. (A TC.BIN value of 1 and/or the use of the IO.RAL (read-all) function overrides the TC.8BC value.)
XT.MTP	(Table 3-14)	Specifies one of the modem types listed in Note 4, Table 3-14.
XT.DLM	(Table 3-15)	Specifies one of the valid dial modes listed in Note 5., Table 3-15.
XT.DMD	(Table 3-16)	Specifies one of the valid data modes listed in Note 6., Table 3-16.
XT.DIT	1,255	Sets the length of the pause between dialed DTMF signals. (TMS multiplies value by 10 milliseconds. See XT.DTT.)
XT.DTT	1,255	Sets the length of time to apply a dialed DTMF signal. (TMS multiplies value by 10 milliseconds. See XT.DIT.)
XT.GOV	(any value)	Temporarily enables a 30-second wait without carrier for a telephone handset off-hook status or a change in data mode. (IO.HNG function cancels the wait and disconnects.)

XT.SDE	(ASCII)	Sets the DTMF escape-sequence for password recognition. The format is:
		`.BYTE XT.SDE,string_char_count`
		`.ASCII /esc_seq_string/`
		`.EVEN`
		Next SF.SMC characteristic must begin on a word boundary. (XT.SDE invokes an error when used with SF.GMC.)

Name	Valid Values	Meaning
XT.TAK	0,1	Enables (1) or disables (0) automatic interpretation of input from keyboard on TMS' optional voice unit.
XT.TSP	0,1	TMS input silence processing. Setting this on any line affects all lines.
XT.TTO	0-255	Sets a time limit for a Codec line that receiving silence. Limit is value multiplied by 100 milliseconds. (Codec only)

End of Table

Notes:

1. TC.FSZ and TC.PAR together determine the number of data bits returned to the task as shown in Table 3-12.

Table 3-12: Frame Size/Parity Values

TC.FSZ	TC.PAR	Number of Data Bits Returned
9	1	8
8	0	8
8	1	7
7	0	7
7	1	6
6	0	6
6	1	5
5	0	5

The following combinations are invalid:

1. TC.FSZ=9 with TC.PAR=0 (Driver sets TC.PAR)

2. TC.FSZ=5 with TC.PAR=1 (Driver clears TC.PAR)

2. The speed selected determines the type of modem to use and overrides the modem-type characteristic when line is off-line. For speeds from 110-300 bps, TMS uses the USFSK modem, and for 1200 bps, TMS uses the USDPSK modem.

Table 3-13: Valid Receive/Transmit Speeds (TC.RSP & TC.XSP)

Value for TC.RSP and/or TC.XSP	Actual Receive Speed (Bits-per-second)
S.110	110
S.134	134.5
S.150	150
S.200	200
S.300	300
S.1200	1200

(75 bps is reserved for future use.)

3. The translate table is made up of three sections: (1) a dial-translation table, (2) a start sequence string, and (3) an end-sequence string. Any (or all) sections can be empty. (TC.TRN is an invalid parameter and returns an error when used with the SF.GMC function.)

The format of SF.SMC's TC.TRN characteristic is:

```
.BYTE XT.TRN,count1,count2,count3
.ASCII /dial_translate_table/
.ASCII /start_sequence/
.ASCII /end_sequence/
.EVEN
```

The dial-translate table is a string of character pairs, input character followed by output character. This translate table is used to convert a phone number, typically to remove format characters, such as parentheses, dashes, commas, and spaces.

If a character in a telephone number matches a character in the input section of the dial-translate table, the character is converted to the character from the output section. If the character from the output section is 0, the character from the telephone number is ignored. The TMS I/O Driver provides a default translation table that removes these characters: (,), -, and space.

The start-sequence ASCII string, if specified, is dialed before the phone number. If an end-sequence string is specified, it is dialed following the phone number. Note that the SF.SMC characteristic following TC.TRN data must begin on a word boundary.

4. The modem-type (XT.MTP) characteristic specifies the type of remote modem in use. Valid values are listed in Table 3-14.

If the specified line is on-line when this function executes, modem types changes are ignored. If the speed is invalid for the current modem, an active USFSK modem switches to 300 bps, and an active DPSK modem switches to 1200 bps.

If the line is off-line, speed has precedence over modem type, see note 2. When modem type is DPSK and TC.FSZ plus TC.STB do not equal 9: parity goes off, TC.STB is 1, and TC.FSZ is 8.

If speed is not 75 (reserved for future use) and TC.FSZ plus TC.STB does not equal 9 or 10: the parity goes off, TC.STB and TC.FSZ are set to 1 and 8, respectively.

Table 3-14: Type of Modem Characteristic (XT.MTP)

XT.MTP Value	Modem Type
XTM.FS	USFSK - 0 to 300 baud Bell 103J
XTM.PS	DPSK - 1200 baud Bell 212

5. The dial-mode (XT.DLM) characteristic specifies the type of telephone equipment TMS is to emulate. Dial mode is dictated by telephone system line limitations. The valid dial modes are listed in Table 3-15.

Table 3-15: Dial Modes (XT.DLM)

XT.DLM Value	Equipment Type
XT.DIA	Dial-pulse, 10 pulses per second (Default)
XT.DTM	Dial-tone multifrequency
XT.D20	Dial-pulse, 20 pulses per second
XT.OHS	Off-Hook service (direct-wired)

6. The data-mode (XT.DMD) characteristic specifies the type of data communications enabled on the line so corresponding TMS hardware is appropriated for the call. Valid data modes are listed in Table 3-16.

Table 3-16: Data Modes (XT.DMD)

XT.DMD Value	Communications type
XT.DTD	Dual-tone Multifrequency (DTMF) data
XT.ENC	Encoded voice data (Codec) (Line 3 default)
XT.SER	Serial data (modem) (Line 2 default)
XT.VOI	Attended voice telephone (Line 1 default)

Status:

IS.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted or message. If the value returned is IE.ABO, the high byte of the first I/O status word contains one of the following three values:

SE.NIH	Characteristic not implemented
SE.VAL	Illegal characteristic value
0	Operation Aborted

End of Chapter

CHAPTER 4

COMMUNICATIONS CHARACTERISTICS

Communications take place in one of the four TMS data modes. TMS can operate in DTMF, serial, voice, or Codec data mode. Because each mode has its own operating characteristics, TMS permits modification of these characteristics on a per-line basis. The TMS I/O driver can specify any TMS data mode for a line, and, by modifying the line's characteristics, can implement that mode of communication for both transmitting and receiving.

For example, line 1 could originate a call for someone speaking on the telephone handset while line 2 operates in serial data mode to send a file to another computer and line 3 (the voice unit's Codec line) accepts dictation from the voice unit. Alternatively, line 2 could receive a call while in DTMF mode in order to decode an incoming password; once the password checks, the line could switch to serial data mode and accept incoming serial data from another computer.

To accommodate the flexibility required by three lines assuming any one of the TMS communications modes, the TMS I/O driver provides two functions that report/modify line characteristics. These QIO functions are (1) the get-multiple-characteristics function (SF.GMC) used for reading the Professional's current line characterics, and (2) set-multiple-characteristics function (SF.SMC) for modifying the current features for a line; for details, refer to these function descriptions in Chapter 3.

Table 4-1 summarizes the modifiable TMS characteristics. Some line characteristics apply to all four data modes, while others are specific to a single data mode. Applicable modes are indicated in the table by initial (D -- DTMF mode, S -- Serial mode, V -- voice mode, C -- Codec mode). The data modes and their characteristics are described in this section.

4.1 GENERAL TMS COMMUNICATIONS CHARACTERISTICS

The following section describes characteristics applicable to all four TMS communications (data) modes. Mode-specific characteristics not applicable to all four modes are described in the sections on serial, voice, DTMF, and Codec characteristics.

Table 4-1: Modifiable Characteristics

Characteristic	Mode*	Function
Data Mode	DSVC	Sets TMS' data mode (DTMF, serial, voice, and Codec).
Dial Mode	DSVC	Sets TMS' call-dialing mode (DTMF, dial-pulse/10 pps, dial-pulse/20pps, and off-hook service.
Auto-Answer Ring Count	DSVC	Sets number of rings (1 to 20) to let ring before answering incoming calls. (0 means do not answer.)
Translation	DSVC	Sets up translation table(s) for dialing.
Voice Unit Keypad	DSVC	Enables/disables special firmware interpretation of TMS' optional voice unit keypad.
DTMF Esc.-Seq.	DSVC	Specifies DTMF escape-sequence signal.
DTMF Signal Length	DSVC	Sets the length of time to apply a dialed DTMF signal.
DTMF Pause Length	DSVC	Sets the length of the pause between dialed DTMF signals.
Hold Control	DSVC	Selects hold control and maintenance mode (the latter for Digital use only.)
Type-ahead Buffer	DS	Evaluate status of the I/O Driver's input type-ahead buffer for a line.
Transmit Speed	S	Set speed (baud rate) for transmitting outgoing data (bits/sec: 110, 134.5, 150, 200, 300, and 1200).

* D -- DTMF, S -- Serial, V -- Voice, C -- Codec

GENERAL TMS COMMUNICATIONS CHARACTERISTICS

Characteristic	Mode*	Function
Receive Speed	S	Set speed (baud rate) for receiving incoming data (bits-per-second -- 110, 134.5, 150, 200, 300, and 1200). (75 and 600 bps - future implementation.)
Modem types	S	Specifies one of the permitted modem (USFSK and DPSK, CCITTV.21 and CCITTV.23/Mode 1 and Mode 2 reserved.)
Parity Generation and Checking	S	Enable (1) or disable (0) parity checking and generation.
Parity	S	Select ODD or EVEN parity sense.
Character Frame Size	S	Define width of exchanged characters. Includes data bits and parity bit, if enabled.
Stop Bits	S	Define the number of stop bits following each character.
Eight-bit Characters	S	Specify 7 or 8-bit input characters.
Binary Characters	S	Enable or disable XON/XOFF.
Control Transmission	S	Resume or suspend block data output.
Prepare to go Voice Disconnect	S	Enable or disable a one-time 30-second wait after carrier-loss for changing to voice communications.
Silence-Detection Time-out	C	Sets a time limit for termination of READ-QIO's on a silent line.
Codec Silence Processing	C	Determines input silence processing.

* D -- DTMF, S -- Serial, V -- Voice, C -- Codec

End of Table

4.1.1 Data Mode

The TMS data mode characteristic determines the type of communications to take place on the specified line. Only lines 1 and 2 can vary their mode of communications; line 3 operates exclusively in Codec mode. The data mode defines the resources to be allocated to a line when calling out or answering a call. The four TMS communications data modes are:

- Dual-Tone Multifrequency (DTMF)
- Encoded voice (Codec)
- Serial
- Voice (Attended)

For example, with line 1 set to serial mode, execution of an Answer or Originate function automatically attaches the TMS on-board modem and proceeds as a serial call. If the modem is not available, the call does not proceed, and TMS reports a device-not-ready status to the I/O driver.

If the specified line is on-line (in-use) during the execution of a Set Multiple Characteristics function that calls for a change in data mode, TMS terminates the current data mode, releasing that line's resources for other lines. The state of the specified line changes to off-line, but the telephone connection does not terminate. A subsequent Answer or Originate function takes place on that line in the new data mode.

To allow for a user at the distant end of the connection to change from serial mode to voice mode, you can enable the prepare-to-go-voice characteristic, establishing a 30-second waiting period. The waiting period maintains the connection during the temporary loss of carrier while switching from serial to voice mode.

4.1.2 Dial Mode

The dial mode characteristic defines the type of signalling for TMS to use when placing outgoing calls. Dial mode is dictated by the local telephone exchange.

Once you specify a specific dial mode for a TMS line, all outgoing calls are signalled in that dial mode. The outgoing dialing signals that TMS can generate are:

- Dial pulse, 10 pulses/sec
- Dial pulse, 20 pulses/sec
- Dual-tone multifrequency
- Off-hook service

Dial pulsing is performed by interrupting the current through the telephone in a precisely timed way at a nominal 10 pulses per second. Dual-tone multifrequency dialing uses tone generators for faster signalling with two sinusoidal signals, one from each of the DTMF signalling frequency groups. Off-hook service dial mode is used for a connection on which dialing is not necessary.

TMS can perform DTMF signalling at the rate specified by the SF.SMC function's XT.DIT and XT.DTT characteristics. (Refer to the DTMF signal and pause-length characteristics for signalling information.)

4.1.3 Auto-Answer Ring Count

The automatic-answer ring count characteristic specifies the number of rings to be counted before TMS answers an incoming call on line 1 or line 2. The ring count can be set in a range of one to 20 rings. If you set the count at 0, TMS disables auto-answering, allowing the user to receive incoming ring notification, which might be ignored or used to trigger an explicit IO.ANS.

TMS detects incoming rings on the line and waits for the total number of rings to equal the value specified before going on, usually to an ANSWER function. For example, with a ring count of 2 specified for line 2, the on-board ring-detector would distinguish two distinct rings before answering an incoming call. TMS resets the count to zero if 20 seconds elapse with no new ringing.

4.1.4 Translation

The translation characteristic identifies one or more specific translation table(s) for non-standard dialing from TMS. A translation table converts telephone number data, changing it to the table's corresponding data value. The TMS I/O driver provides a default translation table that removes (,), -, and space characters.

For example, a translation table could convert letters to the corresponding number to allow a telephone number to be specified as (800) DIGITAL. A translation table also could remove parentheses and/or other characters which are not dialed.

Additionally, optional strings can be specified before and/or after the telephone number. For example, some telephone systems require that accounting information be specified with the number. For details on setting up translation tables, refer to the SET CHARACTERISTICS description in Chapter 3.

4.1.5 Voice Unit Keypad

The voice unit keypad characteristic enables or disables interpretation of signals from the buttons on the optional voice unit. When interpretation is enabled, the TMS on-board firmware interprets certain keys allowing the voice unit operate as a completely independent telephone. As an independent telephone, the voice unit uses Line 1 for outgoing calls, much like the optional telephone handset attached to the same line. An application still can be notified when buttons are depressed to do call accounting or to take special action on specific buttons.

With signal interpretation disabled, an application program can intercept signals from the keypad and can direct the call from TMS Line 1 to Line 2 or can perform non-telephone-related actions, such as storing voice or retrieving previously stored voice. The reaction to keypad input is the responsibility of the application program; the TMS I/O Driver only passes the signals on to the application program when enabled to do so.

4.1.6 DTMF Escape-Sequence

The DTMF escape-sequence characteristic specifies a string of DTMF signals to check when answering an call on line 1 or line 2. Escape-sequence checking takes place in voice, DTMF, and Codec modes. You specify the escape-sequence as a string of ASCII values that represent the DTMF signals to check. TMS can check a sequence of up to eight characters. Escape-sequence checking must start during the first 15 seconds following answering. To disable checking, use this characteristic with a null character string in place of an escape-sequence. Refer to the connect (IO.CON) function description in Chapter 3 for details on DTMF/ASCII signal representation.

Typically, the escape-sequence contains a password that must be matched by incoming DTMF signals so that an application program can take special action. TMS checks incoming calls for that specific sequence while answering until the sequence checks or times out.

4.1.7 DTMF Signal Length

The DTMF signal-length characteristic sets the length of time that TMS applies a DTMF signal tone when dialing an outgoing call. This characteristic works with the DTMF pause-length characteristic to set timing values for DTMF signalling.

The value of the DTMF signal length is an integer in the range 1 to 255. TMS multiplies this value by 10 milliseconds to derive the actual amount of time the hardware applies the signal. For example, a signal length of 7 results in a 70 millisecond signal for each DTMF digit dialed.

4.1.8 DTMF Pause Length

The DTMF pause-length characteristic specifies the length of time that TMS pauses between two dialed DTMF signals. The interdigital pause and the DTMF signal-length characteristic provide DTMF signal timing for TMS.

The value of the DTMF pause-length is an integer in the range 1 to 255. TMS multiplies the value by 10 milliseconds to derive the actual amount of time the hardware pauses between signals. For example, a pause value of 5 sets a 50-millisecond wait between the end of one DTMF signal and the beginning of the next digit.

4.1.9 Hold Control

The value of the hold control characteristic indicates whether to operate the HOLD relay provided by TMS control leads A and A1 (yellow and black wires).

The hold control characteristic consists of the low-order bit of the eight-bit field returned or set by TC.XMM in the SF.SMC or SF.GMC function. The seven high-order bits of this field are reserved for maintenance use by Digital.

When the bit is set to 1, the HOLD relay will be operated. When this bit is set to zero, the HOLD relay will not be operated.

NOTE

> The HOLD relay should only be operated in a system which supports HOLD functions controlled by the telephone circuits A and A1. Activating this function in a telephone system that does not support HOLD functions in this manner can lead to improper operation of the telephone system. This can result in service calls for which your telephone supplier may assess a service charge. Improper operation includes the creation of a short circuit across the yellow and black wires of the telephone system, which, if not used for HOLD control, may be connected to unknown circuits by your telephone supplier. (For example, to additional telephone lines or to power supplies for lights or bells.) Shorting this circuit when it is not used for HOLD can cause damage to your telephone system and to other equipment connected to it.
>
> If you have the PRO/Communications application, you should allow the user to set the hold control characteristic (bit) via the setup menu provided with that application. Set the bit with SF.SMC only if PRO/Communications is unavailable.

Before changing the value of the hold control bit, you must save the states of the seven reserved bits. Use the SF.GMC function to save the current states. Include these saved states when you subsequently invoke the SF.SMC function to change the hold characteristic.

4.2 SERIAL COMMUNICATIONS CHARACTERISTICS

TMS handles serial data mode communications over either of its telephone lines. Serial communications are possible with one of Digital's host computers, another Professional, or with any other device capable of serial communications. Application software can initiate outgoing serial data calls and answer incoming serial calls. The following subsections describe TMS serial characteristics.

Modifiable serial characteristics allow the Professional to match various communications protocols. A protocol is a set of rules or characteristics required by another computer before information can be exchanged. The general characteristics described in the preceding section also apply to serial communications.

4.2.1 Typical Application: Terminal Emulator vs. Work Station

Once in communication with another system, the TMS I/O driver allows the Professional to operate in two different communication methods: as an interactive terminal or as an independent intelligent work station. As an interactive terminal, TMS can emulate any large computer's terminal. As a terminal emulator, the Professional works like any CRT attached to and under control of a host computer, using the host's time, memory, and storage resources to work with programs and data files.

As an independent work station, TMS accesses a host computer like a terminal and then "downloads" host information (copies host program and data files to the Professional's storage media). With the information copied, you can operate independently to run host programs and to modify data files as a stand-alone computer without using any of the resources of the host computer. (The two systems can maintain an idle serial link or can terminate all connection at this point.)

To return modified files to the host, TMS can re-establish connection with the host and then "upload" the files serially (copy its own files of modified information onto the host computer's storage media). The way these capabilities are used depends on the application software that is directing TMS.

4.2.2 Modifiable Serial Characteristics

TMS can modify the communications characteristics of the Professional to establish connection with another computer. The set of characteristics chosen must match the protocol requirements of the other computer for the two to communicate.

Serial characteristics are flexible to meet the protocol requirements of various host computer systems and interface devices. Essentially, the Professional can be camouflaged to look like a compatible terminal by changing its protocol. TMS serial characteristics are described in the following.

Note that TMS establishes full-duplex communications for transmitting and receiving data. Most host systems communicate in full duplex, an exchange protocol which permits both terminal and host to send data simultaneously along parallel circuits. Application software can emulate full-duplex with local copy, commonly referred to as "half-duplex."

4.2.2.1 Transmit Speed - The transmit-speed characteristic selects the per-second rate of speed at which TMS sends bits along the transmission line. Usually the rate is predetermined by the device used for sending and/or by the receiving port on a host system.

The available transmit-speed bit rates are: 110, 134.5, 150, 200, 300, and 1200. Generally, data transfer over telephone lines is done at bit-rates of 300 or 1200.

Transmit and receive speed must be the same.

4.2.2.2 Receive Speed - The receive-speed characteristic selects the per-second rate of speed at which TMS receives bits from the transmission line.

As with transmit speed, the receive rate is pre-determined by the receiving device and/or by the transmitting port on a host system. The available receive-speed bit rates are: 110, 134.5, 150, 200, 300, and 1200.

Transmit and receive speed must be the same.

4.2.2.3 Modem Type - The modem characteristic specifies which type of modem TMS uses for the line. Currently, TMS supports the following modem types:

- USFSK - 110 to 300-baud Bell 103J
- DPSK - 1200-baud Bell 212A

The USFSK 103-type modem (U.S. Frequency Shift Keying) provides 300-baud, asynchronous, full-duplex operation over two-wire telephone circuits by use of frequency division multiplexing. The two data channels are obtained by operating in separate frequency bands, one for each direction of transmission. The DPSK 212A-type modem offers 1200-baud, asynchronous, full-duplex operation. Refer to the SF.SMC description in Chapter 3 for additional modem details.

4.2.2.4 Parity Generation and Checking - The parity-enable characteristic (TC.PAR) determines whether or not a non-information bit is added to each character transmitted. With parity enabled, TMS adds a parity bit, according to the parity selected by the parity-type characteristic (TC.EPA).

With parity enabled, transmitted data could be checked at the other end for dropped bits. When parity is enabled, include the parity bit in the total number of bits per character, character width defined by the frame-size characteristic (TC.FSZ).

4.2.2.5 Parity Type - The parity-type characteristic (TC.EPA) selects the type of parity required by the host. The two available parity types are: ODD, and EVEN. For either type, parity checking must be enabled with the parity-enable characteristic (TC.PAR). With ODD parity selected, the parity bit is manipulated to ensure that the total count of bits for a character is an odd number; For EVEN parity, the total bit count for each character is made an even number.

4.2.2.6 Character Frame Size (5,6,7,8,9) - The character-width (frame-size) characteristic (TC.FSZ) specifies the number of bits to expect in each character being transmitted. When two computers are communicating, they must both use the same number of data bits per character.

A frame can be as small as five bits in width and up to nine bits in width. When determining character-width with parity enabled, include the parity bit in the frame width.

4.2.2.7 Stop Bits - This characteristic specifies the number of inter-character marking bits at the end of character. Stop bits indicate where one character stops and the next character starts. The values for stop-bits are 1 or 2. Usually, one stop bit is sufficient for telephone communications; however, two stop bits often are used by mechanical printing terminals operating at 110 bps or slower.

4.2.2.8 Eight-Bit Characters - The eight-bit-characters characteristic allows you to choose whether to clear the high-order bit on received characters with a character frame size of 8 or 9.

4.2.2.9 Binary Characters - Automatic XON/XOFF support is available. With XON/XOFF support enabled, data transmission stops any time there is danger of losing information (receiving device buffer almost full, screen scrolling halted, set-up mode selected, etc.) and resumes automatically when the condition changes. Without support (binary-mode-endabled), data could be lost on overflow.

4.2.2.10 Control-S State - The control-S state characteristic (TC.CTS) for transmitting data on a line is set by the TMS driver when XOFF is received from the distant end and reset when XON is received. The user can examine the bit to determine the state, if necessary, or can clear the state to resume output. This instance can arise if a noisy communications line causes loss of an XON. Although not normally necessary, the user can set the state, preventing output until reset or until an XON is received. However, since the SF.SMC does not execute until other outstanding I/O completes, use by an application program could require the IO.KIL function.

4.2.2.11 Prepare to Go Voice - This characteristic holds Line 1 or Line 2 in preparation for suspending serial data mode and allowing for temporary human interaction (generally, a change to voice data mode). With the 30-second waiting period enabled, TMS holds the line for 30 seconds, despite loss of carrier signal.

If carrier is not lost during the first waiting period, the wait times out and has no effect. If carrier is lost during the first waiting period but no remedial action is taken by the application program. Before the end of the second waiting period, the line disconnects.

4.2.2.12 Type-Ahead Buffer - The type-ahead-buffer characteristic refers to the number of unread characters in the buffer provided by the TMS I/O Driver. Setting any value into this characteristic clears the buffer.

The buffer is used as a type-ahead buffer for serial or DTMF characters coming into TMS. TMS maintains separate buffers for each line, permitting simultaneous DTMF and serial operations on separate lines.

4.3 CODEC DATA MODE CHARACTERISTICS

The Codec allows TMS to record and store encoded (digitized) voice messages and to play them back at any time. Both lines 1 and 2 can select Codec data mode in order to record or playback messages; line 3 operates exclusively in Codec mode and provides dictation and transcription facilities using the TMS voice unit.

One minute of spoken voice requires approximately 0.25 megabytes of storage for the digital signals that represent the recorded voice. Pauses in voice conversation on an incoming TMS Codec line do not use up unnecessary storage. Silence is represented on disk by the silence pattern value (hexadecimal AA).

To conserve disk space, TMS' silence compression facility keeps track of multiple occurrences of the Codec silence pattern and represents these occurrences by the AA byte value followed by the total count of the number of consecutive occurrences of the byte. For example, rather than storing six consecutive silent patterns as AAAAAAAAAAAA, TMS always represents such silence as AA 6. On playback, the silence is interpreted and restored as a pause of the same length as the original voice pause.

Data received from or sent to the TMS codec always observes this format, regardless of the setting of the silence processing characteristic. When playing previously recorded data, it is not necessary to know the setting of the silence processing characteristic was when the data was recorded.

Two Codec characteristics save additional storage space:

1. The silence processing characteristic permits conservation of disk space by setting a low-level limit for recording incoming voice signals.

2. The silence-detection timeout permits notification of silence when reading from a line in Codec mode.

These two Codec characteristics are described in the following subsections.

4.3.1 Silence Processing

Since silence is seldom perfect (there is usually some amount of background noise), the Codec rarely uses the silence pattern for more than a few bytes at a time. Little compression is done when the combining algorithm is the only silence compression method. The silence processing characteristic utilizes TMS' silence detection circuit. This hardware component detects the level of sound reaching the codec. When this characteristic is set, TMS ignores actual data from the Codec that is below the detection level (such as background noise).

Below-threshhold sounds are interpreted as silence, are presented as the silence pattern, and are stored by the previously described TMS silence-compression method. Compression permits the combining algorithm to conserve more space when storing data.

Without this characteristic set, all sound entering the Codec is recorded, regardless of level. The combining algorithm is still used, but the presence of background noise reduces the amount of space conserved.

4.3.2 Silence-Detection Timeout

The TMS silence-detection timeout permits you to request notification when silence is detected while reaading from a line in Codec mode. When you specify a non-zero value for this characteristic and TMS detects silence on the line for that length of time, read operations in Codec mode terminate on timeout with the IS.TMO (success with timeout) status. A zero value eliminates any timeout period, and TMS continues reading whether incoming data is silence patterns or non-silent data.

This permits an application to take appropriate action when silence is detected. When a read operation terminates on timeout, subsequent read operations that are already queued begin receiving further data, whether silent or not. The timeout is reset and applies again when additional silence is detected.

The timeout value represents 100-millisecond increments. This characteristic allows you to specify timeouts from 1/10th of a second through 25 1/2 seconds. (The TMS silence-compression characteristic state has no effect on the silence-detection timeout characteristic.)

The timeout period is cumulative; it does not necessarily apply to a single read operation. The timeout period is reset and begins to time again any time non-silence occurs or after each notification of silence. (The TMS silence-detection timeout is controlled by the silence detector and occurs regardless of the setting of the silence processing characteristic.)

For example, when you set a silence-detection timeout period of two seconds (a decimal value of 20), any time the silence detector detects two or more seconds below the threshold, the current read is terminated with a partially filled buffer. Depending on when the timeout occurs, any number of bytes from zero to the specified buffer size might be in the buffer. The number of bytes actually stored is contained in the second word of the QIO status block. The next data byte, whether silent or not, is stored in the buffer specified by the next outstanding read.

End of Chapter

CHAPTER 5

TMS EVENT REPORTING

TMS reacts to changes in the state of the telephone lines and other hardware components by reporting them to the Professional as unsolicited events. TMS' on-board processor reports any unsolicited events by sending interrupt signals along the bus to the Professional. The TMS driver software detects and identifies event information coming from TMS, and the application software determines the response to TMS interrupts.

Applications can (1) ignore all event information, (2) note an event occurrence but not respond to it, or (3) note an event and respond to it by discerning the reason for the status change and dealing with the event appropriately. For example, an application that answers incoming calls needs to respond properly when the telephone rings, while an application program for transmitting encoded voice (Codec) data to another computer could ignore a ringing line.

Responses to unsolicited and unscheduled TMS events depend on the application software controlling TMS. This chapter defines TMS events, describes auxiliary voice unit keyboard input, and provides references to event-handling driver functions described in Chapter 3.

5.1 TMS EVENTS

TMS reports line activity, unscheduled events, unsolicited input, and hardware status changes. These include:

1. On-line/off-line and modem changes (carrier detect/loss, XON/XOFF signals, and DTMF escape sequence input)

2. Detection of incoming rings

3. Unsolicited keyboard/indicator input from the voice unit

4. Line 1 telephone handset changes (on-hook or off-hook)

TMS firmware interrupts each time any hardware event occurs. The interrupt is received by the TMS I/O driver through P/OS and sent to the application program. The AST provides an event-type identifier byte (low byte, top word of stack) that specifies the event causing the interrupt. The line-identifing parameter for the event, as specified by the attach function (IO.ATA), is contained in the high byte of the same word to identify the line sensing the event.

5.2 ENABLING AND DISABLING EVENT-HANDLING

There are five TMS I/O driver functions that allow you to ignore or respond to unsolicited, unscheduled TMS events. The functions that control event-handling are:

IO.ATA	Attach a task to a line and report unsolicited events
IO.ATT	Attached a task to a line without event reporting
IO.DET	Detach a task from a line
IO.LTI	Monitor an unattached line for unsolicited events
IO.UTI	Discontinue monitoring for unsolicited events

Each event-handling driver function is detailed in Chapter 3.

The attaching/detaching functions (IO.ATA, IO.ATT, and IO.DET) can be called from the application task that controls the appropriate line. Event reporting within a task permits feedback on status changes and latest interrupt information while the task commands other TMS operations. For example, a serial-data transfer application might need immediate notification if carrier dropped on the transmit line.

The monitoring functions (IO.LTI and IO.UTI) usually are used by a task that does not control the line but simply keeps track of any unsolicited incoming data or unscheduled hardware changes (a ringing line, a record request from the voice unit, etc.) A monitoring-type application could respond to a ringing line by starting up another program to attach to the specific line and answer the phone. For additional information on event-handling, refer to the P/OS System Reference Manual.

5.3 EVENT DESCRIPTIONS

The following subsections describe the various unsolicited and unscheduled events that TMS reports. Each event report is described as to the hardware changes prompting the interrupt. Auxiliary keypad input from the TMS voice unit also is described in this chapter. Table 5-1 summarizes the reported events.

Event reports often represent more than one type of hardware change so that event reports vary in meaning, depending on the line reporting and the current TMS line states. For example, the carrier-detect event indicates successful modem connection when reporting for a line in serial data mode; on the other hand, a carrier-detect message for a line in voice mode indicates that the voice connection is established.

Table 5-1: TMS Unsolicited Event Types

Event Message	Message Meaning
XTU.CD	Carrier Detect/on-line
XTU.CL	Carrier Loss/disconnect complete
XTU.DR	DTMF Escape Sequence Received
XTU.OF	XOFF Received (serial only/ASCII mode)
XTU.ON	XON Received (serial only/ASCII mode)
XTU.RI	Ring (with auto-answer disabled)
XTU.UI	Unsolicited Input (DTMF & serial only)
XTU.TU	Telephone handset Off-hook (Line 1 only)
XTU.TD	Telephone handset On-Hook (Line 1 only)

5.3.1 Carrier Detect (XTU.CD)

TMS issues the carrier-detect event report when line status switches from off-Line (no connection) to on-line (connection made) to show a call answered or originated successfully. This event is reported following a change in data mode once the line comes on-line in the new data mode. On-line status is reported for all three TMS lines (Line 1, Line 2, and Line 3).

When the line is in serial mode, carrier already is detected before TMS issues the event message. For voice, on-line occurs immediately after answering or dialing. For DTMF, on-line occurs on answer after the tone is sent; on originate, on-line occurs following detection of a tone from the called party. For Codec, on-line occurs as soon as the Codec is ready.

5.3.2 Carrier Loss (XTU.CL)

TMS reports the carrier-loss event whenever line status switches from on-line (connection in effect) to off-line (connection broken). This event shows that a call disconnected successfully. Off-line status is reported for all three TMS lines (Line 1, Line 2, and Line 3).

When received for a TMS line operating in serial data mode, this message indicates that the line no longer has carrier signal. This event is valid for either line 1 or line 2 in serial mode.

TMS issues the carrier-loss interrupt report to notify that carrier signal is lost on the specified line. Note that in serial mode, if the distant end disconnects, TMS reports this message twice; once when carrier is lost, and again when the line actually goes off-line. When the TMS end initiates the disconnect, the message only comes in once.

5.3.3 XOFF Received (XTU.OF)

XON/XOFF support can be set with the set-multiple-characteristics (SF.SMC) function's TC.BIN characteristic value set to 0. When communicating with another machine that transmits XOFF and XON signals, TMS reports receiving the XOFF signal as input on one of the lines and blocks data transmission to wait for the incoming XON signal. The TC.CTS characteristic of the SF.SMC function determines XON/XOFF state. (Refer to Chapter 3.)

5.3.4 XON Received (XTU.ON)

See the preceding XOFF Received (XTU.OF) description.

5.3.5 Ring (XTU.RI)

The ring event notifies that the TMS ring detecter received a ring-in signal for the indicated line (line 1 or line 2).

NOTE

With the auto-answer facility enabled, TMS does not send this message to acknowledge incoming rings.

5.3.6 Unsolicited Input (XTU.UI)

TMS reports the unsolicited input event only when the line is in DTMF or serial data mode. When data arrives and no read request is active, the TMS I/O driver reports unsolicited (XTU.UI) to the task.

To ensure capture of all event reports, the AST becomes "disarmed" until the task issues a read request. Once the read request completes, the AST is armed again for new unsolicited events.

Note that in DTMF data mode, prior to the issuance of the first read, DTMF data is reported as an unsolicited character received. Refer to the section on auxiliary keyboard and DTMF unsolicited input further on in this chapter.

5.3.7 DTMF Escape Sequence Received (XTU.DR)

TMS reports the DTMF escape sequence received message to indicate that the TMS DTMF receiver detected a completed escape sequence. Checking occurs in DTMF, voice, and Codec modes. This event message comes in only when the received characters match the sequence specified by the XT.SDE characteristic specified with the SF.SMC function as described in Chapter 3 of this manual.

In DTMF data mode, escape sequence recognition is disabled after the first read commend is issued. For a chart of the ASCII character codes representing DTMF tones, refer to the IO.CON description in Chapter 3.

5.3.8 Telephone-Handset Off-hook (XTU.TU)

The telephone-handset-off-hook event message reports that the telephone handset on Line 1 changed states to off-hook (picked up). This event applies only to Line 1 and indicates that the telephone handset is off-hook.

Once TMS detects the new status, TMS relinquishes all control of the line, and Line 1 reverts to manual control. TMS aborts the current data mode for that line and goes off-line as soon as the telephone handset is lifted off-hook.

5.3.9 Telephone-Handset On-hook (XTU.TD)

The telephone-handset-on-hook event message announces that the Line 1 telephone handset changed states to on-hook (hung-up). This event refers only to the telephone handset on line 1.

5.3.10 Auxiliary Keyboard and DTMF Unsolicited Input

DTMF unsolicited input arrives only on TMS lines 1 and 2. DTMF unsolicited input can come in any time the line is in Codec mode and in voice or DTMF mode while escape sequence checking is active.

In DTMF mode, the DTMF unsolicited input arrives until the first read command is issued; thereafter, unsolicited DTMF input is reported by the unsolicited input message (XTU.UI).

DTMF signalling returns the values described for the connect function. Refer to the IO.CON description in Chapter 3 of this manual.

TMS can receive data from the keypad of the the optional TMS voice unit. The input received depends on the keys depressed on the keypad. This input only appears on TMS Line 3. There is a unique byte value for every possible voice unit action or keystroke.

TMS indicates the key depressed with a negative value. One of the keyboard values shown in Table 5-2 is sent as a negative value in the range -13 to -134. (The negated value can be found in the low byte of the top word of the stack.)

Table 5-2: Voice Unit Key Signals

Decimal	Octal	Meaning
48	060	0 key of numeric telephone keypad
49	061	1 key of numeric telephone keypad
50	062	2 key of numeric telephone keypad
51	063	3 key of numeric telephone keypad
52	064	4 key of numeric telephone keypad
53	065	5 key of numeric telephone keypad
54	066	6 key of numeric telephone keypad
55	067	7 key of numeric telephone keypad
56	070	8 key of numeric telephone keypad
57	071	9 key of numeric telephone keypad
58	072	* key of numeric telephone keypad
59	073	# key of numeric telephone keypad
Press/Release		
6/18	6/22	Record pushbutton
1/13	1/15	Play pushbutton
7/19	7/23	Fast Forward pushbutton
2/14	2/16	Rewind pushbutton
8/20	10/24	Insert pushbutton
3/15	3/17	Comment pushbutton
9/21	11/25	Pause pushbutton
5/17	5/21	ON/OFF pushbutton
4/16	4/20	Mike pushbutton (Muted/No longer Muted)

End of Chapter

CHAPTER 6

TMS HARDWARE COMPONENTS

The TMS hardware components enable a Professional computer to communicate through TMS with people and with other computers. This section provides a functional description of TMS hardware and the data communications interfaces between TMS, external telephone lines, and the Professional computer.

The TMS hardware system has three major separate components: (1) the TMS Controller module, a printed-circuit board within the Professional serving as a dedicated slave computer system for handling data communications between the Professional and TMS users; (2) the Telephone Line Interface (TLI) module connecting the Professional to external telephone lines, an optional telephone handset and the optional TMS Voice Unit; and (3) the TMS Voice Unit, a desktop telephone speaker unit with a pushbutton pad for dialing calls or controlling applications and a set of TMS pushbutton/indicators that report on and control the voice unit's speaker, recording, and playback functions.

6.1 CONTROLLER MODULE COMPONENTS

The TMS controller module has an on-board 8031 micro-processor and a dual-ported memory bus interface to the Professional's own processor. On-board random-access memory (RAM) serves as dedicated buffer space for exchanging data, commands, and TMS status reports with the Professional.

The TMS micro-processor controls a variety of communications subsystems by means of parallel and serial I/O devices located on the module's on-board bus and processor I/O ports. Analog signals are routed among the subsystems and the external telephone lines through a crossbar array controlled from the TMS bus. Figure 6-1 is a block diagram of the TMS controller module, its bus, and interface paths to the Professional and the TMS TLI.

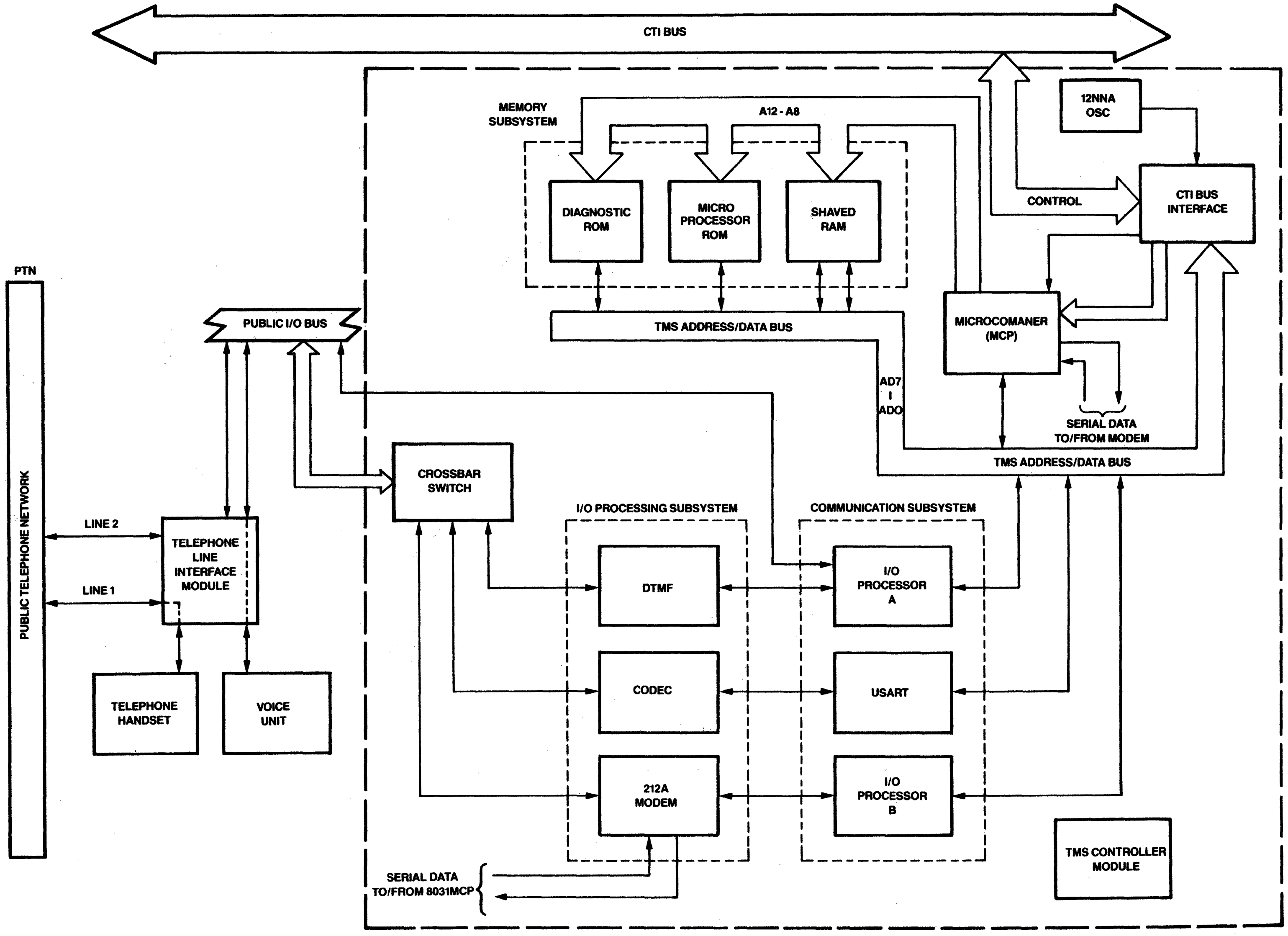

Figure 6-1: TMS System Block Diagram

The four-layer controller board has a 2K-byte random-access memory (RAM) for data and command buffering, timers, and both serial and parallel ports. Parallel ports exist for the telephone line interface (TLI), DTMF-tone transmitter/receiver, the Codec, the on-board modem(s), and the signal switching crossbar. TMS' own 5-source, 2-priority-level nested interrupt structure controls the on-board micro-processor and permits rapid servicing of external events and real-time-driven peripherals. Each of these components is described in the following subsections.

6.1.1 TMS Micro-Processor

TMS' 8-bit on-board processor manages telephone communications resources on the controller module and exchanges data, commands, and status information with the Professional's own processor. The TMS processor has an external 8K-byte read-only memory used for controller system and test procedure firmware.

The TMS micro-processor has two different memory cycle types, program memory cycles of 6 clock cycles (500 nanoseconds) and data memory cycles of 12 clock cycles (1 usec). The firmware ROM is accessed with program memory cycles providing 2-byte fetches in a 12-clock-cycle period. TMS bus buffer RAM and I/O devices are accessed with data memory cycles with single-byte fetches each 12-clock-cycle period. The 8031 is a control-oriented CPU designed for use in sophisticated real-time applications.

The 8031's universal asynchronous receiver/transmitter (UART) is assigned to the TMS on-board modems. The UART operates in standard asynchronous mode with a start bit, eight data bits, and a stop bit. The TMS controller firmware implements a full-duplex 110-baud UART to communicate with the TMS voice unit. FSK modem support is provided in firmware as a full-duplex 75-baud UART.

6.1.2 Random-Access Memory

On-board random-access memory (RAM) serves as dedicated buffer space for exchanging data, commands, and TMS status reports with the Professional. The 2K-byte static RAM appears from the Professional's bus as 64 bytes of memory, mapped as the lower byte in 64 words accessible. The first location is reserved for the TMS page pointer, which selects the RAM page to be accessed through the remaining RAM locations. The first two locations are pre-empted by diagnostics ROM and the TMS address register.

For the TMS on-board processor, RAM appears as 32 discontinuous 62-byte pages. Any the 32 RAM buffers can be selected by loading the page pointer with the appropriate octal address (000 to 037).

NOTE

> Application software access to RAM locations is restricted to the TMS I/O driver interface.

6.1.3 Read-Only Memory

Two on-board 8K-byte components provide the TMS processor with read-only memory for controller system and test firmware. (The 8031 has no internal ROM.) The firmware ROM contains the TMS firmware command set, a TMS-specific instruction set for communication between the Professional and the TMS I/O devices.

Communication with the TMS firmware consists of (1) commands from the TMS I/O driver controlling outgoing serial data, TMS hardware activity, and telephoning functions, and (2) messages from TMS to the driver reporting on line activity, unscheduled events, incoming data, and hardware status changes. The TMS firmware command set is accessible only to the TMS I/O driver; applications cannot access to the TMS firmware instruction set.

The 8K diagnostic ROM contains TMS self-test firmware. This firmware activates every time the Professional executes its power-up self-testing procedures. The power-up self-test analyzes the results of the TMS firmware power-up self-test during module initialization.

The diagnostic firmware checks the RAM, DTMF local loopback, DTMF-Codec local loopback, Codec-tone detector local loopback, modem local analog loopback, and voice unit loopbacks. Voice unit self-test interactively verifies operation of the LEDs, keypad, speaker, and microphone. The Professional handles any errors reported during TMS power-up self-test. System test firmware is available for testing through TMS diagnostic programs, which utilize the TMS I/O driver.

6.1.4 TMS Bus

To manage all hardware resources, the TMS controller has its own bus. This bus controls eight data lines buffered from the 8031 processor's parallel port 0 by a bi-directional bus buffer.

Eight bits of parallel I/O are assigned to controlling the telephone line interface (TLI). Thirteen bits are assigned to the DTMF receiver/transmitter. Twelve bits of parallel I/O are assigned to each modem subsystem. The analog signal switching crossbar array also is controlled from the TMS bus.

The TMS controller module's universal synchronous/asynchronous (USART) and its three bits of parallel I/O are assigned to the CVSD voice coder/decoder (CODEC). The USART operates synchronously at 32K bits per second. Its function is to accept parallel data for output to the Codec as continuous serial data and input of incoming serial Codec data for conversion to parallel format for TMS.

At power-up, the TMS micro-processor initializes the bus I/O devices as follows: sets both the 212A and FSK modems to Bell 103 mode (inactive); turns off the DTMF transmitter and sets the TLI relays to on-hook (hung up) and voice unit microphone unmuted; initializes the USART to synchronous mode with one-byte sync character, external sync detection, and a single sync character for silence-pattern.

6.1.5 Crossbar Array

The TMS controller module drives its associated devices using an 8-input/8-output signal crossbar. The signal switching system is implemented with two CMOS 8 x 4 crossbars. The crossbars provide signaling paths to the following TMS components: three external communications lines (the two telephone lines and the voice unit); the modem subsystem(s); the Codec subsystem; the DTMF subsystem; and the call-progress tone detector.

The TMS crossbar array is controlled by the on-board processor, which makes connections as needed to perform TMS operations.

When the Professional powers up, all crossbar switches are reset to open. Inputs are connected to outputs by writing a 1 to the appropriate bit in the I/O RAM page. Writing a 0 disconnects the inputs from the outputs. The crossbar array permits software assignment of any subsystem to any line and permits connecting systems to themselfves or to each other for testing and diagnostic purposes.

6.1.6 DTMF-Tone Transmitter/Receiver

The TMS on-board DTMF transmitter/receiver is used for data communications and for recognizing DTMF input. DTMF signaling is approximately eleven times faster than rotary pulse-dialing within those telephone systems equipped for DTMF transmissions. (TMS provides both types of dialing.)

The DTMF subsystem handles receiving and sending for dual-tone multifrequency signaling, a pair of high/low frequency sinusoidal signals used to represent the standard 16-character tone set (often referred to as "touch-tone" signaling, a trademark of the American Telephone and Telegraph Company).

6.1.7 Tone Detector

The TMS controller's on-board filter and detector subsystem is an input-only device that detects call-progress tones to assist automatic call placement. The tone detector reports receiving the following tones: dial tone and audible ringback signals. Detected tones are expected at a level above -37 dBm.

6.1.8 Codec

The TMS controller includes a coder/decoder (CODEC) for digitizing voice input and for playing back digitized voice. The device is half-duplex and can either record or play back at one time; it cannot transmit and receive simultaneously. TMS' Codec uses a continuously variable slope delta (CVSD) modulation scheme at a serial data stream rate of 32 kilobits per second.

The Codec subsystem also includes a voice-detect circuit and an automatic level control (ALC) circuit. The voice detector allows detection of silent passages for compacting of periods of recognizable silence in voice input; the ALC adjusts the signal levels to compensate for gain variations in incoming calls.

6.1.9 Modem(s)

The TMS controller is designed for two modems that operate with the two telephone lines. The TMS modems are: (1) the Western Electric high-speed DPSK operation modem compatible with Bell 212A and low-speed FSK operation compatible with Bell 103J, and (2) the international FSK modem, an AM7910 FSK unit supporting communication under five Bell and CCITT protocols, including V.21 and V.23.

Data transmission operates asynchronously at rates ranging from 75 bps to 1200 bps using full-duplex modems. (Application software can effect full-duplex with local copy.) TMS also can operate at data rates of 75 bps to 1200 bps using the half-duplex Bell 202 modem contained on the 7910 chip; however, this modem is no longer in common use, and no software support is planned.

A TMS modem is most sensitive to crosstalk from other crossbar devices due to the amount of information extracted from very low signals in the presence of noise. To avoid any problems, the modems are not connected at the crossbar to the Codec output, the voice unit input, or the DTMF transmitter -- all high amplititude signals with components in the modem passbands.

6.2 TELEPHONE LINE INTERACE (TLI) COMPONENTS

The TMS telephone line interface (TLI) module is a small interface board that resides at the back of the Professional system unit. The TLI plugs into a 22-pin general I/O connector, and 16 wires on the Professional's data bus connect the TLI connector to the TMS controller module.

The TLI module contains plugs for connecting TMS telephone lines to external telephone lines. TLI connections also provides plug attachments for an optional telephone handset and for an optional TMS voice unit. The TLI's bypass switch permits override of TMS telephone line control for any emergency requiring the use of TMS-dedicated telephones.

6.2.1 TLI/Line Connectors

Figure 6-2 shows the connectors and switches on the TLI's external plug panel, along with the interface hardware to the TMS controller module. The TMS TLI provides two jacks to telephone lines, one jack for a telephone handset, and a DIN connector to the TMS voice unit. Two of the back panel jacks are reserved for Line 1 -- one for the external phone connection and one for the user-provided telephone handset.

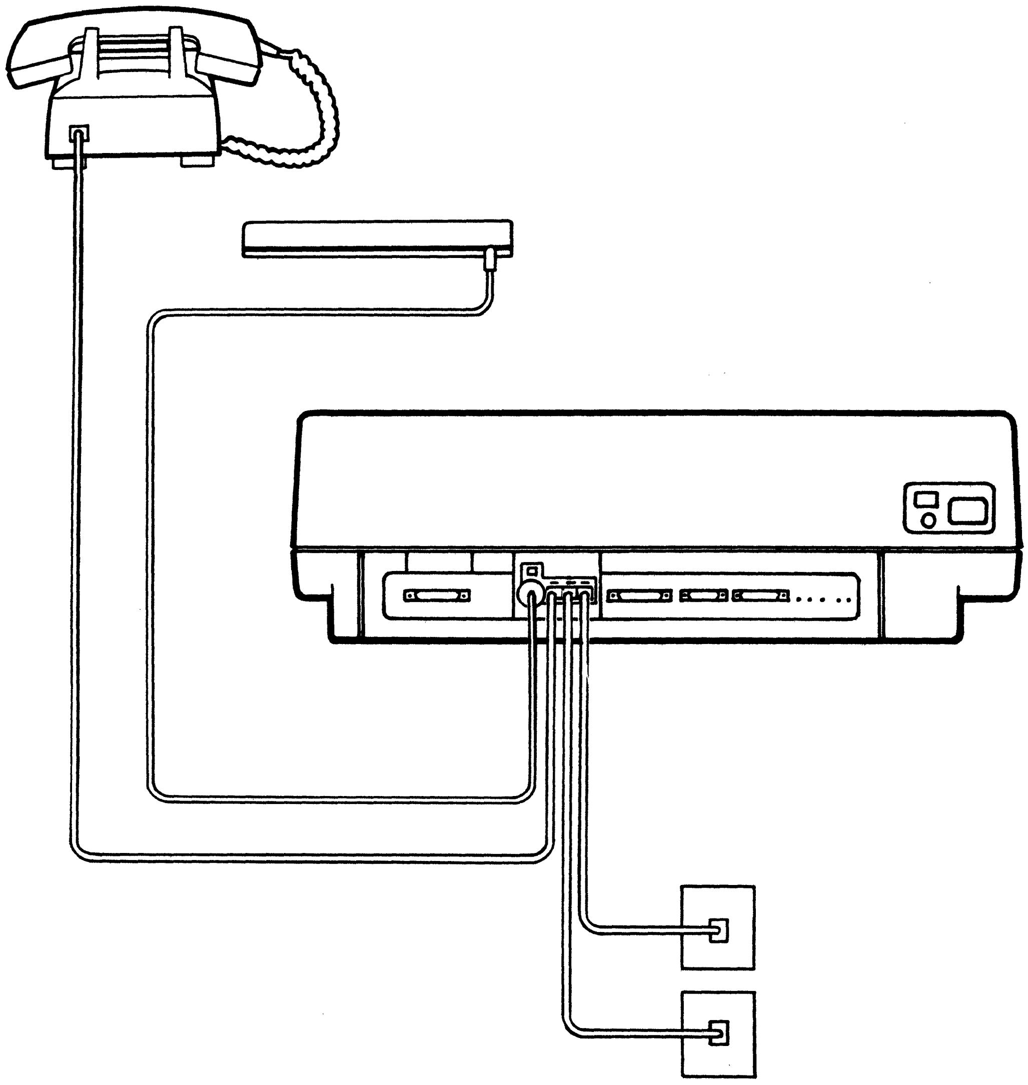

Figure 6-2: TMS Telephone Line Interface (TLI)

6.2.2 TLI/TMS Communications

TMS sends and receives status and data signals along the data bus. Digital signals from the TLI represent status changes on the various external lines. Analog signals are buffered between the bus and the TMS crossbar subsystem. Voice unit serial data also uses the data bus passing information between the voice unit and the TMS controller module's on-board processor.

The TLI contains two optocouplers configured to detect ring-in on the two external telephone lines. Each line also has a hold relay for optional control of the A and A1 leads in multi-button systems. (See the description of the hold control line characteristic -- TC.XMM -- in Chapter 4). A third optocoupler detects offhook condition on the external telephone handset. Each relay has one dedicated line on the Professional data bus. Table 6-1 lists the particular signals that pass between the TLI module and the TMS controller module.

Table 6-1: TLI Signals

Digital Relay Controls and Status Indicators	
	Telephone Line 1 Hook Relay Control
	Telephone Line 2 Hook Relay Control
	Telephone Line 1 Hold Relay Control
	Telephone Line 2 Hold Relay Control
	External Telephone Handset Mute Relay Control
	External Telelphone Handset Offhook Indicator
	Telephone Line 1 Ring-in Indicator
	Telephone Line 2 Ring-in Indicator
Analog Crossbar Signals	
	Transmit to Telephone Line 1
	Receive from Telephone Line 1
	Transmit to Telephone Line 2
	Receive from Telephone Line 2
	Transmit to Voice Unit
	Receive from Voice Unit
Voice Unit Serial Data	
	Transmit Data to Voice Unit
	Receive Data from Voice Unit

6.3 VOICE UNIT COMPONENTS

The TMS voice unit is a speaker/microphone accessory that functions as a

- hands-free speaker telephone,
- dictation recorder/player, and
- pushbutton/indicator panel for application-specific signaling.

The TMS voice unit has its own on-board controller that communicates with the TMS controller module's processor through a pair of 110-baud current-loop serial data lines. Refer to Figure 6-2 for voice unit/TMS controller connections.

6.3.1 Voice Unit Keypad

The voice unit provides a standard numeric telephone keypad for dialing telephone calls and for application-interpreted signaling. A bank of nine LED indicators under control of the TMS I/O driver can be programmed for specific TMS applications.

The voice unit's own microprocessor scans and transmits any keypad input, drives the LEDs as directed by the TMS controller, reports on footswitch status, and controls the automatic level control circuits for the microphone and speaker.

6.3.2 Voice Unit/TMS Communications

Signaling between the voice unit and the TMS controller occurs via the serial cable interfacing through the TLI and from there to the TMS microprocessor on the TMS controller module. The voice unit carries serial signaling information, audio input and output, and power and ground leads. The voice unit contains a connector jack for an optional headset, microphone, or earphone and foot-pedal switch.

The interconnections for the voice unit include the following:

- Transmit voice to speaker
- Receive voice from microphone
- Transmit serial data for LED and switch control
- Receive serial data from keypad and pushbuttons
- Positive Power
- Negative Power
- Reset Voice Unit processor

6.3.3 Voice Unit Speaker/Microphone

The voice unit contains a built-in speaker and a set of circuits to switch and condition voice. The speaker subsystem features automatic level control (ALC), a feedback system for maintaining a modulated voice level. A voice detector lets the voice unit controller sense voice and silence and arbitrate half-duplex operation during a conversation.

The voice unit's built-in microphone and its associated circuits send voice as analog-encoded signals to the Professional. The microphone subsystem includes an external microphone jack, a sensitivity switch, an ALC feedback system to modulate voice level, and a voice detector to sense differences between voice and silence for arbitration of half-duplex operations during conversations.

End of Chapter

APPENDIX A

TMS CALL SUMMARY

Call	TMS Parameters	Purpose
IO.ANS	Databuffer, buffersize	Answer an incoming call
IO.ATA	Astname, [usercodebyte]	Attach a task and report unsolicited events
IO.ATT	(No TMS parameters)	Attach w/o event reporting
IO.BRK	Breaktype	Issue break or long-space
IO.CON	Numbuffer,buffersize, [tmo]	Dial a number and connect
IO.DET	(No TMS parameters)	Detach a task from a line
IO.HLD	(No TMS parameters)	Place a line on hold
IO.HNG	(No TMS parameters)	Hang up a line
IO.KIL	(No TMS parameters)	Cancel pending functions
IO.LTI	Taskbuff, buffsz, [userbyte]	Monitor unattached line for unsolicited events
IO.ORG	Databuffer, buffersize [tmo]	Initiate a call on an existing connection
IO.RAL	Inbuffer, buffersize [tmo]	Read logical data block, pass all bits
IO.RLB	Inbuffer, buffersize [tmo]	Read a logical data block and echo data
IO.RNE	Inbuffer, buffersize [tmo]	Read a logical data block, do not echo data

IO.RVB	Inbuffer, buffersize [tmo]	Read a virtual data block and echo data
IO.UTI	(No TMS parameters)	Stop monitoring line for unsolicited events
IO.WAL	Outbuffer, buffersize [tmo]	Write logical data block and pass all bits
IO.WLB	Outbuffer, buffersize [tmo]	Write logical data block
IO.WSD	Indicatorbuff, buffersize	Write special data to set voice unit controls
IO.WVB	Outbuffer, buffersize [tmo]	Write virtual data block
SF.GMC	Characteristicsbuffer, buffsz	Get line characteristics
SF.SMC	Characteristicsbuffer, buffsz	Set/reset characteristics

End of Appendix

APPENDIX B

SAMPLE TMS PROGRAM

The following Pascal program is a demonstration of the capabilities of the Telephone Management System. The program reads a telephone number from a file called PHONENUM.TXT. Then it prompts the user to press the DO key, causing the program to dial the number it read from the file. The user picks up the phone, presses DO (after being prompted again), hears a tone, and speaks. After a few seconds of silence TMS times-out, or after a period of time TMS beeps the user to finish the message. Then TMS plays back the message for the user to hear.

The program accesses TMS line 1, which should be connected from the TLI unit to a wall telephone jack. To run the program, you must:

- Store the telephone number in PHONENUM.TXT, which is a relative file. (See the file's OPEN statement near the beginning of the program.)
- Create a voice file and store it on disk.
- Install the TMS driver (installing the PRO/Communications application does this for you).

The file PHONENUM.TXT contains three records:

1. the phone number,
2. the voice file name (where the message is stored),
3. a playback line, which is a number whose value determines whether TMS plays back the message on a telset (value=1) or on the voice unit (value=2).

The program opens the voice file by calling the routine Open_Voice_File. Like other routines that the program calls, Open_Voice_File is a MACRO-11 subroutine contained in a file called HANDLER.MAC. We show this file after the main module. Definitions for the MACRO-11 subroutines, included in the main module from the file TMSINC.PAS, also appear later in this appendix.

```
{-----------------------  Pascal Main Module  -----------------------}
Program NccRecord (Input,Output);
    { This program will print a message to the screen for the user and then
      allow the user to record a message and hear it played back.


{            Record.pab

                  Record/fp/cp,Record/-sp=Record/MP
                  CLSTR=PASRES,POSSUM,POSRES,RMSRES:RO
                  Task = Record
                  Stack = 30
                  units = 46
                  asg = ti:10:13
                  gbldef=g$lun:12          ; 10.
                  gbldef=ms$lun:15         ; 13.
                  gbldef=V1$LUN:14         ; 12.
                  gbldef=V2$LUN:13         ; 11.
                  gbldef=tt$lun:5          ; 5.
                  gbldef=L1$Lun:20         ; 16.
                  gbldef=L2$Lun:17         ; 15.
                  gbldef=L3$Lun:16         ; 14.
                  gbldef=tt$efn:1
                  gbldef=xt$ef1:2
                  gbldef=xt$ef2:3
                  gbldef=xt$ef3:4
                  gbldef=xt$ef4:5
                  gbldef=wc$lun:0
                  gbldef=mn$lun:0
                  gbldef=hl$lun:0
                  //

                  Record.odl

                  .root   User -  Handler  - RecrdSpw  - PAS - RMSROT
          USER:   .fctr   Record
          PAS:    .fctr   lb:[1,5]:PASLIB/lb
          @lb:[1,5]:rmsRLX
                  .end


%include 'TMSINC.PAS/nolist'

Type              LineType = Record
                       Len : Integer;
                       Dat : TextType;
                     End;

   { Global Variable Definitions }
var
  Phone_Text      : File of LineType;
  Phonenum        : TextType;
  Phonenumlen     : Integer;
  TableFile       : file of VoiceTableType;
  TableEntry      : VoiceTableType;
  EntryNumber     : integer;
  LongStatus      : LongStatusType;
  Status          : StatusType;
  FileNumber      : integer;
  Name            : TextType;
  NameLength      : Integer;
  RABAddress      : Integer;
  Number          : TextType;
  NumberLength   : integer;
  StartingBlock  : integer;
  NumberOfBlocks : integer;
  Answer         : char;
  RecordLine     : integer;
  Record_Line    : LineType;
  PlaybackLine   : integer;
  PlayBack       : TextType;
  TimeOut        : integer;
  NumGotten      : integer;
  i              : integer;
  Key            : Statustype;
  Beep           : Char;
  RecNum         : Integer;

(************          Procedures and Functions          ************)

[External (Getkey)]
Procedure Getkey (Var Key   : StatusType);
Seqll;

Procedure ExitSt(Status : integer);
  external;

function Length(S:TextType):integer;

  var
    i : integer;
    Done : boolean;

  begin
    i := 80;
    Done := false;
    while not Done do
      begin
        Done := (i < 1);
        if not Done then
          Done := (S[i] <> ' ');
        if not Done then
          i := i - 1
      end;
    Length := i
  end; { of Function Length}

Procedure Phone;

Begin
  Open (Phone_Text,'SysDisk:[ZZSYS]Phonenum.txt',history := old,
       organization := relative,Access_Method := direct access);
  Reset (Phone_Text);
  RecNum := 1;
  Find (Phone_Text,RecNum);
  Record_Line := Phone_Text^;
  Get (Phone_Text);                    { Reading Phone number }
  Phonenum := Record_Line.dat;
  Phonenumlen := Length(Phonenum);
  RecNum := 2;
  Find (Phone_Text,RecNum);
  Record_Line := Phone_Text^;
  Get (Phone_Text);                    { Reading Voice File }
  Name := Record_Line.dat;
  NameLength := Length(Name);
  RecNum := 3;
  Find (Phone_Text,RecNum);
  Record_Line := Phone_Text^;
  Get (Phone_Text);                        { Reading Playback Line number }
  PlayBack := Record_Line.dat;
  End; {of procedure phone}


(*********                Main Program                *********)

Begin { Main Routine }
    Beep := chr(7);                               { Beep keyboard        }

    Initialize_TMS;

  { Call the phone routine that gets the phonenumber etc... }
   Phone;
   Close (Phone_Text);

  {Record and Playback}
  RecordLine := 1;
  PlaybackLine := ord(Playback[1]) - ord('0');
  Hang_Up_Phone(Recordline);    {Making sure phone is hung up before we start}
  Writeln;
  If RecordLine <> PlaybackLine then Hang_Up_Phone(PlaybackLine);

  Writeln (''(27)' #3    Telephone Management System');
  Writeln (''(27)' #4    Telephone Management System');
  Writeln;Writeln;Writeln;
  Writeln ('This is Digital Equipment Corporation',"'",'s Telephone Management')
  Writeln ('System (TMS) demonstration.');
  Writeln;
  Writeln ('The Professional 300 Series computers can be configured with ');
  Writeln ('the TMS option, which uses the phone network for both voice ');
  Writeln ('and data communications.  Its hardware, with appropriate ');
  Writeln ('software, can support: voice storage and forward, dictation, ');
  Writeln ('voice annotation of text, and simultaneous voice/data calls to ');
  Writeln ('allow voice communications concurrent with screen transmission.');
  Writeln;
  Writeln ('TMS was designed with the PROFESSIONAL in mind.');
  Writeln;
  Writeln ('This is your chance to give the new Telephone Management System ');
  Writeln ('(TMS) a try.');
  Writeln;
  Write ('Press the DO key for instructions on how to run this ');
  Writeln ('demonstration. ');

  Repeat
   Getkey(key);
    If (Key[1]<>2) then Begin Write(Beep);Break(Output);end;
    If (Key [1] = 2)and(key[2] <> 16) then begin write(Beep);Break(Output);end;
  Until (Key[1] = 2) and (Key[2] = 16);

  writeln(''(27)'[H'(27)'[J');            { Clear Text Screen }
  Writeln('                 DEMONSTRATION INSTRUCTIONS                    ');
  Writeln;Writeln;
  Writeln ('** NOTE:  PLEASE READ ALL INSTRUCTIONS BEFORE YOU CONTINUE     ');
  Writeln;
  Writeln('In this demonstration you will have the opportunity to talk to a ');
  Writeln('Telephone Management System (TMS).  Your voice will be recorded ');
  Writeln ('on the disk using the telephone next to the machine.  You can ');
  Writeln('then listen to your message as it is played back by TMS. ');
  Writeln;
  Writeln('1)   Press the DO key and the telephone will ring.  ');
Writeln;
Writeln('2) Pick up the phone after it rings.                    ');
Writeln;
Writeln('3)   When ready to record press DO.          ');
Writeln;
Writeln('4)   Start speaking at the sound of the tone.          ');
Writeln;
Writeln('5) TMS will timeout after a few seconds of silence on the line ');
Writeln('   or you will hear another tone to signal that your time is up. ');
Writeln;
Writeln('6) Listen for the playback of your message.          ');
Writeln;
Writeln('7) Hang up the telephone.                            ');
Repeat
 Getkey(key);
 If (Key[1]<>2) then begin Write(Beep);Break(Output);end;
 If(Key [1] = 2)and(key[2] <> 16) then Begin write(Beep);Break(output);end;
Until (Key[1] = 2) and (Key[2] = 16);

FileNumber := 1;
StartingBlock := 1;
Open_Voice_File(LongStatus,Name,NameLength,FileNumber,RABAddress);
    if (LongStatus[1] <> 1) or (LongStatus[3] <> 1) then
      Begin
        writeln(''(27)'[H'(27)'[J');
        Writeln ('Error occured in Opening Voice File');
        Write ('Status = ',LongStatus[1]:5, LongStatus[2]:5);
        Writeln(LongStatus[3]:5,LongStatus[4]:5);
        Writeln;
        Writeln ('Press RESUME to Return to Main Menu');
        Repeat
          Getkey(key);
          If (Key[1]<>2) then Begin Write(beep);Break(output);end;
          If (Key [1] = 2)and(key[2] <> 7) then
              Begin
                write(Beep);
                Break(output);
              End;
        Until (Key[1] = 2) and (Key[2] = 7);
        ExitSt(1);
      end;

TimeOut := 20;                          { 2 seconds }
if TimeOut <> 0 then
  Set_TMS_TimeOut(RecordLine,TimeOut);
repeat
 Set_TMS_To_Codec(RecordLine);
 if PhoneNumLen <> 0 then
   Dial_Phone_Number(RecordLine,PhoneNum,PhoneNumLen)
  else
   Pick_Up_Phone(RecordLine);
 NumberOfBlocks := 10;
 writeln('Press DO to start recording.');

  Repeat
   Getkey(key);
   If (Key[1]<>2) then Begin Write(Beep);Break(output);End;
   If (Key [1] = 2)and(key[2] <> 16) then
     Begin
      write(Beep);
      Break(output);
     End;
  Until (Key[1] = 2) and (Key[2] = 16);
```

```
    TMS_Tone(RecordLine);
    writeln('Recording...');
    TMS_Record (Status,RABAddress,RecordLine,StartingBlock,
                NumberOfBlocks,NumGotten);
    TMS_Tone(RecordLine);
    Writeln;
    writeln('The recording is done...');
    If RecordLine <> PlaybackLine then
      Begin
        Hang_Up_Phone(RecordLine);
        Pick_Up_Phone(PlaybackLine);
      End;
        key[1] := 0; key[2] := 0;
        Writeln;
        Writeln('Playing back your message...');
       TMS_PlayBack(Status,RABAddress,PlaybackLine,StartingBlock,NumGotten);
        Writeln;
        writeln('Press DO to hear the message again or EXIT to quit');
        Repeat
          getkey(key);
          If (key[1] <> 2) then Begin write(beep);Break(output);end;
          If ((key[1]=2) and (key[2]<>16)) and ((key[1]=2)and(key[2]<>10))
            Then
             Begin
              Write(Beep);
              Break(Output);
            End;
        Until ((key[1]=2) and (key[2]=16)) or ((key[1]=2) and (key[2]=10));
        While (key[1] =2) and (key[2] =16) do
          Begin
           Writeln('Playing back your message...');
           TMS_Playback(Status,RABAddress,PlaybackLine,StartingBlock,NumGotten);
           Writeln;
           Writeln('Press DO to hear the message again or EXIT to quit');
           Repeat
              getkey(key);
              If (key[1] <> 2) then Begin write(beep);Break(output);end;
              If ((key[1]=2) and (key[2]<>16)) and ((key[1]=2)and(key[2]<>10))
              Then
               Begin
                Write(Beep);
                Break(Output);
               End;
           Until ((key[1]=2) and (key[2]=16)) or ((key[1]=2) and (key[2]=10));
          End;
    Writeln;
  Hang_Up_Phone(PlaybackLine);
 until (key[1] = 2) and (key[2] = 10);
  Close_Voice_File(Status,FileNumber);
  Writeln;
  Writeln ('Hope you enjoyed talking to TMS; I enjoyed talking with you.');
  Writeln ('I am hanging up now.');
  Writeln;
  Hang_Up_Phone(RecordLine);
  if RecordLine <> PlaybackLine then
    Hang_Up_Phone(PlaybackLine);
 end.
{---------------- end of Pascal main module  -------------------}

{----------------       File TMSINC.PAS      -------------------}

{ TMS Handler Include File.}
{ This file contains definitions of the MACRO-11 routines used by }
{ the Pascal program.                                             }

type
  TextType = packed array [1..80] of char;
  LongStatusType = packed array [1..4] of integer;
  StatusType = packed array [1..2] of integer;

  [EXTERNAL (OPNVOI)]
  procedure Open_Voice_File(var Status: LongStatusType;
                            var Name : TextType;
                            var NameLength : integer;
                                FileNumber : integer;
                            var RABAddress : integer);
    external;

  [EXTERNAL (CREVOI)]
  Procedure Create_Voice_File(var Status: LongStatusType;
                              var Name : TextType;
                              var NameLength : integer;
                                  FileNumber : integer;
                              var RABAddress : integer);
    external;


  [EXTERNAL (CLOVOI)]
  Procedure Close_Voice_File(var Status : StatusType;
                                 FileNumber : integer);
    external;

  [EXTERNAL (TMTALK)]
  Procedure TMS_Playback(var Status     : StatusType;
                             RABAddress : integer;
                             LineNumber : integer;
                             StartBlock : integer;
                             NumOfSegs  : integer);
    external;

  [EXTERNAL (TMLSTN)]
  Procedure TMS_Record(var Status     : StatusType;
                           RABAddress : integer;
                           LineNumber : integer;
                           StartBlock : integer;
                           NumToGet   : integer;
                       var NumGotten  : integer);
    external;

  [EXTERNAL (TMSCDC)]
  Procedure Set_TMS_To_Codec(var LineNumber : [ReadOnly] integer);
    external;



  [EXTERNAL (TMTOUT)]
  Procedure Set_TMS_TimeOut(var LineNumber : [ReadOnly] integer;
                            var TimeOut    : integer);
    external;

  [EXTERNAL (TMSDTM)]
  Procedure Set_TMS_To_DTMF(var LineNumber : [ReadOnly] integer);
    external;

  [EXTERNAL (TMPICK)]
  Procedure Pick_Up_Phone(var LineNumber : [ReadOnly] integer);
    external;

  [EXTERNAL (TMDIAL)]
  Procedure Dial_Phone_Number(var LineNumber : [ReadOnly] integer;
                              var Number : TextType;
                              var NumLen : integer);
    external;

  [EXTERNAL (TMHANG)]
  Procedure Hang_Up_Phone(var LineNumber : [ReadOnly] integer);
    external;

  [EXTERNAL (TMWDTM)]
  Procedure Write_DTMF_Chars( var LineNumber : [ReadOnly] integer;
                              var Str : TextType;
                              var Len : integer);
    external;

  [EXTERNAL (TMRDTM)]
  Procedure Read_DTMF_Chars(var LineNumber : [ReadOnly] integer;
                            var NumOfCharsToGet : integer;
                            var TimeOut : integer;
                            var Str : TextType;
                            var NumOfCharsGotten : integer);
    external;

  [EXTERNAL (TMTONE)]

  Procedure TMS_Tone(var LineNumber : [ReadOnly] integer);
    external;

  [EXTERNAL (TMINIT)]
  Procedure Initialize_TMS;
    external;

  [EXTERNAL (TMVUKD)]
  Procedure Disable_Voice_Unit_Keys;
    external;

{----------------------- End Of TMS Include File --------------------}

; -----------------------  File HANDLER.MAC   --------------------

; This file contains MACRO-11 routines used by the Pascal demonstratior
; program.

        .enable lc,mcl,gbl
        .title  HANDLR - TMS Handlers.
        .ident  /JMS2.0/
;
;       LST$$ = 1                       ; Define RMS Listing Symbol.
        .globl  V1$LUN,V2$LUN,L1$LUN,L2$LUN,L3$LUN
        TTSym$
;
        .sbttl  OPNVOI - Open A Voice File.
;
;       Open Voice File.
;       Calling Sequence:
;
;       [EXTERNAL (OPNVOI)]
;       procedure Open_Voice_File(var Status: packed array [1..4] of integer;
;                                 var Name : StringText;
;                               var NameLength : integer;
;                                   FileNumber : integer;
;                               var RABAddress : integer);
;         external;
;
;       This routine opens an existing file with the name of Name (whose
;       length is NameLength).  It assigns it to the channel specified by
;       FileNumber.  Returned the address of the appropriate RAB (for use
;       by VOREAD and VOWRIT), and a status block. (First two words contain
;       the OPEN Status, second two contain the CONNECT status.)
;
OPNVOI::
        mov     r0,-(sp)        ; Save the regs.
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)
        mov     r4,-(sp)
        mov     r5,-(sp)
;
        mov     20(sp),r1       ; Get the File number.
        dec     r1              ; Subtract one to calculate the offset.
        asl     r1
        mov     r1,r3           ; Remember the offset for a bit.
        mov     RabTbl(r1),r1   ; Get the address of the RAB
        $fetch  r0,FAB,r1       ; Get the address of the FAB
        mov     r1,-(sp)        ; Save RAB for later.
        mov     RMSLun(r3),r1   ; Get the LUN Number.
        $store  r1,LCH,r0       ; Tell RMS what lun to use.
        mov     24(sp),r1       ; Get the address of the string length.
        jsr     pc,CopyNm       ; Copy the name.
        $store  (r1),FNS,r0     ; Save the name length.
        mov     (sp)+,r1        ; Get the RAB back.
        $open   r0              ; Open the file.
        mov     26(sp),r3       ; Get the address of the status block.
        $fetch  (r3),STS,r0     ; get the status.
        $fetch  2(r3),STV,r0
        cmp     (r3), #1        ; Check the status.
        bne     Exit            ; If bad, exit without doing the connect.


;
;       Connect the FAB to a RAB.
;
Conn:   $connect r1             ; Do the connect.
        $fetch  4(r3),STS,r1    ; Get the first status word.
        $fetch  6(r3),STV,r1    ; and the second.
        cmp     4(r3), #1       ; Connect go okay?
        bne     Exit
        mov     16(sp),r4       ; Get the address of where to store the RAB.
        mov     r1,(r4)         ; And store the RAB address there.
;
Exit:   mov     (sp)+,r5        ; restore the silly registers.
        mov     (sp)+,r4
        mov     (sp)+,r3
        mov     (sp)+,r2
        mov     (sp)+,r1
        mov     (sp)+,r0
        mov     (sp),12(sp)     ; Move the return address to the top.
        add     #12,sp          ; Move the stack to the top.
        rts     pc              ; and sure enough, everything is cool.
;

        .sbttl  CREVOI - Create a Voice File.
```

```
;
;       Create a Voice File.
;
;       Calling Procedure:
;
;       [EXTERNAL (CREVOI)]
;       Procedure Create_Voice_File(var Status: packed array [1..4] of integer;
;                                   var Name : StringText;
;                                   var NameLength : integer;
;                                       FileNumber : integer;
;                                   var RABAddress : integer);
;         external;
;
;       This routine opens an existing file with the name of Name (whose
;       length is NameLength).  It assigns it to the channel specified by
;       FileNumber.  Returned the address of the appropriate RAB (for use
;       by VOREAD and VOWRIT), and a status block. (First two words contain
;       the OPEN Status, second two contain the CONNECT status.)
;
CREVOI::
        mov     r0,-(sp)        ; Save the registers.
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)
        mov     r4,-(sp)
        mov     r5,-(sp)
;
        mov     20(sp),r1       ; Get the File number.
        dec     r1              ; Subtract one to calculate the offset.
        asl     r1
        mov     r1,r3           ; Remember the offset for a bit.
        mov     RabTbl(r1),r1   ; Get the address of the RAB
        $fetch  r0,FAB,r1       ; Get the address of the FAB
        mov     r1,-(sp)        ; store away the RAB for now.
        mov     RMSLun(r3),r1   ; Get the LUN Number.
        $store  r1,LCH,r0       ; Tell RMS what lun to use.
        mov     24(sp),r1       ; Get the address of the string length.
        jsr     pc,CopyNm       ; Copy the name.
        $store  (r1),FNS,r0     ; Fill in the file name length.
        mov     (sp)+,r1        ; Get the RAB back.
        $create r0              ; Go and create.
        mov     26(sp),r3       ; Get the address of the status block.
        $fetch  (r3),STS,r0     ; Save the status.
        $fetch  2(r3),STV,r0
        cmp     (r3),#1         ; Did the open go well?
        bne     Exit
        br      Conn            ; if so, go do the connect.
;
;
CopyNm: mov     30(sp),r4       ; Starting address of name.
        mov     (r1),r3         ; Get string address.
        beq     2$              ; Return if it is zero.
        $fetch  r5,FNA,r0       ; Address of Buffer in R5.

1$:     movb    (r4)+,(r5)+     ; Copy the character.
        sob     r3,1$           ; Are we done?

2$:     rts     pc              ; uh, yep...
;

        .sbttl  VOREAD - Read a Voice Block from Disk.
;
;       Read A Voice Block from the disk.
```

```
;
;       Calling procedure:
;
;       mov     #StatusBlock,-(sp)
;       mov     RABAddr,-(sp)
;       mov     #FirstBlock,-(sp)
;       mov     BufferStart,-(sp)
;       mov     #BufferSize,-(sp)
;       jsr     pc,VOREAD
;
;       This routine reads from the stream specified by RABAddr a voice
;       block (4Kb).  It starts at the VBN specified by the contents of
;       FirstBlock.  It places the contents starting at BufferStart, and
;       copies Buffersize bytes to the file.  (It is recommended that
;       BufferSize be equal to 4096.)
;


VOREAD::
        mov     r0,-(sp)          ; Save Registers.
        mov     r1,-(sp)
        mov     r2,-(sp)
;
        mov     16(sp),r0         ; Get address of the RAB.
        $fetch  r1,FAB,r0         ; And get the Fab address.
        mov     14(sp),r1         ; Get the address of the VBN.
        mov     (r1),r1           ; Get the VBN.
        clr     r2                ; First word is zero.
        $store  r1,BKT,r0         ; Tell RMS where to start.
        $store  12(sp),UBF,r0     ; Tell RMS where to put it.
        mov     10(sp),r1         ; Get the address of the buffer size.
        $store  (r1),USZ,r0       ; Tell RMS how much to put.
        $read   r0                ; and go get the stuff.
        mov     20(sp),r2         ; Get the status block.
        $fetch  (r2),STS,r0       ; Find out how we did.
        $fetch  2(r2),STV,r0
;
        mov     (sp)+,r2
        mov     (sp)+,r1
        mov     (sp)+,r0
        mov     (sp)+,10(sp)      ; Restore Pc.
        add     #10,sp
        rts     pc
;
        .sbttl  VOWRIT - Write a Voice Block from Disk.
;
;       Write A Voice Block from the disk.
;
;       Calling procedure:
;
;       mov     #StatusBlock,-(sp)
;       mov     RABAddr,-(sp)
;       mov     #FirstBlock,-(sp)
;       mov     #BufferStart,-(sp)
;       mov     #BufferSize,-(sp)
;       jsr     pc,VOWRIT
;
;       This routine reads from the stream specified by RABAddr a voice
;       block (4Kb).  It starts at the VBN specified by the contents of
;       FirstBlock.  It places the contents starting at BufferStart, and
;       copies Buffersize bytes to the file.  (It is recommended that
;       BufferSize be equal to 4096.)
;
VOWRIT::
        mov     r0,-(sp)          ; Save Registers.
        mov     r1,-(sp)
        mov     r2,-(sp)
;
        mov     16(sp),r0         ; Get address of the RAB.
        mov     14(sp),r1         ; Get the address of the VBN.
        mov     (r1),r1           ; Get the VBN.
        clr     r2                ; First word is zero.
        $store  r1,BKT,r0         ; Tell RMS where to start.
        mov     12(sp),r1         ; Get the address of the input buffer.
        $store  r1,RBF,r0         ; Tell RMS where to put it.
        mov     10(sp),r1         ; Get the address of the buffer size.
        $store  (r1),RSZ,r0       ; Tell RMS how much to put.
        $write  r0                ; and go put the stuff.
        mov     20(sp),r2         ; Get the status block.
        $fetch  (r2),STS,r0       ; Find out how we did.
        $fetch  2(r2),STV,r0
;
        mov     (sp)+,r2
        mov     (sp)+,r1
        mov     (sp)+,r0
        mov     (sp)+,10(sp)      ; Restore Pc.
        add     #10,sp
        rts     pc
;
        .sbttl  CLOVOI - Close A Voice File.
;
;       Close the Voice File.
;
;       Calling Procedure:
;
;       [EXTERNAL (CLOVOI)]
;       Procedure Close_Voice_File(var Status : packed array [1..2] of integer;
;                                      FileNumber : integer);
;
CLOVOI::
        mov     r0,-(sp)          ; Save the regs.
        mov     r1,-(sp)
        mov     r2,-(sp)
;
        mov     10(sp),r1         ; Get the File number.
        dec     r1                ; Subtract one to calculate the offset.
        asl     r1
        mov     r1,r2             ; Remember the offset for a bit.
        mov     RabTbl(r1),r1     ; Get the address of the RAB
        $fetch  r0,FAB,r1         ; Get the address of the FAB
        $close  r0
        mov     12(sp),r2         ; Get the address of the status block.
        $fetch  (r2),STS,r0       ; get the status.
        $fetch  2(r2),STV,r0
;
        mov     (sp)+,r2          ; Restore the silly registers.
        mov     (sp)+,r1
        mov     (sp)+,r0
        mov     (sp)+,2(sp)       ; Move the return address to the top.
        clr     (sp)+
        rts     pc                ; and sure enough, everything is cool.
;
```

```
        .sbttl  TMTALK - TMS Playback Routine.
;
;       TMS Playback Routine
;       Calling Procedure:
;
;       [EXTERNAL (TMTALK)]
;       Proc
dure TMS_Playback(var Status      : packed array [1..2] of integer;
;                                   RABAddress : integer;
;                                   LineNumber : integer;
;                                   StartBlock : integer;
;                                   NumOfSegs  : integer);
;         external;
;
;       This procedure plays back the voice stored in the file that
;       RABAddress refers to.  The voice that gets played back starts
;       at the VBN specified by StartBlock*8.  It is NumOfSegs*8 blocks
;       long.  A RMS Status is returned in the Status buffer.
;
;       The LineNumber parameter determines which LUN the QIO will use.
;       The TMS unit must be set to Encoded Voice Mode (Codec Mode).
;
TMTALK::
        mov     r0,-(sp)                ; Save registers.
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     20(sp),TMStat           ; Store the Status.
        mov     16(sp),TMRAB            ; And the RAB Address.
        mov     14(sp),r0               ; Get the Line Number.
        dec     r0                      ; Calculate what lun to use.
        asl     r0
        mov     TMSLun(r0),Talk1+Q.IOLU ; And tell the QIOs.
        mov     TMSLun(r0),Talk2+Q.IOLU
        mov     12(sp),r0               ; And the Starting Voice Block.
        dec     r0
        mul     #8.,r0                  ; Multiply by eight to get the VBN.
        inc     r1                      ; But, start at VBN 1.
        mov     r1,TMVBN
        mov     10(sp),r0               ; Get the address of the length.
        clr     r1                      ; Start with first set of directives.
        dir$    #SetEF2                 ; Make sure first wait doesn't.

1$:     mov     r1,r2
        asl     r2                      ; Get Buffer Table Offset.
        wtse$s  LockTb(r2)              ; Wait for buffer to free up.
        clef$s  LockTb(r2)              ; And then lock it.
        mov     TMStat,-(sp)            ; Pass the Status.
        mov     TMRAB,-(sp)             ; And the RAB Address.
        mov     #TMVBN,-(sp)            ; And the VBN to get.
        mov     BufTbl(r2),-(sp)        ; And pass the address of the Buffer.
        mov     #TMBFSZ,-(sp)           ; And the size of said buffer.
        call    VoRead                  ; Go fetch the Voice Block.
        dir$    TalkTb(r2)              ; Playback what we found.

        bcc     2$                      ; Did we succeed?
        bpt                             ; No, do something about it later.

2$:     dir$    WaitTb(r2)              ; Wait for last talk to finish.
        add     #8.,TMVBN               ; Add the voiceblock size to the VBN.
        inc     r1                      ; Switch Buffers.
        bic     #177776,r1
        sob     r0,1$                   ; And read the next block if we want.
        mov     (sp)+,r2                ; Restore ourselves.
        mov     (sp)+,r1
        mov     (sp)+,r0
        mov     (sp)+,10(sp)
        add     #10,sp
        rts     pc
;
Bf1Ast: setf$s  Bf1Lck                  ; Clear the Lock on this buffer.
        tst     (sp)+                   ; Get rid of IOSB address.
        astx$s
;
Bf2Ast: setf$s  Bf2Lck                  ; Clear the lock on this buffer.
        tst     (sp)+                   ; Clean up the stack.
        astx$s
;


        .sbttl  TMLSTN - TMS Record Routine.
;
;       TMS Record Routine
;
;       Calling Procedure:
;
;       [EXTERNAL (TMLSTN)]
;       Procedure TMS_Record(var Status       : packed array [1..2] of integer;
;                                RABAddress   : integer;
;                                LineNumber   : integer;
;                                StartBlock   : integer;
;                                LenToRecord  : integer;
;                            var LenRecorded  : integer);
;         external;
;
;       This procedure plays back the voice stored in the file that
;       RABAddress refers to.  The voice that gets played back starts
;       at the VBN specified by StartBlock.  It is NumOfSegs*8 blocks
;       long.  A RMS Status is returned in the Status buffer.
;
;       The TMS unit must be set to Encoded Voice Mode (Codec Mode).
;

TMLSTN::
        mov     r0,-(sp)                ; Save registers.
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)

        mov     24(sp),TMStat           ; Store the Status.
        mov     22(sp),TMRAB            ; And the RAB Address.
        mov     20(sp),r0               ; Get the LineNumber.
        dec     r0                      ; Calculate which lun to use.
        asl     r0
        mov     TMSLun(r0),Lstn1+Q.IOLU ; And load up the directives.
        mov     TMSLun(r0),Lstn2+Q.IOLU
        mov     TMSLun(r0),SMC+Q.IOLU
        mov     TMSLun(r0),Kill+Q.IOLU
        mov     #STMOut,SMC+Q.IOPL      ; Also, tell SMC what buff to use.
        mov     #STMOSz,SMC+Q.IOPL+2
        movb    TimOut(r0),STMOut+1     ; Add Silence Detection Timeout.
        dir$    #SMC
        mov     16(sp),r0               ; And the Starting Block.
        dec     r0
        mul     #8.,r0                  ; Multiply by 8 to get the VBN
        inc     r1
        mov     r1,TMVBN                ; One word is big enough for us.
        mov     14(sp),r0               ; Get the length.
        wtse$s  Bf1Lck                  ; Wait for the buffer to be free.
        clef$s  Bf1Lck                  ; Lock the buffer for further use.
        dir$    #Lstn1                  ; Go off and start filling buffers.
        wtse$s  Bf2Lck                  ; Wait for the buffer to be free.
        clef$s  Bf2Lck                  ; Lock the buffer for further use.
        dir$    #Lstn2
        mov     #1,r1                   ; Initialize Buffer Counter to Buf 2
        mov     #2,r2                   ; and Buffer Offset.
        clr     r3                      ; Contains # of completed reads.

1$:     dir$    WaitTb(r2)              ; Wait until current buffer full.
        inc     r1                      ; Set up to use other buffer.
        bic     #177776,r1
        mov     r1,r2                   ; Calculate new offset.
        asl     r2

        cmp     #TMSBSZ,@SizeTb(r2)     ; Did we get a full block or timeout?
        beq     4$                      ; Skip next bit if we got a full block.
        mov     @SizeTb(r2),r0          ; Save what we did get!
        mov     #TMSBSZ,r1              ; Now, calculate the amount to fill.
        sub     r0,r1
        add     BufTbl(r2),r0           ; Add the base address of the buffer.
;       movb    #1,STMOut+1             ; Put in a 10 Milisecond.
;       dir$    #SMC                    ; Make the outstanding read go away.
        dir$    #Kill                   ; Kill the outstanding IO.
        bit     #1,r0                   ; Is the last byte odd?
        beq     2$
        movb    #125,(r0)+              ; Yes, shove some silence into it.
2$:     asr     r1                      ; Now we have the # of words to fill.
3$:     mov     #652,(r0)+              ; Fill with silence.
        sob     r1,3$                   ; Until no more to fill.
        mov     #177777,r1              ; Set stop flag.
4$:     mov     TMStat,-(sp)            ; Pass the Status.
        mov     TMRAB,-(sp)             ; And the RAB Address.
        mov     #TMVBN,-(sp)            ; And the VBN to get.
        mov     BufTbl(r2),-(sp)        ; And the address of the Buffer.
        mov     #TMBFSZ,-(sp)           ; And the size of said buffer.
        call    VoWrit                  ; Go store the Voice Block.
        inc     r3                      ; Up our completed counter.

        cmp     #177777,r1              ; See if we should stop now.
        beq     6$                      ; Yes, we should

        add     #8.,TMVBN               ; Add the voiceblock size to the VBN.
        cmp     r0,#2.                  ; Should we queue up another request?
        ble     5$
        wtse$s  LockTb(r2)              ; Buffer should be free, but we check
        clef$s  LockTb(r2)              ; anyways, just in case!  Then lock it.
        dir$    LstnTb(r2)              ; Queue up this buffer again.
5$:     dec     r0                      ; And read the next block if we should.
        beq     6$                      ; Oops, we shouldn't...
        jmp     1$                      ; Go start this all over again!

6$:     mov     12(sp),r0               ; Get address of Number recorded.
        mov     r3,(r0)                 ; And save the number we did get.

        mov     (sp)+,r3                ; Restore ourselves.
        mov     (sp)+,r2
        mov     (sp)+,r1
        mov     (sp)+,r0
        mov     (sp)+,12(sp)
        add     #12,sp
        rts     pc
;
        .sbttl  TMTONE - TMS Tone Emitter.
;
;       TMS Tone Emitter
;
;       Calling Procedure:
;
;       [EXTERNAL (TMTONE)]
;       Procedure TMS_Tone(var LineNumber : integer);
;         external;
;
;       This procedure emits a beep through the line specified by LineNumber.
;
TMTONE::
        mov     r0,-(sp)                ; Save our registers.

        mov     4(sp),r0                ; Get the line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which lun to use.
        asl     r0
        mov     TMSLun(r0),Tone+Q.IOLU  ; And store the lun we want.

        mov     #8.,r0                  ; Repeat this loop 8 times.
1$:     dir$    #Tone                   ; Beep Away!
        sob     r0,1$
        mov     (sp)+,r0                ; Restore ourselves.
        mov     (sp)+,(sp)
        rts     pc
;

        .sbttl  TMSCDC - Set TMS to Codec Mode.
;
;       Set TMS to Codec Mode.
;
;       Calling Procedure:
;
;       [EXTERNAL (TMSCDC)]
;       Procedure Set_TMS_To_Codec(var LineNumber : integer);
;         external;
;
;       This routine Sets the TMS unit to work in Encoded Voice mode.  This
;       allows Voice Input and Output.
;
TMSCDC::
        mov     r0,-(sp)                ; Save some registers.

        mov     4(sp),r0                ; Get the Line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which Lun to use.
        asl     r0
        mov     TMSLun(r0),SMC+Q.IOLU   ; And tell the directive which to use.
        mov     #SetCdc,SMC+Q.IOPL      ; Also, tell it what to set.
        mov     #SCdcSz,SMC+Q.IOPL+2
        DIR$    #SMC                    ; Set Multiple Characteristics.

        bcc     1$                      ; If no error, then skip to end.
        bpt

1$:     mov     (sp)+,r0                ; Restore to whence we came.
```

```
        mov     (sp)+,(sp)
        rts     pc
;
        .sbttl TMVUKD - Voice Unit Key Disable
;
;       Voice Unit Key Disable
;       Calling Procedure:
;
;       [EXTERNAL (TMVUKD)]
;       Procedure Disable_Voice_Unit_Keys;
;         external;
;
;       This procedure disables the keys on the voice unit.
;
TMVUKD::
        mov     TMSLun,SMC+Q.IOLU       ; Tell the directive to use Line 1.
        mov     #SetKyD,SMC+Q.IOPL      ; Also, tell it what to set.
        mov     #SKyDSz,SMC+Q.IOPL+2
        DIR$    #SMC                    ; Set Multiple Characteristics.

        bcc     1$
        bpt

1$:     rts     pc
;
        .sbttl  TMTOUT - Set TMS Timeout
;
;       Set TMS Timeout
;
;       Calling Procedure:
;
;       [EXTERNAL (TMTOUT)]
;       procedure Set_TMS_Timeout(var LineNumber : integer;
;                                 var TimeOut : integer);
;       external;
;
;       This procedure sets a silence timeout for the line specified.
;
TMTOUT::
        mov     r0,-(sp)                ; Save some registers.
        mov     r1,-(sp)

        mov     10(sp),r0               ; Get the Line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which Lun to use.
        asl     r0
        mov     6(sp),r1                ; Get the amount to timeout.
        mov     (r1),TimOut(r0)         ; And remember it for the read.

        mov     (sp)+,r1                ; Restore to whence we came.
        mov     (sp)+,r0
        mov     (sp)+,2(sp)
        clr     (sp)+
        rts     pc
;
        .sbttl  TMSDTM - Set TMS to DTMF Mode.
;
;       Set TMS to DTMF Mode.
;
;       Calling Procedure:
;
;       [EXTERNAL (TMSDTM)]
;       Procedure Set_TMS_To_DTMF(var LineNumber : integer);
;         external;
;
;       This routine sets the TMS unit to receive DTMF (Dual Tone,
;       Multi-Frequency) signals.  This allows for touch-tone input.
;
;       This does not put the unit into DTMF mode, as this does not
;       work correctly yet.
;

TMSDTM::
        mov     r0,-(sp)
        mov     4(sp),r0                ; Get the Line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which Lun to use.
        asl     r0
        mov     TMSLun(r0),SMC+Q.IOLU   ; And tell the directives which to use.
        mov     TMSLun(r0),Attach+Q.IOLU
        mov     TMSLun(r0),Pick+Q.IOLU
        mov     TMSLun(r0),Detach+Q.IOLU

        dir$    #ClrEF2
        mov     #SetAST,Attach+Q.IOPL   ; Tell Attach what AST to use.
        dir$    #Attach
        mov     #SetCdc,SMC+Q.IOPL      ; Set for CODEC mode.  (Bizarre, eh?)
        mov     #SCdcSz,SMC+Q.IOPL+2
        dir$    #SMC
        dir$    #Waitl                  ; Wait for the Set to complete.
        dir$    #Pick                   ; Good, now we can pick up the phone.
        dir$    #Detach                 ; And get rid of the line.

        mov     (sp)+,r0                ; Restore ourselves.
        mov     (sp)+,(sp)
        rts     pc

SetAst: dir$    #SetEF2                 ; Wake up our program.
        tst     (sp)+                   ; Remove the request on the stack.
        astx$s
;
        .sbttl  TMPICK - Pick Up the Phone.
;
;       Pick Up The Phone.
;
;       Calling Procedure:
;
;       [EXTERNAL (TMPICK)]
;       Procedure Pick_Up_Phone(var LineNumber : integer);
;         external;
;
;       This procedure takes the phone line off-hook.  It pays no heed to
;       whether it was ringing or not.
;
TMPICK::
        mov     r0,-(sp)                ; Save some registers.

        mov     4(sp),r0                ; Get the Line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which Lun to use.
        asl     r0
        mov     TMSLun(r0),Pick+Q.IOLU  ; And tell the directive.
        DIR$    #Pick                   ; Pick up the phone.

        bcc     1$                      ; If no error, then finish up.
        bpt
```

```
1$:     mov     (sp)+,r0                ; Restore to whence we came.
        mov     (sp)+,(sp)
        rts     pc
;
        .sbttl  TMDIAL - Dial a Phone Number.
;
;       Dial a Phone Number.
;
;       Calling Procedure:
;
;       [EXTERNAL (TMDIAL)]
;       Procedure Dial_Phone_Number(var LineNumber : integer;
;                                   var Number : packed array [1..80] of char;
;                                   var NumLen     : integer;
;         external;
;
;       This procedure dials the phone number specified by the array Number,
;       whose size is NumLen.  The phone number may contain all of the legal
;       characters as specified by the TMS Functional Specification for
;       dialing a phone number.
;
TMDIAL::
        mov     r0,     -(sp)           ; Save scratch register.

        mov     4(sp),  r0              ; Get address of the string record.
        mov     (r0),   Dial+Q.IOPL+2   ; Set string length in bytes.
        mov     6(sp),  Dial+Q.IOPL+0   ; Set string address.
        mov     10(sp),r0               ; Get the Line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which Lun to use.
        asl     r0
        mov     TMSLun(r0),Dial+Q.IOLU  ; And tell the directive.
        DIR$    #Dial                   ; Dial the phone.

        mov     (sp)+,  r0              ; Restore scratch register.
        mov     (sp)+,  4(sp)           ; Pop the parameter off the stack.
        add     #4,sp
        RETURN                          ; Done.
;
        .sbttl  TMHANG - Hang up the Line.
;
;       Hang Up the Phone Line.
;
;       Calling Procedure:
;
;       [EXTERNAL (TMHANG)]
;       Procedure Hang_Up_Phone(var LineNumber : integer);
;         external;
;
;       This procedure puts the phone line on hook.
;


TMHANG::
        mov     r0,-(sp)                ; Save some registers.

        mov     4(sp),r0                ; Get the Line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which Lun to use.
        asl     r0
        mov     TMSLun(r0),Hangup+Q.IOLU        ; And tell the directive.
        DIR$    #Hangup                 ; Hang up the line.
        mov     (sp)+,r0                ; Restore to whence we came.
        mov     (sp)+,(sp)
        rts     pc
;

        .sbttl  TMWDTM - Write DTMF characters.
;
;       Write DTMF characters.
;
;       Calling Procedure:
;
;       [EXTERNAL (TMWDTM)]
;       Procedure Write_DTMF_Chars(var LineNumber : integer;
;                                  var Str : packed array [1..80] of char;
;                                  var Len : integer);
;         external;
;
;       This routine writes the characters specified in the array Str to
;       the TMS unit, using DTMF mode.
;
TMWDTM::
        mov     r0,     -(sp)           ; Save scratch register.

        mov     4(sp),  r0              ; Get address of the string record.
        mov     (r0),   DTMOut+Q.IOPL+2 ; Set string length in bytes.
        mov     6(sp),  DTMOut+Q.IOPL+0 ; Set string address.
        mov     10(sp),r0               ; Get the Line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which Lun to use.
        asl     r0
        mov     TMSLun(r0),DTMOut+Q.IOLU        ; And tell the directive.
        DIR$    #DTMOut                 ; Dial the phone.

        mov     (sp)+,  r0              ; Restore scratch register.
        mov     (sp)+,  2(sp)           ; Pop the parameter off the stack.
        clr     (sp)+
        return                          ; Done.

;
        .sbttl  TMRDTM - Read DTMF Characters
;
;       Read DTMF Characters
;
;       Calling Procedure:
;
;       [EXTERNAL (TMRDTM)]
;       Procedure Read_DTMF_Chars(var LineNumber : integer;
;                                 var NumOfCharsToGet : integer;
;                                 var TimeOut : integer;
;                                 var Str : packed array [1..80] of char;
;                                 var NumOfCharsGotten : integer);
;         external;
;
;       This routine waits Timeout seconds to read NumOfCharsToGet characters
;       from the TMS unit in DTMF mode.  Upon completion, NumOfCharsGotten
;       characters are stored in the Str buffer.
;
TMRDTM::
        mov     r0,-(sp)                ; Save the registers!
        mov     r1,-(sp)
;
        mov     16(sp),r0               ; Get the Line Number.
        mov     (r0),r0
        dec     r0                      ; Calculate which Lun to use.
        asl     r0
        mov     TMSLun(r0),Attach+Q.IOLU        ; And tell the directive.
        mov     TMSLun(r0),Detach+Q.IOLU
        mov     14(sp),r0               ; Get the address of NumOfCharsToGet
        mov     (r0),NumLft             ; And remember it's contents.
        mov     10(sp),r1               ; Get address of Str Buffer.
        mov     #RedAst,Attach+Q.IOPL   ; Tell Attach which AST to use.
        dir$    #Attach                 ; Attach the line.
;
1$:     dir$    #ClrEF2                 ; Make sure we will wait.
        dir$    #Waitl                  ; And then wait.  (dumdeedumdum)
        cmp     NumLft,NumGot           ; See if we have gotten them all yet.
        bne     1$
;
        dir$    #Detach                 ; We don't need any more interruptions
        mov     6(sp),r0                ; Get address of NumOfCharsGotten.
        mov     NumGot,(r0)             ; and save what we got.
;
        mov     (sp)+,r1                ; Restore our registers
        mov     (sp)+,r0
        mov     (sp)+,10(sp)
        add     #10,sp
        return
;


RedAST: mov     (sp)+,r0                ; Get the character off of the stack.
        neg     r0                      ;
        movb    r0,(r1)+                ; Add character to the buffer.
        inc     NumGot                  ; Update our counter.
        dir$    #SetEF2                 ; Tell the rest of the world.
        astx$s                          ; and return.
;

        .sbttl  TMINIT - Initialize TMS unit.
;
;       Initialize TMS unit.
;
;       Calling Procedure:
;
;       [EXTERNAL (TMINIT)]
;       procedure Initialize_TMS_Unit;
;
;       This procedure initializes the various things that have to be
;       initialized.  This should be called before any other TMS handler
;       routines are called.
;
TMINIT::
        setf$s  Bf1Lck                  ; Unlock the buffers to start.
        setf$s  Bf2Lck
        alun$s  #L1$LUN,#"XT,#1         ; Assign the luns we will need.
        bcs     1$
        alun$s  #L2$LUN,#"XT,#2
        bcs     1$
        alun$s  #L3$LUN,#"XT,#3
        bcc     2$

1$:     bpt

2$:     rts     pc
;
```

```
        .sbttl  TMS & RMS Data Definitions.
;
        .psect  TMSDAT,rw,d,con,gbl,rel
;
;       TMS Data Definitions.
;
TMStat: .word   0         ; Address of the Status Block.
TMRAB:  .word   0         ; Address of the RAB.
TMVBN:  .word   0         ; The VBN of the current block.
BUFTBL: .word   TMBuf1
        .word   TMBuf2
SizeTb: .word   ISB1+2
        .word   ISB2+2
TimOut: .blkw   3         ; Stores the Timeout for each line.
SetCdc: .byte   XT.DMD    ; Set to Codec Mode SMC Buffer.
        .byte   XT.ENC
SCDCSZ = .-SetCdc
SetKyD: .byte   XT.TAK    ; Disable Voice Unit Keyboard.
SKyDSZ = .-SetKyD
STmOut: .byte   XT.TTO    ; Set TimeOut.
        .byte   0
STmOSz = .-StmOut
SDTMBF: .byte   XT.DMD    ; Set to DTMF Mode SMC Buffer.
        .byte   XT.DTM
SDTMSZ = .-SDTMBF
ISB1:   .blkw   2         ; I/O Status Buffer.
ISB2:   .blkw   2
TMBFSZ: .word   RMSBSZ    ; The size of the Voice Buffer.
TMBuf1: .blkb   4096.     ; The first TMS Voice Buffer.
TMSBSZ = .-TMBUF1
RMSBSZ = .-TMBUF1
TMBuf2: .blkb   TMSBSZ    ; The second TMS Voice Buffer.
LockTb:
Bf1Lck: .word   XT$EF3    ; The buffer 1 Lock.
Bf2Lck: .word   XT$EF4    ; The buffer 2 Lock.
TMSLun: .word   L1$LUN    ; The Luns for the various Lines.
        .word   L2$LUN
        .word   L3$LUN
TonBuf: .rept   30.
        .word   177400    ; Alternating all 1's and all 0's.
        .endr
TonSiz = .-TonBuf
NumGot: .word   0         ; The number of DTMF chars we have gotten so far.
NumLft: .word   0         ; The number we still have to get.
;
        .psect  TMSMAC,rw,d,con,gbl,rel
;
;       TMS Macro Tables.
;
WaitTb: .word   Wait1
        .word   Wait2
TalkTb: .word   Talk1
        .word   Talk2
TlkSiz: .word   Talk1+Q.IOPL+2
        .word   Talk2+Q.IOPL+2
LstnTb: .word   Lstn1
        .word   Lstn2


;       TMS Macro Definitions.
;
Wait1:  WTSE$   XT$EF2
```

```
Wait2:  WTSE$   XT$EF1
ClrEF2: CLEF$   XT$EF2
SetEF2: SETF$   XT$EF2
Talk1:  QIO$    IO.WAL,0,XT$EF1,,ISB1,Bf1Ast,<TMBuf1,TMSBSZ,0>
Talk2:  QIO$    IO.WAL,0,XT$EF2,,ISB2,Bf2Ast,<TMBuf2,TMSBSZ,0>
Lstn1:  QIO$    IO.RAL,0,XT$EF1,,ISB1,Bf1Ast,<TMBuf1,TMSBSZ,0>
Lstn2:  QIO$    IO.RAL,0,XT$EF2,,ISB2,Bf2Ast,<TMBuf2,TMSBSZ,0>
Tone:   QIOW$   IO.WAL,0,XT$EF1,,ISB1,,<TonBuf,TonSiz,0>
SMC:    QIOW$   SF.SMC,0,XT$EF1,,ISB1,,<0,0>
Pick:   QIOW$   IO.ANS,0,XT$EF1,,ISB1,,<TMBuf1,1>
Dial:   QIOW$   IO.CON,0,XT$EF1,,ISB1,,<0,0,0>
Hangup: QIOW$   IO.HNG,0,XT$EF1,,ISB1
DTMOut: QIOW$   IO.WAL,0,XT$EF1,,ISB1,,<0,0,0>
DTMIn:  QIOW$   IO.RAL,0,XT$EF1,,ISB1,,<0,0,0>
Kill:   QIOW$   IO.KIL,0,XT$EF1,,ISB1
Attach: QIOW$   IO.ATA,0,XT$EF1,,ISB1,,<0>
Detach: QIOW$   IO.DET,0,XT$EF1,,ISB1
;
        .psect  TMSRMS,rw,d,con,gbl,rel
;
;       RMS Data Definitions.
;
;       Define some pool space.
;
        pool$b
        p$fab   2                ; Two files.
        p$rab   2                ; Two Streams.
        p$buf   2*512.           ; Two sequential file IO buffers.
        p$bdb   4                ; Four Buffer Block Pool
        pool$e
;
RMSLun: .word   V1$LUN           ; Logical Unit Table.
        .word   V2$LUN
;
RabTbl: .word   Rab1             ; Rab lookup Table.
        .word   Rab2
;
Fab1:   fab$b
        f$alq   8.               ; Initial allocation.
        f$deq   8.               ; Default Extension size.
        f$fac   fb$rea!fb$wrt    ; Access is read/write.
        f$org   fb$rel           ; Relative organization.
        f$fop   fb$dlk!fb$ctg    ; don't lock + contiguous.
        f$shr   fb$nil           ; no access sharing.
        f$rat   fb$blk           ; Blocked format.
        f$fna   Name             ; Name block address.
        fab$e
;
Rab1:   rab$b
        r$fab   Fab1             ; link to the Fab.
        r$mbf   8                ; Number of blocks to xfer at a time.
;
Fab2:   fab$b
        f$alq   8.               ; Initial allocation.
        f$deq   8.               ; Default Extension size.
        f$fac   fb$rea!fb$wrt    ; Access is read/write.
        f$org   fb$rel           ; Relative organization.
        f$fop   fb$dlk!fb$ctg    ; don't lock + contiguous.
        f$shr   fb$nil           ; no access sharing.
        f$rat   fb$blk           ; Blocked format.
        f$fna   Name             ; Name block address.
        fab$e
;
```

```
Rab2:   rab$b
        r$fab   Fab2            ; link to the Fab.
        r$mbf   8               ; Number of blocks to xfer at a time.
        rab$e
;
Name:   .blkb   80.             ; Name buffer.
        .end

;-------------------- end of MACRO-11 subroutines file  -------------
```

End of Appendix

APPENDIX C

TMS ERROR AND STATUS CODES

In addition to status codes generated by the executive (documented in the P/OS documentation), the TMS driver generates the following codes:

Status codes are returned in the first byte of the I/O Status Block. Unless otherwise specified, the second byte is undefined.

Success:

IS.SUC	Successful completion of the requested operation.
IS.TMO	Read operation completed successfully, but with timeout.

Error:

IE.TMO	Operation failed due to exceeding a user supplied timeout. (not applicable to READ operations)
IE.FHE	Fatal hardware error on device. The TMS hardware returned an unexpected error code. The TMS hardware error code is contained in the second byte, and should be reported to Digital.
IE.RSU	Shared resource in use. An attempt was made to activate a component of TMS which is already in use and therefore not available.
IE.BAD	Bad Parameters. The meaning of this depends on the value of the second byte: 0 Illegal value specified in the buffer of an IO.WSD QIO.

	2	The TMS firmware has reported a SYNTAX error. The most likely case is an illegal digit in a telephone number. Please report any other occurence of this to Digital.
	3	The TMS firmware has reported illegal option. This can occur if an attempt is made to transmit data while not in SERIAL, CODEC, or DTMF data mode.
	1	The TMS firmware has reported illegal digit. This occurs when a DTMF digit is specified in a telephone number but rotary dialing has been requested.

IE.DAO Data overrun. This indicates that TMS was not able to get a response from the host. This should not happen unless there is a hardware malfunction. Note that data overruns due to application programs not processing data fast enough are not reported.

IE.ABO Operation aborted. If the second byte is zero, this means that a QIO was aborted as the result of an IO.KIL or that a READ operation was aborted as a result of termination of the current on-line state (e.g. loss of carrier).

A second byte of SE.VAL means that an attempt was made to specify an inappropriate value for a characteristic in an IO.SMC.

A second byte of SE.NIH means that an attempt was made to reference an unimplemented or undefined characteristic or an attempt was made to GET a SET-only characteristic.

In these two cases, the second word of the IOSB indicates the point in the characteristics list which caused the error. Characteristics prior to the error point were correctly pro- cessed.

IE.DNR Device not ready. This is returned in the following cases:

If a READ or WRITE is issued in Codec mode while the opposite command is active (Codec is half-duplex).

If a read is issued and TMS is not on-line in CODEC, SERIAL, or DTMF data mode.

If the zero timeout form of the IO.ORG!TF.TMO command (test on-line status) is issued and TMS is not on-line.

If the TMS hardware cannot complete an operation within the TMS hardware timeout period or the operation is cancelled by pre-emption (such as picking up the telephone).

IE.ALC	Illegal User Buffer. This is returned if a telephone number is too long (greater than 48 digits after translation and inclusion of the prefix and/or suffix strings, if any were specified) or if more than 30 bytes of indicator control commands were specified in an IO.WSD QIO, or if the buffer specified in an IO.LTI is not 6 bytes. This is also returned by the executive if the executive address checking determines that the user buffer is illegal.
IE.OFL	Device Off-line. This is returned by the executive if the TMS driver is not active. It is returned by the driver to terminate I/O requests active when the driver is being unloaded. Note that prior to the first load of the TMS driver (when there is no database), this is not returned, since it is not possible to have assigned a LUN to TMS. See the P/OS documentation for more details.
IE.CNR	Connection Rejected. This is returned if an attempt is made to issue an IO.CON if carrier is up in SERIAL mode.

End of Appendix

INDEX

Telephone Management System
(TMS) Programmer's Manual
Order No. AA-AD34A-TH

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____________

Name ____________ Date ____________

Organization ____________

Street ____________

City ____________ State ____________ Zip Code ____________
or
Country

Please cut along this line.

---- Do Not Tear - Fold Here and Tape ----

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional 300 Series Publications
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754

---- Do Not Tear - Fold Here ----

Cut Along Dotted Line