



Telephone Manager Developer's Kit

Version 1.0

RU145LL A



Apple Computer, Inc.

20525 Mariani Avenue, M/S 33-G
Cupertino, CA 95014
(408) 996-1010
TLX 171-576

To reorder products, please call:
1-800-282-2732 (in the United States)
1-800-637-0029 (in Canada)
1-408-562-3910 (International)



Telephone Manager Developer's Guide

 APPLE COMPUTER, INC.

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose.

The Apple logo is a registered trademark of Apple Computer, Inc. Use of the "keyboard" Apple logo (Option-Shift-k) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

© Apple Computer, Inc., 1991
20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

Apple, the Apple logo, APDA, AppleLink, and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Classic is a registered trademark, licensed to kApple Computer, Inc.

Adobe Illustrator and PostScript are trademarks of Adobe Systems, Inc.

ITC Garamond and ITC Zapf Dingbats are registered trademarks of International Typeface Corp.

Microsoft is a registered trademark of Microsoft Corp.

Varityper is a registered trademark of Varityper, Inc.

Contents

Figures and Tables / v

Preface / vii

1 Introduction to the Telephone Manager / 1

About the Telephone Manager / 2

Telephone Manager concepts / 4

 Telephone terminals / 4

 Directory numbers / 4

 Call appearances / 5

 Telephone Manager records / 6

System requirements / 6

2 Inside the Telephone Manager / 7

Data structures of the Telephone Manager / 8

The telephone record / 9

 Telephone record data structure: `TELRecord` / 10

 Telephone record data structure: `TELTermRecord` / 12

The directory-number record / 15

 Directory-number data structure / 16

The call-appearance record / 22

 Call-appearance data structure / 23

Telephone Manager routines / 30

Preparing to handle calls / 31

Custom configuration of a telephone tool / 36

Interfacing with a scripting language / 40

Opening, using, and closing the terminal / 42

Handling events / 45

Handling messages / 47

Placing and receiving calls / 54

Using Drop, Hold, Transfer, Forward, and Conference / 59

Using less-common supplementary features / 65

- Accessing special features of switches and tools / 72
- Localizing configuration strings / 75
- Monitoring and controlling the terminal / 77
- Controlling directory numbers / 85
- Controlling call appearances / 90
- Miscellaneous routines / 94
- Routines your application must provide / 95

3 Writing Telephone Tools / 97

- About writing a telephone tool../ 98
- The six tool resources / 98
 - The bundle resource / 99
 - The validation code resource / 100
 - The setup-definition code resource / 102
 - The scripting language interface code resource / 107
 - The localization code resource / 109
- `config`: the configuration record / 111

4 Writing Your Tool's Main Code Resource / 113

- The main code resource / 114
- Messages that the main code resource accepts / 115
- Messages that the main code resource sends / 149
 - General call-appearance messages / 151
 - Incoming/outgoing call-appearance messages / 155
 - Call-appearance message for transferring calls / 158
 - Call-appearance messages for conference calls / 159
 - Directory-number messages / 160
 - Terminal messages / 162

Appendix A Result Codes for Routines / 165

Appendix B Message Codes for Applications / 169

- Call-appearance message codes for applications / 170
- Directory-number message codes for applications / 183
- Terminal message codes for applications / 185

Appendix C Call-Appearance States / 189

Index / 191

Figures

CHAPTER 1 Introduction to the Telephone Manager / 1

Figure 1-1 Where the Telephone Manager fits into the Communications Toolbox / 2

Figure 1-2 How the Telephone Manager interacts with applications and tools / 3

Figure 1-3 An example state progression for an outgoing call appearance / 5

Figure 1-4 An example state progression for an incoming call appearance / 5

Figure 1-5 An example state progression for an active call appearance / 5

CHAPTER 2 Inside the Telephone Manager / 7

Figure 2-1 How the Telephone Manager data structures are related / 8

Figure 2-2 A sample tool-settings dialog box / 34

Preface

The *Telephone Manager Developer's Guide* provides definitive information for application software developers and telephone tool developers who want to use services provided by the Telephone Manager. For application software developers, this document describes and shows how to use the Telephone Manager routines that make it easier to write telephony applications for the Apple Macintosh computer. For telephone tool developers, this document shows how to develop tools that can be used by the Telephone Manager.

About this document

Chapter 1 contains an overview of the Telephone Manager and describes the hardware and software you need to run it. Chapter 2 describes the Telephone Manager data structures and application-programming routines. Chapters 3 and 4 show how to create a telephone tool. Although tool developers need to read Chapters 3 and 4, most application developers do not. Appendix A describes result codes that the Telephone Manager routines return. Appendix B describes messages that the Telephone Manager relays from telephone tools to applications. Appendix C describes call-appearance states.

The *Telephone Manager Developer's Guide* is written for experienced programmers. Readers should know how to program the Macintosh and how to use the Macintosh Communications Toolbox, and should have some familiarity with telephony applications. The next section lists resources for reference information about the technical concepts used in this document.

For more information

Refer to the following books in the Apple Technical Library and Apple Communications Library, published by Addison-Wesley, for additional information about the subjects covered in this manual:

- *Designing Cards and Drivers for the Macintosh Family*
- *Human Interface Guidelines: The Apple Desktop Interface*
- *Inside Macintosh* (Volumes I–VI, X–Ref)
- *Inside the Macintosh Communications Toolbox*
- *Programmer's Introduction to the Macintosh Family*
- *Technical Introduction to the Macintosh Family*

You may also refer to the following documents from APDA:

- *Apple ISDN Telephone Tool*
- *Software Development for International Markets: A Technical Reference*
- *Macintosh Technical Notes*

APDA offers convenient worldwide access to over 300 development tools, resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the quarterly *APDA Tools Catalog* featuring the most current versions of Apple development tools and the most popular third-party development tools. Ordering is easy; there are no membership fees, and application forms are not required for most of our products. APDA offers convenient payment and shipping options, including site licensing.

To order products or get additional information, contact

APDA

Apple Computer, Inc.

20525 Mariani Avenue, M/S 33-G

Cupertino, CA 95014-6299

USA

800-282-2732 (United States)

800-637-0029 (Canada)

408-562-3910 (International)

Fax: 408-562-3971

Telex: 171-576

AppleLink address: APDA

Conventions used in this document

The following notations are used in this document to draw attention to particular items of information:

- ◆ *Note:* Information that is interesting or useful.
- △ **Important** A note of particular importance.
- ▲ **Warning** A point that warns you to be cautious.

Names of routines (procedures or functions), constants, and code fragments appear in Courier, a special typeface, as in the following example:

```
PROCEDURE GetDown(andBoogie : ONEMORETIME);
```


Chapter 1 **Introduction to the Telephone Manager**

THIS CHAPTER gives you an overview of the Telephone Manager. It explains how the Telephone Manager works with the Macintosh Communications Toolbox and presents key concepts relating to the Telephone Manager.

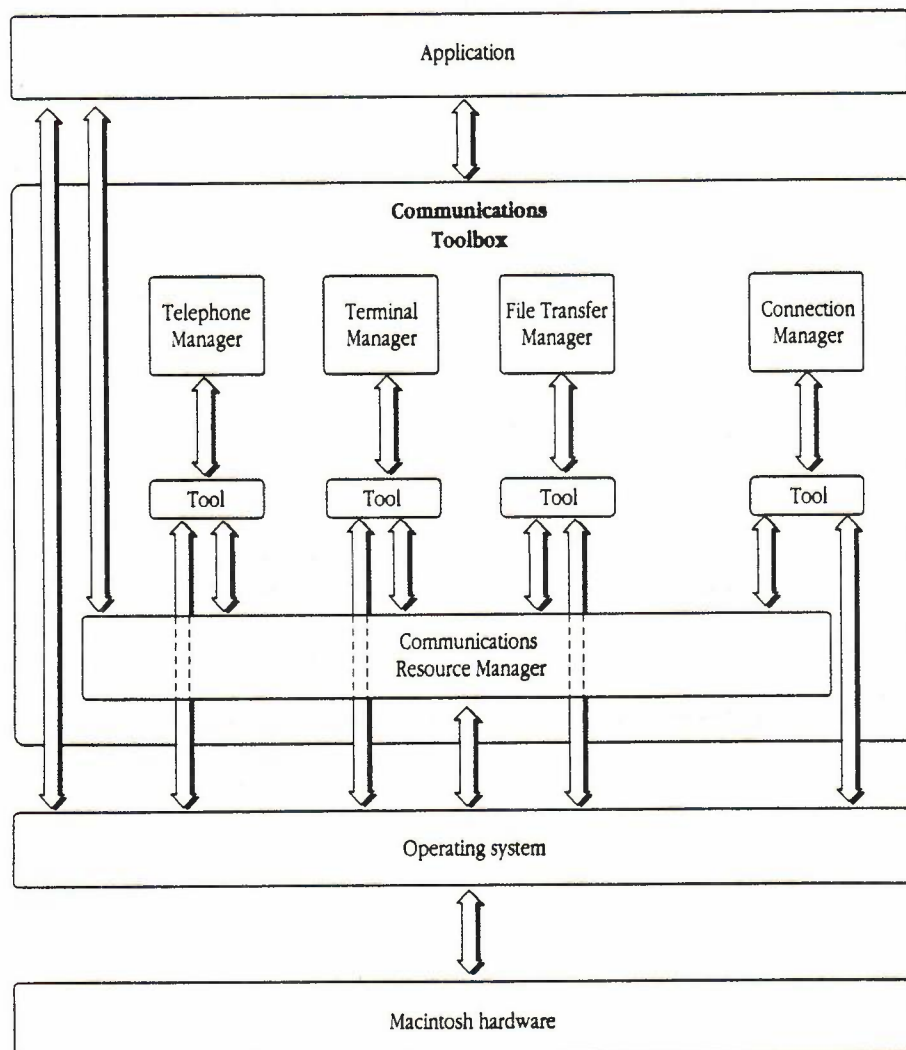
About the Telephone Manager

The Telephone Manager is a new manager for the Macintosh Communications Toolbox. It provides a programming interface that lets you develop a variety of telephony applications, including screen-based telephony applications and Macintosh-based answering machines.

Using the Telephone Manager, applications can offer telephone services to users yet operate independently of the user's network type or telephone type. For example, a Macintosh application can serve as a virtual telephone—whether the telephone network provides Integrated Services Digital Network (ISDN) service or “plain old telephone service” (POTS), and whether the attached telephone set is a speakerphone or a mobile telephone.

After you install the Telephone Manager, it works with the Communications Toolbox in much the same way that other managers do, as shown in *Figure 1-1*.

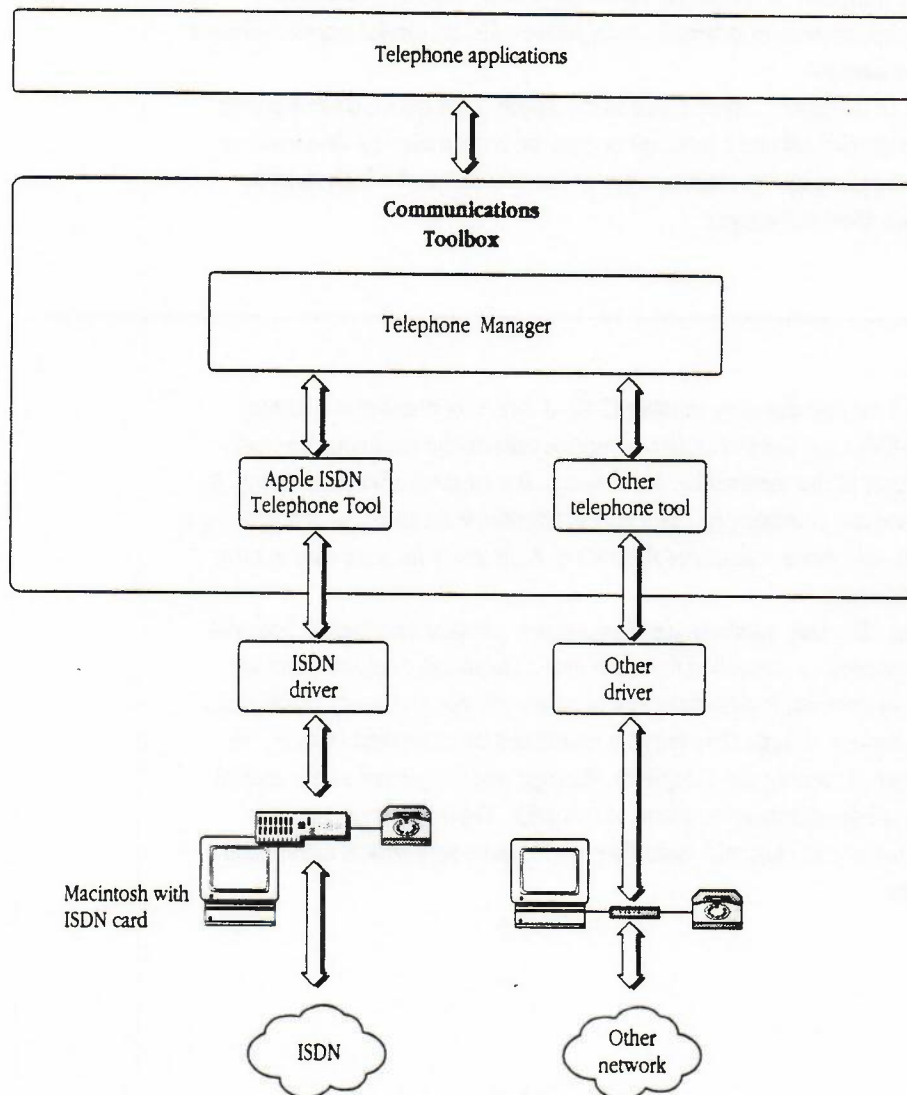
■ **Figure 1-1** Where the Telephone Manager fits into the Communications Toolbox



The Telephone Manager accesses the telephone network through telephone tools, which the user installs and which the Telephone Manager manages. Telephone tools control the terminal drivers of the telephony hardware (such as an ISDN card) installed on the user's system. Each telephone tool is designed for specific hardware. For example, the Apple ISDN Telephone Tool is designed for the Apple ISDN NB Card.

Figure 1-2 shows how the Telephone Manager interacts with applications and tools. An application makes a request of the Telephone Manager when it needs a telephone service—for example, when it needs a call dialed. The Telephone Manager then sends this request to one of the telephone tools it manages. The tool provides the service according to the specifics of the telephone network protocol. While providing the service, the tool sends messages to the Telephone Manager, passing parameters that indicate how handling of the request is proceeding. The Telephone Manager then relays the tool's messages to the application.

■ **Figure 1-2** How the Telephone Manager interacts with applications and tools



Telephone Manager concepts

To use the Telephone Manager, you first need to understand several key concepts: telephone terminals, directory numbers, call appearances, and Telephone Manager records. This section briefly summarizes each of these concepts.

Telephone terminals

In this book, a telephone terminal (a “terminal,” for short) is hardware, such as an ISDN card, that provides the physical interface between a Macintosh computer and a telephone network switch—such as a private branch exchange (PBX) or a central-office switch. A telephone terminal also provides, optionally, the physical interface between the telephone network switch and a telephone set attached to the Macintosh computer. A telephone set is any device, such as a table-top telephone, used to manually dial, answer, or otherwise manipulate calls. Be careful not to confuse a telephone set and a telephone terminal.

A terminal can consist of an integrated device (such as the Apple ISDN NB Card) or separate devices (such as the Apple Serial NB Card and a modem). A terminal is controlled by device-driver software, which is generally supplied by the manufacturer of the terminal and which must be compatible with the Macintosh Device Manager.

Directory numbers

Each installed terminal has at least one directory number (DN). A directory number is a named reference point, such as (408) 555-1212, used to initiate or receive calls on the terminal. Directory numbers are assigned to the user of the terminal by, for instance, the local telephone company. A terminal can have multiple directory numbers, just as a typical telephone set in an office might have multiple buttons, one for each telephone number of that office. A directory number can, in turn, have multiple network subaddresses.

In the Telephone Manager, directory numbers are of two types: physical and logical. Physical directory numbers can be monitored or controlled from the user's Macintosh computer and are physically associated with it. In contrast, logical directory numbers are not physically associated with the user's Macintosh computer, though they may be monitored or controlled from it. For example, a Macintosh computer, if running the Telephone Manager and connected to the control port of a PBX, might monitor all the directory numbers of that PBX. The Telephone Manager would treat those directory numbers as “logical,” since they would have no physical association with the Macintosh computer.

Call appearances

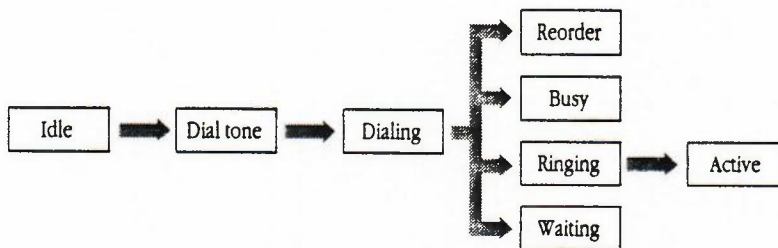
Each directory number can carry at least one call appearance (a CA, or “call,” for short). A call appearance is a connection between two or more directory numbers, as when one telephone user places a call to another. Directory numbers can carry multiple call appearances concurrently—as on a telephone that has a Call Waiting feature or a Conference feature.

At any particular time, each call appearance is in one particular state. For instance, the call appearance might be in an alerting state (ringing or flashing, for example), a held state (on hold), or an active state (meaning voice or data can flow end to end).

- ◆ **Note:** The state of a call appearance can change to “idle” (`telcAidlestate`) directly from any other state. Refer to Appendix C for a complete list of the call-appearance states recognized by the Telephone Manager.

Figures 1-3 and 1-4 show example sequences of states through which outgoing call appearances and incoming call appearances might progress. Figure 1-5 shows an example sequence of states through which an active call appearance might progress.

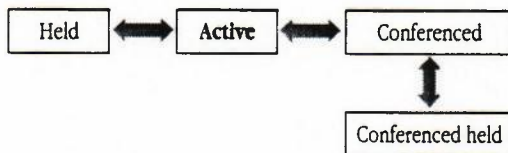
- **Figure 1-3** An example state progression for an outgoing call appearance



- **Figure 1-4** An example state progression for an incoming call appearance



- **Figure 1-5** An example state progression for an active call appearance



Telephone Manager records

For each terminal, directory number, and call appearance, the Telephone Manager maintains a corresponding data structure—a telephone record, directory-number record, or call-appearance record. Applications and telephone tools reference these records using handles—telephone handles, directory-number handles, and call-appearance handles. Chapter 2 describes each of the Telephone Manager records in detail.

System requirements

To run the Telephone Manager, you need one of the following Macintosh computers:

- a Macintosh Plus, Classic®, SE, SE/30, Portable, LC, II, IIx, IICx, IIsi, IICI, or IIfx computer with at least 2 megabytes of RAM and a hard disk
- a Macintosh 128K, 512K, or 512K enhanced computer with a Macintosh Plus Logic Board Upgrade, at least 2 megabytes of RAM, and a hard disk

Your Macintosh computer must be running Macintosh system software version 7.0, of which the Macintosh Communications Toolbox is a part.

Chapter 2 **Inside the Telephone Manager**

THIS CHAPTER describes the main data structures of the Telephone Manager and each of the routines the Telephone Manager provides.

In this chapter, the term *your application* refers to the application you are writing for the Macintosh, which will implement telephone services for users. Be careful not to confuse the services your application provides with the services that tools provide.

To use the Telephone Manager, you need to be familiar with

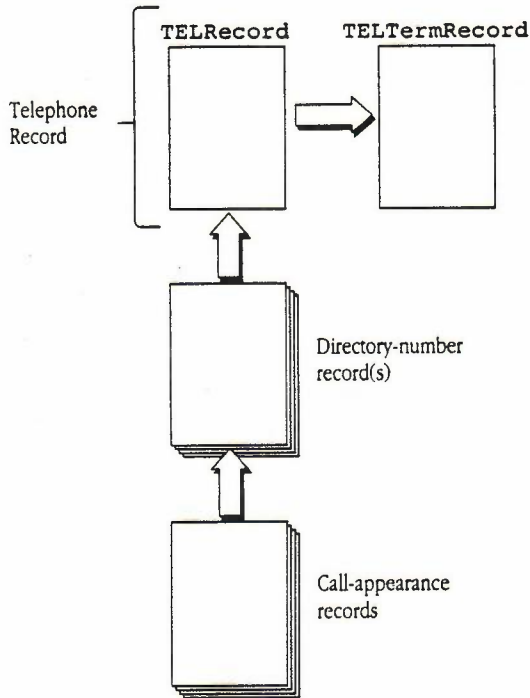
- the Communications Resource Manager and the Communications Toolbox Utilities (described in *Inside the Macintosh Communications Toolbox*)
- any one of the other managers in the Macintosh Communications Toolbox—for instance, the Connection Manager (described in *Inside the Macintosh Communications Toolbox*)

Data structures of the Telephone Manager

The Telephone Manager maintains three main types of data structures—the telephone record, the directory-number record, and the call-appearance record. In this respect, the Telephone Manager differs from most other Communications Toolbox managers, which maintain only one main type of data structure. For example, the Connection Manager maintains only the connection record.

Figure 2-1 shows how the main data structures of the Telephone Manager relate to one another.

■ **Figure 2-1** How the Telephone Manager data structures are related



An important aspect of the Telephone Manager data structures is that they allow the interface of Telephone Manager routines to be network-independent. This independence lets applications use Telephone Manager services without regard for the underlying network type or telephone type. In other words, to place a call, an application tells the Telephone Manager what number to dial. The Telephone Manager then invokes a telephone tool, which figures out exactly how to place a call on the given telephone network switch.

Another important aspect of the Telephone Manager data structures is that they let applications use multiple instances of the same tool. The same tool can be used by different processes at the same time or by different threads in a given application.

The sections that follow explain each of the main Telephone Manager data structures.

△ **Important** In the descriptions of the Telephone Manager data structures and in the rest of this book, all strings are Pascal-style strings, unless otherwise noted. △

The telephone record

The telephone record describes a particular terminal and its associated tool, and contains pointers to Telephone Manager internal data structures. The Telephone Manager uses this information to “translate” the network-independent routines used by an application into a service implemented according to the protocols of a particular network. Most of the fields in the telephone record are filled in when an application calls `TELNew`, described later in this chapter.

The Telephone Manager creates a telephone record for each terminal an application uses. For example, assume that a Macintosh computer has two ISDN cards and that an application needs to communicate over both cards. The application would request that the Telephone Manager create two telephone records, one for each card.

Because the telephone record describes how communication takes place on a given terminal, an application can communicate on more than one terminal at the same time. The application need only create a new telephone record for each terminal.

△ **Important** Your application, in order to be compatible with future releases of the Telephone Manager, should not directly manipulate the fields of the telephone record, except `refCon` and `userData`. The Telephone Manager provides routines that applications can use to change telephone record fields. These routines are discussed later in this chapter. △

Telephone record data structure: **TELRecord**

```
TYPE
    TELHandle      =    ^TELPtr;
    TELPtr         =    ^TELRecord;
    TELRecord      =    RECORD
        procID      :    INTEGER;

        flags       :    TELFlags;
        reserved    :    INTEGER;

        refCon      :    LONGINT;
        userData    :    LONGINT;

        defproc     :    ProcPtr;

        config      :    Ptr;
        oldConfig   :    Ptr;

        pTELTerm    :    TELTermPtr;

        telPrivate  :    LONGINT;
        reserved1   :    LONGINT;
        reserved2   :    LONGINT;

        pTELTermSize :    LONGINT;
        version     :    INTEGER;

    END;
```

procID

procID is the telephone tool ID. This value is dynamically assigned by the Telephone Manager when your application calls `TELGetProcID`.

flags

flags is a bit field that indicates certain specifics about a terminal when the telephone record is first created. The bit masks for **flags** are as follows:

```
TYPE
    flags =    LONGINT;

CONST
    telNoMenus    =    $10000;
    telQuiet      =    $20000;
    {All other bits in flags are reserved for Apple.}
```

Your application can turn on the `telNoMenus` bit, the `telQuiet` bit, or both when calling `TELNew` (discussed later in this chapter). The telephone tool will not display any custom menus if your application sets the `telNoMenus` bit. The telephone tool will not display any status dialog boxes or error alerts if your application sets the `telQuiet` bit. If your application turns the `telQuiet` bit on, it is responsible for displaying status dialog boxes and error alerts that the tool would have displayed. Applications typically use these two bits to hide the telephone tool from the user.

reserved

`reserved` is reserved for the Telephone Manager. Your application must not use this field.

refCon

`refCon` is a 4-byte field that your application can use. This field is ignored by the Telephone Manager.

userData

`userData` is a 4-byte field that your application can use. This field is ignored by the Telephone Manager.

defproc

`defproc` is a procedure pointer to the main code resource of the telephone tool, and is maintained by the Telephone Manager.

config

`config` is a pointer to a data block that is private to the telephone tool. It can contain information like the directory numbers and telephone features associated with the terminal; the contents vary from tool to tool.

Your application can store the contents of `config` to save the state of a tool and terminal. The structure, size, and contents of the configuration record are set by the tool. Your application can determine the size of the configuration record by calling `GetPtrSize`, overwriting its contents by using `BlockMove`, and then validating the contents with `TELValidate`.

Your application can use `TELGetConfig` and `TELSetConfig` to manipulate fields in this record. For details, see "Interfacing with a Scripting Language," later in this chapter. Your application can save the state of the telephone record by saving the string returned from `TELGetConfig`. Also, your application can restore the configuration of the telephone record by passing a saved string to `TELSetConfig`.

oldConfig

`oldConfig` is a pointer to a data block that is private to the telephone tool and contains the most recently saved version of `config`. Your application is responsible for setting `oldConfig` when the user saves a session document. Your application can use `TELGetConfig` and `TELSetConfig` to manipulate fields in this record.

pTELTerm

pTELTerm is pointer to a record of type TELTermRecord, defined in the next section.

telPrivate

telPrivate is reserved for use by telephone tools.

reserved1

reserved1 is reserved for the Telephone Manager. Your application must not use this field.

reserved2

reserved2 is reserved for the Telephone Manager. Your application must not use this field.

pTELTermSize

pTELTermSize contains the size (in bytes) of the record pointed to by the field pTELTerm.

version

version is the version number of the Telephone Manager for which the telephone tool is intended. The tool fills in this value when the terminal is opened.

Telephone record data structure: TELTermRecord
TYPE

```

TELTermPtr          =      ^TELTermRecord;
TELTermRecord       =      RECORD
    tRef             :      INTEGER;

    featureFlags      :      TELFeatureFlags;

    handsetSpeakerVol :      INTEGER;
    speakerphoneVol   :      INTEGER;
    handsetMicVol      :      INTEGER;
    ringerVol          :      INTEGER;
    otherVol           :      INTEGER;
    ringerTypes        :      INTEGER;
    hasDisplay         :      INTEGER;
    displayRows        :      INTEGER;

    numDNs             :      INTEGER;

    maxAllocCA         :      INTEGER;
    curAllocCA         :      INTEGER;

    reserved           :      LONGINT;

```

END;

tRef

tRef is the terminal reference number. This value is dynamically assigned by the telephone tool when your application calls **TELNew**. The **tRef** field is private to the telephone tool; your application must not change the value of this field.

featureFlags

featureFlags is a bit field that indicates which features and characteristics the terminal has when it is first opened (by the routine **TELOpenTerm**). The bit masks for **featureFlags** are as follows:

TYPE

featureFlags = LONGINT;

CONST

pcmAvail = \$00000001;
hasHandset = \$00000002;
hasSpeakerphone = \$00000004;
canOnHookDial = \$00000008;
hasRinger = \$00000010;
canSetDisplay = \$00000020;
hasKeypad = \$00000040;
hasVideo = \$00000080;
hasOther = \$00000100;

crossDNConference = \$00000200;
hasSubaddress = \$00000400;
hasUserUserInfo = \$00000800;

{All other bits in **featureFlags** are reserved for Apple.}

If a tool sets a bit in **featureFlags**, the terminal has the corresponding characteristic or capability. If **pcmAvail** is set, the terminal can access pulse-code-modulated (PCM) data. If **hasHandset**, **hasSpeakerphone**, and **canOnHookDial** are set, the terminal has an attached handset and a speakerphone, and can dial while the handset's receiver is on the switch hook. If **hasRinger** and **canSetDisplay** are set, the terminal has its own ringer (as opposed to that of the telephone) and can write to the telephone's display. If **hasKeypad** and **hasVideo** are set, the terminal has a typical 12-button keypad and has a videophone. The flag **hasOther** is reserved for Apple.

The remaining flags of **featureFlags** indicate whether the corresponding features are available. If **crossDNConference** is set, the terminal can group calls into a conference, even if the calls are on different directory numbers. If **hasSubaddress** and **hasUserUserInfo** are set, the network to which the terminal is attached allows subaddressing and user-to-user information.

handsetSpeakerVol

`handsetSpeakerVol` indicates the number of levels to which the volume control of the handset can be set. If `handsetSpeakerVol` is zero, the volume of the handset is fixed; it cannot be adjusted.

speakerphoneVol

`speakerphoneVol` indicates the number of levels to which the volume control of the speakerphone can be set. If `speakerphoneVol` is zero, the volume of the speakerphone is fixed; it cannot be adjusted.

handsetMicVol

`handsetMicVol` indicates the number of levels to which the volume control of the microphone can be set. If `handsetMicVol` is zero, the volume of the microphone is fixed; it cannot be adjusted.

ringerVol

`ringerVol` indicates the number of levels to which the volume control of the ringer can be set. If `ringerVol` is zero, the volume of the ringer is fixed; it cannot be adjusted.

otherVol

`otherVol` is reserved by Apple for future use. Your application must not use this field.

ringerTypes

`ringerTypes` indicates how many types of ringing sounds the terminal can emit. If `ringerTypes` is zero, the terminal is not capable of emitting a ringing sound.

hasDisplay

`hasDisplay` indicates the number of characters per line that the telephone can display. If `hasDisplay` is zero, the telephone has no display.

displayRows

`displayRows` indicates the number of lines that the telephone can display. If `hasDisplay` is zero, `displayRows` must also be zero.

numDNs

`numDNs` indicates how many directory numbers are currently assigned to the terminal.

maxAllocCAs

maxAllocCAs indicates the maximum number of call appearances that the network has allocated to this terminal for placing independent outgoing calls. (For example, on a residential POTS line with Call Waiting, **maxAllocCAs** would be set to 1, meaning the line can place only one outgoing call at a time.) If **maxAllocCAs** is set to -1, there is no maximum.

curAllocCAs

curAllocCAs indicates how many of the call appearances currently on the terminal are in states other than **telCAIdleState**.

reserved

reserved is reserved for Apple. Your application must not use this field.

The directory-number record

The Telephone Manager creates a directory-number record for each telephone number, or directory number, associated with a particular telephone terminal. Each directory-number record describes the characteristics of a particular directory number (such as its subscribed features) and its state (such as "in use"). The Telephone Manager accesses this information when placing calls, receiving calls, or otherwise handling calls for that directory number.

The fields of the directory-number record are filled in by the telephone tool when the record is created. Because these fields are updated only at the application's request, the directory-number record is like a "snapshot." It describes the characteristics and state of the directory number as of the most recent update.

- △ **Important** Your application, in order to be compatible with future releases of the Telephone Manager, should not directly manipulate the fields of the directory-number record, except **refCon** and **userData**. △

Directory-number data structure

TYPE

```
TELDNHandle      =    ^TELDNPtr;
TELDNPtr         =    ^TELDNRecord;
TELDNRecord      =    RECORD

    dnRef         :    INTEGER;
    dn            :    StringPtr;
    dnSubaddress  :    StringPtr;
    dnPartyName   :    StringPtr;
    hTEL          :    TELHandle;
    maxAllocCAs   :    INTEGER;
    curAllocCAs   :    INTEGER;
    dnType        :    INTEGER;

    featureFlags  :    TELDNFeatureFlags;

    numPageIDs    :    INTEGER;
    numIntercomIDs :    INTEGER;
    numPickupIDs  :    INTEGER;

    forwardFlags  :    TELDNForwardFlags;
    iForwardDN    :    StringPtr;
    iForwardSubaddress :    StringPtr;
    iForwardPartyName :    StringPtr;
    bForwardDN    :    StringPtr;
    bForwardSubaddress :    StringPtr;
    bForwardPartyName :    StringPtr;
    naForwardDN   :    StringPtr;
    naForwardSubaddress :    StringPtr;
    naForwardPartyName :    StringPtr;
    naForwardRings :    INTEGER;

    telDNPrivate  :    LONGINT;
    refCon        :    LONGINT;
    userData      :    LONGINT;
    reserved      :    LONGINT;
```

END;

dnRef

dnRef is the directory-number reference number, dynamically assigned by the telephone tool to refer to this particular directory number. Your application is permitted to read **dnRef** but should not change the value of the field.

dn

dn is a pointer to a Pascal-style string storing the name (telephone number) associated with the directory number. Tools recognize only the following characters in this string: the digits 0 through 9, the number sign (*), the comma (,), the asterisk (*), and the exclamation point (!). The comma is treated as a 1-second pause; the exclamation point is treated as a flash-hook. All other characters are parsed as spaces but do not stop the telephone tool's processing. If your application accepts directory-number names that include alphabetic characters, it should translate these characters to the appropriate digits.

dnSubaddress

dnSubaddress is a pointer to a Pascal-style string storing the network subaddress, if any, associated with the directory number.

dnPartyName

dnPartyName is a pointer to a Pascal-style string storing the name of the person to whom this directory number is assigned.

hTEL

hTEL contains a handle to the telephone record (and hence the terminal) with which this directory number is associated.

maxAllocCAs

maxAllocCAs indicates the maximum number of call appearances that the network has allocated to this directory number for placing independent outgoing calls. If **maxAllocCAs** is set to -1, there is no maximum.

curAllocCAs

curAllocCAs is the number of calls currently allocated for this directory number.

dnType

dnType represents the type of this directory number (internal, external, and so on). Refer to the description of the routine **TELDNLookupByIndex** for an explanation of directory-number types.

featureFlags

featureFlags is a bit field that indicates which features (other than call forwarding) and characteristics a directory number has, and indicates their state—subscribed, available, or active. The bit masks for **featureFlags** are as follows:

```

TYPE
    featureFlags          =    LONGINT;

CONST
    dndSub                =    $00000001;
    dndAvail              =    $00000002;
    dndActive              =    $00000004;

    voiceMailAccessSub    =    $00000008;
    voiceMailAccessAvail  =    $00000010;
    voiceMailAccessActive =    $00000020;

    pagingSub             =    $00000040;
    pagingAvail           =    $00000080;
    pagingActive          =    $00000100;

    intercomSub           =    $00000200;
    intercomAvail         =    $00000400;
    intercomActive        =    $00000800;

    dnSelectSub           =    $00001000;
    dnSelectAvail         =    $00002000;
    dnSelectActive        =    $00004000;

    callPickupSub         =    $00008000;
    callPickupAvail       =    $00010000;

    dnInUse               =    $00020000;
    logicalDN             =    $00040000;
    dnAccessible          =    $00080000;
    canInitiate           =    $00100000;
    voiceMessageWaiting   =    $00200000;
    {All other bits in featureFlags are reserved by Apple for future use.}

```

The bits `dndSub`, `dndAvail`, and `dndActive` show the state of Do Not Disturb. `voiceMailAccessSub`, `voiceMailAccessAvail`, and `voiceMailAccessActive` show the state of the Voice Mail feature. `pagingSub`, `PagingSubAvail`, and `PagingSubActive` show the state of Paging. `intercomSub`, `intercomAvail`, and `intercomActive` show the state of Intercom.

The bits `dnSelectSub`, `dnSelectAvail`, and `dnSelectActive` show the state of Directory Number Select. `callPickupSub` and `callPickupAvail` show the state of Call Pickup. `dnInUse` indicates whether the directory number is in use, meaning that call appearances are currently allocated for it. `logicalDN` indicates whether the directory number is a logical directory number, meaning that no corresponding physical channel is connected to the terminal. `dnAccessible` indicates whether the directory number can accept commands—for example, `TELSetupCall`. `canInitiate` indicates whether the directory number can place unrestricted calls. `voiceMessageWaiting` indicates whether a voice-mail message is waiting for this directory number.

numPageIDs

numPageIDs is the number of page IDs, or “page keys,” configured for this directory number.

numIntercomIDs

numIntercomIDs is the number of intercom IDs, or “intercom keys,” configured for this directory number.

numPickupIDs

numPickupIDs is the number of pickup IDs configured for this directory number.

forwardFlags

forwardFlags is a bit field that indicates which capabilities of call forwarding this directory number has and indicates the state of those capabilities (subscribed, available, and active). If a tool sets a bit in forwardFlags, the corresponding forwarding capability is in the indicated state. The bit masks for forwardFlags are as follows:

TYPE

forwardFlags : LONGINT;

CONST

immediateForwardSub = \$00000001;
immediateForwardAvail = \$00000002;
immediateForwardActive = \$00000004;

busyForwardSub = \$00000008;
busyForwardAvail = \$00000010;
busyForwardActive = \$00000020;

noAnswerForwardSub = \$00000040;
noAnswerForwardAvail = \$00000080;
naFwdActive = \$00000100;

busyNAForwardSub = \$00000200;
busyNAForwardAvail = \$00000400;
busyNAForwardActive = \$00000800;

{All other bits in forwardFlags are reserved by Apple for future use.}

The bits immediateForwardSub, immediateForwardAvail, and immediateForwardActive indicate the state of Immediate Call Forwarding—whether it is subscribed, available, or active. Likewise, busyForwardSub, busyForwardAvail, and busyForwardActive show the state of Forward On Busy. The bits noAnswerForwardSub, noAnswerForwardAvail, and noAnswerForwardActive indicate the state of Forward On No Answer. The bits busyNAForwardSub, busyNAForwardAvail, and busyNAForwardActive indicate the state of Forward On Busy And No Answer.

iForwardDN

iForwardDN is a pointer to a string storing either `NIL` or the telephone number to which calls are forwarded when the Immediate Call Forwarding feature is active.

iForwardSubaddress

iForwardSubaddress is a pointer to a string storing either `NIL` or the network subaddress of the telephone number to which calls are forwarded when the Immediate Call Forwarding feature is active.

iForwardPartyName

iForwardPartyName is a pointer to a string storing either `NIL` or the name of the person to whom calls are forwarded when the Immediate Call Forwarding feature is active.

bForwardDN

bForwardDN is a pointer to a string storing either `NIL` or the telephone number to which calls are forwarded when the Forward On Busy feature is active.

bForwardSubaddress

bForwardSubaddress is a pointer to a string storing either `NIL` or the network subaddress of the telephone number to which calls are forwarded when the Forward On Busy feature is active.

bForwardPartyName

bForwardPartyName is a pointer to a string storing either `NIL` or the name of the person to whom calls are forwarded when the Forward On Busy feature is active.

naForwardDN

naForwardDN is a pointer to a string storing either `NIL` or the telephone number to which calls are forwarded when the Forward On No Answer feature is active.

naForwardSubaddress

naForwardSubaddress is a pointer to a string storing either `NIL` or the network subaddress of the telephone number to which calls are forwarded when the Forward On No Answer feature is active.

naForwardPartyName

naForwardPartyName is a pointer to a string storing either `NIL` or the name of the person to whom calls are forwarded when the Forward On No Answer feature is active.

naForwardRings

naForwardRings is the number of times the telephone or terminal rings before Forward On No Answer is activated.

telDNPrivate

`telDNPrivate` is reserved for use by telephone tools.

refCon

`refCon` is a 4-byte field that your application can use. This field is ignored by the Telephone Manager.

userData

`userData` is a 4-byte field that your application can use. This field is ignored by the Telephone Manager.

reserved

`reserved` is reserved for Apple. Your application must not use this field.

The call-appearance record

The Telephone Manager creates a call-appearance record for each call appearance associated with a particular directory number. Each call-appearance record describes the characteristics of a particular call appearance (such as the directory number being called) and its state (such as "on hold"). The Telephone Manager accesses this information when handling that call appearance.

Most of the fields of the call-appearance record are filled in by the telephone tool when the record is created. The Telephone Manager updates the fields of the record only at your application's request. Thus, the call-appearance record is like a "snapshot." It describes the characteristics and state of the call appearance as of the most recent update.

△ **Important** Your application, in order to be compatible with future releases of the Telephone Manager, should not directly manipulate the fields of the call-appearance record (except `refCon`, `userData`, and `connectTime`). △

Call-appearance data structure

TYPE

```
TELCAHandle      = ^TELCAPtr;
TELCAPtr         = ^TELCAREcord;
TELCAREcord      = RECORD
    caRef        : INTEGER;

    hTELDN       : TELDNHandle;
    hTEL         : TELHandle;
    caState       : INTEGER;
    relatedCA     : TELCAHandle;
    connectTime   : LONGINT;

    intExt        : INTEGER;
    callType      : INTEGER;
    dialType      : INTEGER;
    bearerType    : INTEGER;
    rate          : INTEGER;

    rmtDN         : StringPtr;
    rmtPartyName  : StringPtr;
    rmtSubaddress : StringPtr;
    routeDN       : StringPtr;
    routePartyName : StringPtr;
    routeSubaddress : StringPtr;

    priority      : INTEGER;

    confLimit     : INTEGER;

    featureFlags  : TELCAFeatureFlags;
    otherFeatures : TELCAOtherFeatures;

    telCAPrivate  : LONGINT;
    refCon        : LONGINT;
    userData      : LONGINT;
    reserved      : LONGINT;
```

END;

caRef

caRef is the call-appearance reference number, dynamically assigned by the telephone tool to refer to this particular call appearance. Your application is permitted to read **caRef** but should not change the value of the field.

hTELDN

hTELDN is a handle to the directory-number record for this call appearance.

hTEL

hTEL is a handle to the telephone record for directory number **hTELDN**.

caState

caState is an integer that represents the state of the call associated with this call appearance. The valid values of **caState** are as follows:

CONST

telCAIdleState	=	0;	{No call exists now}
telCAInUseState	=	1;	{This call active at another terminal}
telCAOfferState	=	2;	{Call being offered to this terminal}
telCAQueuedState	=	3;	{Call being queued to this terminal}
telCAAlertingState	=	4;	{Call alerting at this terminal}
telCADialToneState	=	5;	{Initiated outgoing call has dial tone}
telCADialingState	=	6;	{Initiated outgoing call now dialing}
telCAWaitingState	=	7;	{Initiated outgoing call awaiting destination's response}
telCARingingState	=	8;	{Outgoing call now ringing}
telCABusyState	=	9;	{Destination busy or unreachable}
telCAHeldState	=	10;	{This call put on hold by this terminal}
telCAConferencedState	=	11;	{This call now part of a conference}
telCAActiveState	=	12;	{This call active: parties can speak or exchange data}
telCAREorderState	=	13;	{This call in a reorder state}
telCAUnknownState	=	15;	{Call state unknown}

relatedCA

relatedCA is a handle to a call-appearance record with which the current call appearance is associated, as in a conference or a transfer.

connectTime

connectTime is the time at which the connection was made. Your application is responsible for maintaining this field if needed.

intExt

intExt indicates whether the call is internal or external. **intExt** has one of the following values:

telInternalCall	=	0;	{Internal call}
telExternalCall	=	1;	{External call}
telUnknownCallOrigin	=	2;	{Call type unknown}

callType

callType indicates the route by which the call reached this terminal—for instance whether it was transferred or forwarded. **callType** can have any of the following values:

telNormalIn	=	0;	{Direct inbound call}
telForwardedNoAnswer	=	1;	{Inbound forward on no answer}
telForwardedBusy	=	2;	{Inbound forward on busy}
telForwardedImmediate	=	3;	{Inbound forward immediate}
telTransfer	=	4;	{Inbound call transfer}
telDeflected	=	5;	{Inbound deflected call}
telIntercepted	=	6;	{Inbound intercepted call}
telDeflectRecall	=	7;	{Recall of deflected call}
telCallbackIn	=	8;	{Inbound call back}
telParkRecall	=	9;	{Recall of parked call}
telPickup	=	10;	{Inbound call pickup}
telTransferRecall	=	11;	{Recall of transferred call}

dialType

dialType indicates the type of dialable number as one of the following values:

telDNDialable	=	0;	{This dn could be dialed via TELSetupCall }
telDNNorthAmerican	=	1;	{rmtDN is standard North America 10 digit number }
telDNInternational	=	2;	{rmtDN is an international number}
telDNAalmostDialable	=	3;	{rmtDN is almost dialable; it is missing a prefix, such as 9 or 1 }
telDNUnknown	=	15;	{Unknown whether dn is dialable }

bearerType

`bearerType` is unused and set to zero.

rate

`rate` is unused and set to zero.

rmtDN

`rmtDN` is a pointer to the remote telephone number associated with this call. The telephone number is a Pascal-style string. If the remote telephone number is unknown, `rmtDN` is `NIL`.

rmtPartyName

`rmtPartyName` is a pointer to the name of the remote party associated with this call. The name is a Pascal-style string. If the name of the remote party is unknown, `rmtPartyName` is `NIL`.

rmtSubaddress

`rmtSubaddress` is a pointer to the network subaddress of `rmtDN`. If there is no subaddress, `rmtSubaddress` is `NIL`.

routedDN

`routedDN` is a pointer to the telephone number through which this call was routed. The telephone number is a Pascal-style string. If the call was not routed or if the routing telephone number is unknown, `routedDN` is `NIL`.

routePartyName

`routePartyName` is a pointer to the name of the party associated with `routedDN`. The name is a Pascal-style string. If the name of the party is unknown, `routePartyName` is `NIL`.

routeSubaddress

`routeSubaddress` is a pointer to the network subaddress of `routedDN`. If there is no subaddress, `routeSubaddress` is `NIL`.

priority

`priority` is reserved by Apple for future use.

confLimit

`confLimit` is the maximum number of parties that can be concurrently conferenced with this call appearance. This maximum includes the two parties initially associated with the call appearance. If `confLimit` equals zero, there is no limit on the number of calls that can be conferenced.

featureFlags

featureFlags is a bit field that indicates which features can be applied to this call appearance and indicates their state (subscribed, available, or active). The bit masks for **featureFlags** are as follows:

TYPE

featureFlags = LONGINT;

CONST

holdSub = \$00000001;
holdAvail = \$00000002;
holdActive = \$00000004;

conferenceSub = \$00000008;
conferenceAvail = \$00000010;
conferenceActive = \$00000020;

conferenceDropSub = \$00000040;
conferenceDropAvail = \$00000080;

conferenceSplitSub = \$00000100;
conferenceSplitAvail = \$00000200;
numToConferenceRequired = \$00000400;

transferSub = \$00000800;
transferAvail = \$00001000;
transferActive = \$00002000;
caRelated = \$00004000;

{All other bits in **featureFlags** are reserved by Apple for future use.}

The bits **holdSub**, **holdAvail**, and **holdActive** show the state of the Hold feature; **conferenceSub**, **conferenceAvail**, and **conferenceActive** show the state of the Conference feature. Likewise, **conferenceDropSub**, **conferenceDropAvail**, **conferenceSplitSub**, and **conferenceSplitAvail** show the state of the features Conference Drop and Conference Split. If **numToConferenceRequired** is set, your application must indicate the number of calls in this conference when invoking the routine **TELConferencePrep**. The **caRelated** bit is set if this call appearance is specified in the **relatedCA** field of another call-appearance record. The bits **transferSub**, **transferAvail**, and **transferActive** show the state of the Transfer feature.

otherFeatures

otherFeatures is a bit field that indicates which features, other than those in **featureFlags**, can be applied to this call appearance and indicates their state (subscribed, available, active, or clearable). The bit masks for **otherFeatures** are as follows:

TYPE

```
otherFeatures      =      LONGINT;
```

CONST

```
callbackSub        =      $00000001;
callbackAvail      =      $00000002;
callbackActive     =      $00000004;
callbackClearSub   =      $00000008;
```

```
callbackNowSub     =      $00000010;
callbackNowAvail   =      $00000020;
```

```
callbackBusy       =      $00000040;
callbackNoAnswer   =      $00000080;
callbackReturnsRef =      $00000100;
```

```
parkSub           =      $00000200;
parkAvail         =      $00000400;
parkActive        =      $00000800;
parkRetrieveSub    =      $00001000;
parkRetrieveWithID =      $00002000;
parkWithID        =      $00004000;
```

```
rejectable        =      $00008000;
deflectable       =      $00010000;
acceptable        =      $00020000;
```

{All other bits in otherFeatures are reserved by Apple for future use.}

The bits `callbackSub`, `callbackAvail`, `callbackActive`, and `callbackClearSub` show the state of the Call Back feature (subscribed, available, active, or clearable). The bits `callbackNowSub` and `callbackNowAvail` show the state of the Call Back Now feature. When the Call Back feature is available, `callbackBusy` is set if the feature can be activated when the remote party is busy. Similarly, `callbackNoAnswer` is set if the Call Back feature can be activated when the remote party does not answer. If `callbackReturnsRef` is set, the Call Back feature returns a reference number, allowing the user to identify multiple callbacks.

The bits `parkSub`, `parkAvail`, and `parkActive` show the state of the Call Park feature. If `parkRetrieveSub` is set, the Call Park Retrieve feature is subscribed. `parkRetrieveWithID` and `parkWithID` show whether the Call Park feature assigns an ID or requests the destination directory number when parking calls. `Rejectable`, `deflectable`, and `acceptable` show whether the terminal can reject, deflect, or accept the call.

telCAPrivate

`telCAPrivate` is reserved for use by telephone tools.

refCon

`refCon` is a 4-byte field that your application can use.

userData

userData is a 4-byte field that your application can use.

reserved

reserved is reserved for Apple. Your application must not use this field.

Telephone Manager routines

The sections that follow describe the routines that tools and applications can use to access Telephone Manager services. These routines are protocol independent; your application does not need to be familiar with the specifics of a particular network protocol in order to use these telephone services.

△ **Important** For a list and description of each result code returned by the Telephone Manager routines, refer to Appendix A. △

Here is an alphabetical listing of the routines described in this section.

InitTEL / 31	TELDNDClear / 67	TELOpenTerm / 42
TELAcceptCall / 56	TELDNDispose / 89	TELOtherFeatureImplement / 73
TELActivate / 45	TELDNDSet / 66	TELOtherFeatureList / 72
TELAlert / 82	TELDNEventsSupp / 52	TELOtherFunction / 73
TELAnswerCall / 56	TELDNLookupByIndex / 86	TELPaging / 70
TELCADispose / 93	TELDNLookupByName / 87	TELParkCall / 68
TELCASupp / 53	TELDNMsgHand / 49	TELRejectCall / 57
TELCallbackClear / 66	TELDNSelect / 88	TELResetTerm / 42
TELCallbackNow / 66	TELDrop / 59	TELResume / 45
TELCallbackSet / 65	TELEnglishToIntl / 76	TELRetrieve / 60
TELCallPickup / 68	TELEvent / 46	TELRetrieveParkedCall / 69
TELCALookup / 90	TELForwardClear / 62	TELSetConfig / 41
TELCAMsgHand / 50	TELForwardSet / 62	TELSetDisplay / 84
TELChoose / 34	TELGetCAFlags / 92	TELSetHooksw / 78
TELCloseTerm / 43	TELGetCAInfo / 91	TELSetupCall / 54
TELClrCAMsgHand / 52	TELGetCAState / 92	TELSetupCleanup / 38
TELClrDNMsgHand / 51	TELGetConfig / 40	TELSetupFilter / 37
TELClrTermMsgHand / 51	TELGetDisplay / 83	TELSetupItem / 38
TELConferenceEstablish / 64	TELGetDNFlags / 88	TELSetupPostflight / 39
TELConferencePrep / 63	TELGetDNInfo / 87	TELSetupPreflight / 36
TELConferenceSplit / 64	TELGetHooksw / 77	TELSetupSetup / 37
TELConnect / 55	TELGetInfo / 44	TELSetVolume / 80
TELCountCAs / 90	TELGetProcID / 31	TELTermEventsSupp / 52
TELCountDNs / 85	TELGetTELVersion / 94	TELTermMsgHand / 48
TELDefault / 33	TELGetToolName / 94	TELToolFunctions / 74
TELDeflectCall / 58	TELGetVersion / 94	TELTransferBlind / 61
TELDialDigits / 55	TELGetVolume / 78	TELTransferEstablish / 61
TELDiscard / 43	TELHold / 60	TELTransferPrep / 60
	TELIde / 43	TELValidate / 33
	TELIntercom / 71	TELVoiceMailAccess / 70
	TELIntlToEnglish / 75	
	TELMenu / 45	
	TELNew / 32	

Preparing to handle calls

Before your application can place calls or receive them, it must initialize the Telephone Manager (by calling `InitTEL`), find out the `procID` of the tool it requires (by calling `TELGetProcID`), create a telephone record (by calling `TELNew`), and then configure the telephone tool (by restoring `config` from a saved document, or by calling either `TELChoose` or `TELSetConfig`).

InitTEL

Initializing the Telephone Manager

`InitTEL` initializes the Telephone Manager. Your application should call this routine only once, before making any other calls.

Before calling `InitTEL` your application must initialize the Macintosh Toolbox, the Communications Resource Manager, and the Communications Toolbox Utilities.

▲ **Warning** Your application must initialize the Communications Resource Manager (by calling `InitCRM`) and then the Communications Toolbox Utilities (by calling `InitCTBUtilities`), whether or not your application uses any of their calls. ▲

Function	<code>InitTEL : TELErr;</code>
Description	<code>InitTEL</code> returns an operating-system error code if appropriate. Your application must check for the presence of the Communications Toolbox before calling this function. Sample code under "Determining Whether the Managers are Installed" in Appendix C of <i>Inside the Macintosh Communications Toolbox</i> shows you how your application can make this check.
Result Codes	<code>noErr</code> , <code>telNoCommFolder</code> , <code>telInitFailed</code> , <code>telNoTools</code>

TELGetProcID

Getting the `procID` of a tool

Your application should call `TELGetProcID` just before creating a new telephone record, to find the `procID` of a tool.

Function	<code>TELGetProcID(name: Str255): INTEGER;</code>
Description	<code>name</code> specifies the filename of a telephone tool—for example, "Apple ISDN Telephone Tool." If there is a telephone tool with the specified <code>name</code> in the Extensions folder, the <code>procID</code> is returned. If there is no such telephone tool, <code>TELGetProcID</code> returns <code>-1</code> .

Creating a telephone record

Before your application can handle calls, it must create a telephone record so the Telephone Manager knows what type of terminal you are using. `TELNew` creates a new telephone record of type `TELRecord`; fills in the fields it can, based on the parameters passed to it; and returns a handle to the new record in `TELHandle`. In this new record, the field `version` is filled in by the telephone tool. The tool also fills in any fields it can in the record's associated `TELTermRecord`.

`TELNew` makes two calls to `TELDefault` (described later in this chapter) to fill in `config` and `oldConfig`. The Telephone Manager then loads the telephone tool main code resource, moves it high in the current heap, and locks it. If an error occurs that prevents a new telephone record from being created (for example, running out of memory), `TELNew` passes back `NIL` in `TELHandle`.

Function `TELNew(procID : INTEGER; refCon : LONGINT; userData : LONGINT) : TELHandle;`

Description `procID` is dynamically assigned by the Telephone Manager to tools at run time. Applications should not store `procID` values in settings files. Instead, they should store tool names, which can be converted to `procID` values with the `TELGetProcID` routine. Your application should use the ID that `TELGetProcID` returns for `procID`. `refCon` and `userData` are fields that your application can use.

TELDefault

Initializing the telephone record

TELDefault fills theConfig with the default configuration specified by the telephone tool.

Function TELDefault(VAR theConfig: Ptr; procID: INTEGER; allocate: BOOLEAN): TELErr;

Description TELDefault is called by TELNew twice, when that routine fills in the config and oldConfig fields of a new telephone record.

procID is equal to the value returned by TELGetProcID.

If allocate is TRUE, the telephone tool allocates space for theConfig in the current heap zone.

Result Codes noErr, telUnknownErr

TELValidate

Validating the configuration fields

TELValidate validates the configuration fields and other fields of the telephone record by comparing their values with those allowed by the telephone tool. TELNew and TELSetConfig call TELValidate after they have created a new telephone record.

Function TELValidate(hTEL: TELHandle): BOOLEAN;

Description If the validation succeeds, the Telephone Manager returns FALSE and the tool leaves the configuration record unchanged. If the validation fails, the Telephone Manager returns TRUE and the tool fills the configuration record with default values by calling TELDefault.

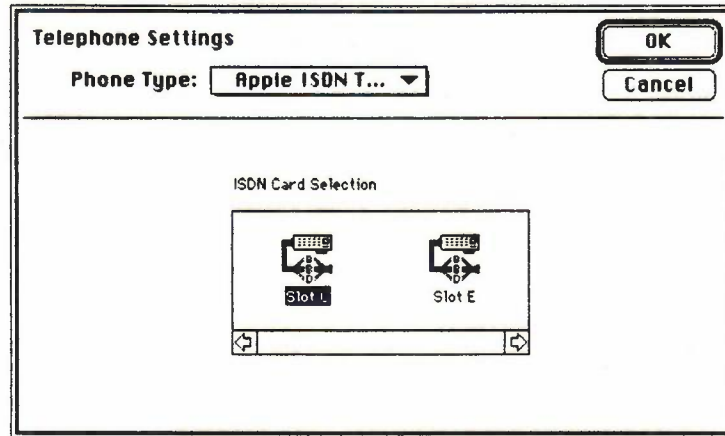
Your application can call TELValidate after restoring a configuration, to verify that the telephone record contains the correct information, as in the following example:

```
BlockMove(saveConfig,hTEL^.config,GetPtrSize(hTEL^.config));
IF TELValidate(hTEL) THEN BEGIN
    { validate failed }
END
ELSE BEGIN
    { validate succeeded }
END
```

Configuring a telephone tool

An application can configure a telephone tool in one of three ways. The easiest and most straightforward way is by calling the `TELChoose` routine. This routine presents the user with a dialog box similar to the one shown in *Figure 2-2*.

■ **Figure 2-2** A sample tool-settings dialog box



The second way your application can configure a telephone tool is by presenting the user with a custom tool-settings dialog box. This method is much more difficult and involves calling six routines. The routines are described in the next section, “Custom Configuration of a Telephone Tool.”

The third way your application can configure a telephone tool is by using the scripting language interface, described in “Interfacing With a Scripting Language,” later in this chapter. This method allows your application to bypass user interface elements.

Regardless of which configuration method you use, the configuration is stored in a telephone record, which your application refers to using a handle. If your application reconfigures the tool by either configuration method, the Telephone Manager returns a new handle to a new telephone record and disposes of all handles associated with the old telephone record.

Function	<code>TELChoose(VAR hTEL:TELHandle; where: Point; idleProc: ProcPtr): TELErr;</code>
Description	<code>TELChoose</code> allows users of your application to choose and configure a telephone tool by filling in a dialog box like the one in <i>Figure 2-2</i> .

`hTEL` is a handle to a telephone record. The fields of this record are filled in when the user of your application selects or reconfigures a tool. Each telephone record contains the configuration of only one tool on one terminal, yet a tool may support multiple terminals. For this reason, applications that support multiple terminals must make multiple calls to `TELNew` and then to either `TELChoose` or `TELSetConfig`.

`where` is the point, specified in global coordinates, where the upper-left corner of the dialog box should appear. It is recommended that your application place the dialog box as close as possible to the upper-left corner of the screen, because the size of the dialog box varies from tool to tool.

`idleProc` is a procedure, with no parameters, that the Telephone Manager calls each time `TELChoose` calls the setup dialog box filter procedure. Pass `NIL` if your application has no `idleProc`. Refer to *Inside the Macintosh Communications Toolbox* for more information about `idleProc`.

Result Codes

`noErr`, `telChooseAborted`, `telChooseCancel`, `telChooseDisaster`,
`telChooseFailed`, `telChooseOKMajor`, `telChooseOKMinor`,
`telChooseOKTermChanged`

Custom configuration of a telephone tool

Your application can create a custom tool-settings dialog box and present it to the user by calling the following six Telephone Manager routines: `TELSetupPreflight`, `TELSetupSetup`, `TELSetupFilter`, `TELSetupItem`, `TELSetupCleanup`, and `TELSetupPostflight`. Using these routines is more involved than calling `TELChoose`, but they provide your application with much more flexibility.

To build a list of available telephone tools, use the routine `CRMGetIndToolName`, which is described in *Inside the Macintosh Communications Toolbox*.

`TELSetupPreflight`

Setting up the custom tool-settings dialog box

`TELSetupPreflight` returns a handle to a dialog item list that your application appends to the custom tool-settings dialog box. The handle comes from the telephone tool. (The calling application uses `AppendDITL`, discussed in *Inside the Macintosh Communications Toolbox*.) This handle is not a resource handle. Your application is responsible for disposing of the handle when done with it.

The telephone tool can use `TELSetupPreflight` to allocate a block of private storage and to store the pointer to that block in `magicCookie`. The `magicCookie` value should be passed to the other routines used to set up the custom tool-settings dialog box.

Function `TELSetupPreflight(procID: INTEGER; VAR magicCookie: LONGINT): Handle;`

Description `procID` is the ID for the telephone tool being configured. Your application should get this value by calling the `TELGetProcID` routine, discussed earlier in this chapter.

- ◆ *Note:* The `refCon` of the custom tool-settings dialog box should point to a data structure (shown next) in which the first two bytes are the tool `procID` and the next four bytes are `magicCookie`. `UserItem` routines, for example, may require `procID` to obtain tool resources.

```
TYPE
    chosenDLOGdata    =    RECORD
        procID        :    INTEGER
        magicCookie    :    LONGINT
    END;
```

TELSetupSetup

Setting up custom tool-settings dialog box items

`TELSetupSetup` tells the telephone tool to set up controls (such as radio buttons or check boxes) in the dialog item list returned by `TELSetupPreflight`.

Procedure

```
TELSetupSetup(procID: INTEGER; theConfig: Ptr; count: INTEGER;  
theDialog: DialogPtr; VAR magicCookie: LONGINT);
```

Description

`procID` is the ID for the telephone tool that is being configured. Your application should use the same value for `procID` that it passed to `TELSetupPreflight`.

`theConfig` is a pointer to a configuration record for the tool being configured.

`count` is the number of the first item in the dialog item list appended to the dialog box.

`theDialog` is the dialog box in which configuration is taking place.

`magicCookie` is a pointer to private storage for the telephone tool.

TELSetupFilter

Filtering custom tool-settings dialog box events

Your application calls `TELSetupFilter` as a filter procedure before it calls the standard modal dialog box filter procedure for the custom tool-settings dialog box. This routine allows telephone tools to filter events in the custom tool-settings dialog box.

Function

```
TELSetupFilter(procID: INTEGER; theConfig: Ptr; count: INTEGER;  
theDialog: DialogPtr; VAR theEvent: EventRecord; VAR theItem:  
INTEGER; VAR magicCookie: LONGINT): BOOLEAN;
```

Description

`procID` is the ID for the telephone tool that is being configured. Your application should use the same value for `procID` that it passed to `TELSetupPreflight`.

`theConfig` is a pointer to the configuration record for the tool being configured.

`count` is the number of the first item in the dialog item list appended to the dialog box.

`theDialog` is the dialog box performing the configuration.

`theEvent` is the event record for which filtering is to take place.

`theItem` can return the item clicked in the dialog box.

`magicCookie` is a pointer to private storage for the telephone tool.

If the event passed in was handled, `TELSetupFilter` returns `TRUE`. `FALSE` indicates that your application should perform standard dialog box filtering.

Processing custom tool-settings dialog box events

TELSetupItem processes events for controls in the custom tool-settings dialog box.

Procedure

```
TELSetupItem(procID: INTEGER; theConfig: Ptr; count: INTEGER;  
theDialog: DialogPtr; VAR theItem: INTEGER; VAR magicCookie:  
LONGINT);
```

Description

procID is the ID for the telephone tool being configured. Your application should use the same value for procID that it passed to TELSetupPreflight.

theConfig is a pointer to the configuration record for the tool being configured.

count is the number of the first item in the dialog item list appended to the dialog box.

theDialog is the dialog box performing the configuration.

theItem is the item clicked in the dialog box. This value can be modified and sent back.

magicCookie is a pointer to private storage for the telephone tool.

Performing clean-up operations

TELSetupCleanup disposes of any storage allocated in TELSetupPreflight and performs other clean-up operations. If your application needs to shorten a dialog box, it should do so after calling this routine.

Procedure

```
TELSetupCleanup(procID: INTEGER; theConfig: Ptr; count: INTEGER;  
theDialog: DialogPtr; VAR magicCookie: LONGINT);
```

Description

procID is the ID for the telephone tool that is being configured. Your application should use the same value for procID that it passed to TELSetupPreflight.

theConfig is a pointer to the configuration record for the tool being configured.

count is the number of the first item in the dialog item list appended to the dialog box.

theDialog is the dialog box performing the configuration.

magicCookie is a pointer to private storage for the telephone tool.

TELSetupPostflight

Closing the tool file

TELSetupPostflight closes the tool file if it is not being used by any sessions.

Procedure

```
TELSetupPostflight(procID:INTEGER);
```

Description

procID is the ID for the telephone tool that is being configured. Your application should use the same value for procID that it passed to TELSetupPreflight.

Interfacing with a scripting language

Your application does not have to rely on users making selections from dialog boxes in order to configure a telephone tool. `TELGetConfig` and `TELSetConfig` provide the services that your application needs to interface with a scripting language.

TELGetConfig

Getting the configuration string

`TELGetConfig` gets a configuration string from the telephone tool.

Function `TELGetConfig(hTEL: TELHandle): Ptr;`

Description `TELGetConfig` returns a null-terminated, C-style string from the telephone tool, containing tokens that fully describe the configuration of the telephone record. (For an example, see the description of `TELSetConfig`.) If an error occurs, `TELGetConfig` returns `NIL`.

Your application is responsible for disposing of `Ptr`.

Because the value that `TELGetConfig` returns specifies a null-terminated, C-style string, that string is not subject to the length limitations of Pascal strings.

Result Codes None

Setting the configuration with a string

TELSetConfig passes a configuration string to the telephone tool.

Function

TELSetConfig(hTEL: TELHandle; thePtr: Ptr): INTEGER;

Description

TELSetConfig passes a null-terminated, C-style string to the telephone tool for parsing. The string, which can be of any length, is pointed to by thePtr and must contain tokens that describe the configuration of the telephone record. These tokens are defined by the tool; the string returned by TELGetConfig contains valid tokens.

TELSetConfig ignores items it does not recognize or find relevant; such an occurrence causes the telephone tool to stop parsing the string and to return the character position where the error occurred. If the telephone tool successfully parses the string, it returns noErr. If the telephone tool does not successfully parse the string, it returns one of the following values: a number less than -1 to indicate that an error occurred and no parsing was done, or a positive number to indicate the character position where parsing was stopped.

Individual telephone tools are responsible for the parsing operation.

Sample

A null-terminated, C-style configuration string

```
SLOT 9 OTHERFIELDS XXXX OTHERBOOLEANS TRUE\0
```

Result Codes

noErr, telBadTermErr, telUnknownErr

Opening, using, and closing the terminal

After your application has performed the required tasks described in the previous sections, it can open and use the terminal. When the terminal is open, your application can send commands through the Telephone Manager and telephone tool to the device drivers and terminal hardware. When the terminal is closed, your application cannot send it commands.

TELOpenTerm

Opening a terminal

`TELOpenTerm` attempts to open a terminal, based on information in a telephone record.

- △ **Important** Your application must call `TELOpenTerm` before calling any of the routines, except `TELTermMsgHand`, described in the rest of this chapter. Otherwise, all calls to terminal-related routines will fail. △

Function	<code>TELOpenTerm(hTEL: TELHandle): TELErr;</code>
Description	<code>TELOpenTerm</code> opens the driver associated with a specific tool and a specific terminal for the telephone record <code>hTEL</code>. In addition to opening the driver, <code>TELOpenTerm</code> finishes the initialization of the telephone record by assigning values to any of its fields that have not yet been filled in.
Result Codes	<code>noErr</code> , <code>telAlreadyOpen</code> , <code>telBadTermErr</code>

TELResetTerm

Resetting a terminal

`TELResetTerm` resets a terminal, based on information in a telephone record.

Function	<code>TELResetTerm(hTEL: TELHandle): TELErr;</code>
Description	<code>TELResetTerm</code> resets the terminal hardware and software associated with the telephone record <code>hTEL</code>, if the hardware and software have a reset capability.
Result Codes	<code>noErr</code> , <code>telBadTermErr</code> , <code>telStillNeeded</code>

TELCloseTerm

Closing a terminal

TELCloseTerm closes the terminal associated with the specified telephone record.

Function `TELCloseTerm(hTEL: TELHandle): TELErr;`

Description Your application should call `TELCloseTerm` when the Macintosh computer is being shut down or when the application is quitting. `TELCloseTerm` closes the terminal, if no other application has opened it.

hTEL specifies the telephone record associated with the terminal.

Result Codes `noErr, telBadTermErr, telStillNeeded`

TELDispose

Stopping the monitoring of a terminal

TELDispose cancels your application's monitoring of the terminal associated with a specified telephone record.

Note that `TELDispose` does not drop call appearances currently active on the terminal. To drop a call appearance, use the routine `TELDrop`.

Function `TELDispose(hTEL: TELHandle): TELErr;`

Description hTEL specifies the telephone record associated with the terminal.

TELDispose disposes of hTEL and all handles associated with directory numbers and call appearances on hTEL. In addition, TELDispose removes the terminal message handler for hTEL, and removes any directory-number messages handlers and call-appearance message handlers related to hTEL.

Result Codes `noErr, telBadTermErr`

TELIdle

Providing necessary idle time

Your application should call `TELIdle` at least once every time it goes through its main event loop, so that the connection tool can perform idle-loop tasks.

Procedure `TELIdle(hTEL: TELHandle);`

Description hTEL specifies the terminal for which idle-loop tasks are to be performed.

Getting information about a terminal

`TELGetInfo` returns state and capability information about a terminal/tool combination.

Function `TELGetInfo(hTEL: TELHandle) : TELErr;`

Description `hTEL` specifies the terminal for which information is requested.

The information that `TELGetInfo` returns is a “snapshot” of the current state of the telephone record. The Telephone Manager passes `hTEL` to the telephone tool. The tool inspects the `tRef` value in the structure and fills in the structure accordingly.

If `hTEL` has become invalid, the Telephone Manager returns `telBadTermErr`, and your application should call `TELDispose` to dispose of the invalid `hTEL`.

Result Codes `noErr`, `telBadTermErr`

Handling events

The Telephone Manager event-processing routines provide useful extensions to the Macintosh Toolbox Event Manager. This section explains the four routines that the Telephone Manager provides.

TELActivate

Activate events

TELActivate processes an activate or deactivate event (for instance, installing or removing a custom tool menu) for a window associated with the terminal.

Procedure

TELActivate(hTEL: TELHandle; activate: BOOLEAN);

Description

hTEL specifies the telephone record associated with the terminal.

If **activate** is **TRUE**, the telephone tool processes the activate event. Otherwise, it processes a deactivate event.

TELResume

Resume events

TELResume processes a resume or suspend event for a window associated with the terminal.

Procedure

TELResume(hTEL: TELHandle; resume: BOOLEAN);

Description

hTEL specifies the telephone record associated with the terminal.

If **resume** is **TRUE**, the telephone tool processes a resume event. Otherwise, it processes a suspend event.

TELMenu

Menu events

Your application must call **TELMenu** when the user chooses an item from a menu that is installed by the telephone tool.

Function

TELMenu(hTEL: TELHandle; menuID: INTEGER; item: INTEGER): BOOLEAN;

Description

hTEL specifies the telephone record associated with the terminal.

TELMenu returns **FALSE** if the telephone tool did not handle the menu event. **TELMenu** returns **TRUE** if the telephone tool did handle the menu event.

Other events

When your application receives an event, it should check whether the `refCon` of the window is a tool's `TELHandle`. Such an event occurs, for example, when the user clicks a button in a dialog box displayed by the telephone tool. If the event does belong to a telephone tool's window, your application can call `TELEvent`.

Procedure `TELEvent(hTEL: TELHandle; VAR theEvent: EventRecord);`

Description A window (or dialog box) created by a telephone tool has a telephone record handle stored in the `refCon` field for `windowRecord`.

`hTEL` specifies the telephone record associated with the terminal.

`theEvent` is a Macintosh system event, such as a mouse-down event.

Handling messages

The telephone tool for your application receives information from the telephone network switch whenever, for example, a call appears for a certain directory number. This information is sent as one or more telephone network events. The tool passes this information to the Telephone Manager which, in turn, passes it to your application as one or more messages.

△ **Important** Telephone network events relay information from a telephone network switch to a telephone tool. They are not to be confused with messages, which tools send to applications through the Telephone Manager, or with Macintosh system events. △

The Telephone Manager sends three main types of messages to applications: messages about the terminal as a whole, messages about particular directory numbers, and messages about particular call appearances. Your application must contain a message-handling routine (message handler) for each of the three main message types.

The Telephone Manager keeps a list of all message handlers for all applications. Before your message handlers can receive messages, your application must register them with the Telephone Manager and must specify the kinds of messages the handlers are to receive. To register message handlers, your application must call `TELTermMsgHand`, `TELDNMsgHand`, and `TELCAMsgHand`, passing pointers to your message handlers—referred to in this book as `MyTrmMsgHandler`, `MyDNMsgHandler`, and `MyCAMsgHandler`. (Refer to “Routines Your Application Must Provide” for information about writing `MyTrmMsgHandler`, `MyDNMsgHandler`, and `MyCAMsgHandler`.)

Each time a tool processes a telephone network event, the tool sends one or more messages to the Telephone Manager. The Telephone Manager then relays the message to all message handlers registered to receive that kind of message. An application can register more than one message handler of each type—for example, an application might register several directory-number message handlers. Each such handler could be registered to receive a different set of messages.

Registering a message handler for the terminal

TELTermMsgHand assigns a routine to handle messages from the terminal.

Function

```
TELTermMsgHand(hTEL: TELHandle; eventMask: LONGINT; msgProc: ProcPtr;
globals: LONGINT) : TELErr;
```

Description

TELTermMsgHand assigns the message handler `msgProc` to the telephone record `hTEL`. `eventMask` is a bit field that specifies which types of messages `msgProc` is to receive. The valid values for `eventMask` are as follows:

CONST

<code>telTermHookMsg</code>	=	<code>\$00000001;</code>
<code>telTermKeyMsg</code>	=	<code>\$00000002;</code>
<code>telTermVolMsg</code>	=	<code>\$00000004;</code>
<code>telTermDisplayMsg</code>	=	<code>\$00000008;</code>
 <code>telTermEnableMsg</code>	 =	 <code>\$00000010;</code>
<code>telTermOpenMsg</code>	=	<code>\$00000020;</code>
<code>termShutdownMsg</code>	=	<code>\$00000040;</code>
<code>telTermResetMsg</code>	=	<code>\$00000080;</code>
 <code>telTermErrorMsg</code>	 =	 <code>\$00000100;</code>
<code>telTermOtherMsg</code>	=	<code>\$00000200;</code>

{All other bits in `eventMask` are reserved by Apple for future use.}

Refer to Appendix B for descriptions of these and other message codes.

If a bit in `eventMask` is set to 1, messages of the corresponding type are sent to `msgProc`. If the bit is set to 0, they are not.

`msgProc` is a procedure pointer to your application's message handler for terminal-related messages.

`globals` is a pointer to a location in memory—for example, to your application's global variables (register A5). Each time procedure `msgProc` is called, `globals` is passed to it.

If, after registering this message handler, you wish to change `eventMask`, you must first clear the message handler by calling `TELClrTermMsgHand` and then register it again by calling `TELTermMsgHand`.

Result Codes

`noErr`, `telBadTermErr`

Registering a message handler for a directory number

`TELDNMsgHand` assigns a routine to handle messages for a particular directory number or, optionally, for all directory numbers.

Function

```
TELDNMsgHand(hTELDN: TELDNHandle; allDNS : BOOLEAN; eventMask:
LONGINT; msgProc: ProcPtr; globals: LONGINT) : TELErr;
```

Description

`TELDNMsgHand` assigns the message handler `msgProc` to the directory-number record `hTELDN`.

`allDNS` is a Boolean variable that, when equal to `TRUE`, shows that `msgProc` handles messages for all directory numbers assigned to the same terminal as `hTELDN`.

`eventMask` specifies which types of events `msgProc` is to receive. The valid values for `eventMask` are as follows:

```
CONST
    telDNForwardMsg      =    $01;
    telDNDNDMsg          =    $02;
    telDNVoiceMailMsg    =    $04;
    telDNSelectedMsg     =    $08;
    telDNOtherMsg        =    $8000;
{All other bits in eventMask are reserved by Apple for future use.}
```

Refer to Appendix B for descriptions of these and other message codes.

If a bit in `eventMask` is set to 1, messages of the corresponding type are sent to `msgProc`. If the bit is set to 0, they are not.

`msgProc` is a procedure pointer to your application's message handler for `hTELDN`.

`globals` is a pointer to a location in memory—for example, to your application's global variables (register A5). Each time procedure `msgProc` is called, `globals` is passed to it.

If after registering this message handler, you wish to change `eventMask`, you must first clear the message handler by calling `TELClrDNMsgHand` and then register it again by calling `TELDNMsgHand`.

Result Codes

`noErr`, `telBadDNErr`

Registering a message handler for call appearances

TELCAMsgHand assigns a routine to handle messages for all call appearances associated with the specified directory number.

Function TELCAMsgHand(hTELDN: TELDNHandle; eventMask: LONGINT; msgProc: ProcPtr; globals: LONGINT) : TELErr;

Description TELCAMsgHand assigns the message handler msgProc to all calls associated with hTELDN.

hTELDN is a handle to a directory-number record.

eventMask specifies which types of events msgProc is to receive. The valid values for eventMask are as follows:

CONST

telCAAlertingMsg	=	\$00000001;
telCAOfferMsg	=	\$00000002;
telCAProgressMsg	=	\$00000004;
telCAOutgoingMsg	=	\$00000008;
telCADisconnectMsg	=	\$00000010;
telCAActiveMsg	=	\$00000020;
telCAConferenceMsg	=	\$00000040;
telCATransferMsg	=	\$00000080;
telCAHoldMsg	=	\$00000100;
telCADigitsMsg	=	\$00000200;
telCACallParkMsg	=	\$00000400;
telCACallbackMsg	=	\$00000800;
telCAREjectMsg	=	\$00001000;
CADeflectedMs	=	\$00002000;
telCAForwardMsg	=	\$00004000;
telCAConferenceSplitMsg	=	\$00008000;
telCAConferenceDropMsg	=	\$00010000;
telCAQueuedMsg	=	\$00020000;
telCAInUseMsg	=	\$00040000;
telCACallPickupMsg	=	\$00080000;
telCAPagingMsg	=	\$00100000;
telCAIntercomMsg	=	\$00200000;
telCAModemToneMsg	=	\$00400000;
telCAFaxToneMsg	=	\$00800000;
telCAIdleMsg	=	\$01000000;
telCASuccessiveAlertMsg	=	\$02000000;
CAUserInfoMsg	=	\$04000000;

{All other bits in eventMask are reserved by Apple for future use.}

Refer to Appendix B for descriptions of these and other message codes.

If a bit in `eventMask` is set to 1, messages of the corresponding type are sent to `msgProc`. If the bit is set to 0, they are not.

`msgProc` is a procedure pointer to your application's message handler for call appearances associated with `hTELDN`.

`globals` is a pointer to a location in memory—for example, to your application's global variables (register A5). Each time procedure `msgProc` is called, `globals` is passed to it.

If after registering this message handler, you wish to change `eventMask`, you must first clear the message handler by calling `TELClrCAMsgHand` and then register it again by calling `TELCAMsgHand`.

Result Codes `noErr`, `telBadCAErr`

TELClrTermMsgHand

Clearing a terminal message handler

`TELClrTermMsgHand` removes a terminal message handler from the handler list.

Function `TELClrTermMsgHand(hTEL: TELHandle; msgProc: ProcPtr): TELErr;`

Description `hTEL` is a handle to a telephone record for a terminal.
`msgProc` is a procedure pointer to the terminal message handler for `hTEL`.

Result Codes `noErr`, `telBadProcErr`, `telBadTermErr`

TELClrDNMsgHand

Clearing a directory-number message handler

`TELClrDNMsgHand` removes a directory-number message handler from the handler list.

Function `TELClrDNMsgHand(hTELDN: TELDNHandle; msgProc: ProcPtr): TELErr;`

Description `hTELDN` is a handle to a directory-number record.
`msgProc` is a procedure pointer to the directory-number message handler for `hTELDN`.

Result Codes `noErr`, `telBadDNErr`, `telBadProcErr`, `telBadTermErr`

TELClrCMsgHand

Clearing a call-appearance message handler

TELClrCMsgHand removes a call-appearance message handler from the handler list.

Function TELClrCMsgHand(hTELDN: TELDNHandle; msgProc:ProcPtr): TELErr;

Description hTELDN is a handle to a directory-number record.
msgProc is a procedure pointer to the call-appearance message handler for hTELDN.

Result Codes noErr, telBadCAErr, telBadProcErr

TELTermEventsSupp

Finding supported terminal messages

TELTermEventsSupp returns a mask indicating which terminal messages a telephone tool supports.

Function TELTermEventsSupp(hTEL : TELHandle; VAR eventMask : LONGINT): TELErr;

Description hTEL is a handle to a telephone record for a terminal.
eventMask is a mask that indicates which terminal messages are supported by the telephone tool for hTEL.

Result Codes noErr, telBadTermErr

TELDNEventsSupp

Finding supported directory-number messages

TELDNEventsSupp returns a mask indicating which directory-number messages a telephone tool supports.

Function TELDNEventsSupp(hTELDN : TELDNHandle; VAR eventMask : LONGINT):
TELErr;

Description hTELDN is a handle to a directory-number record.
eventMask is a mask that indicates which directory-number messages are supported by the telephone tool for hTELDN.

Result Codes noErr, telBadDNErr

TELCAEventsSupp

Finding supported call-appearance messages

TELCAEventsSupp returns a mask indicating which call-appearance messages a telephone tool supports.

Function	TELCAEventsSupp(hTELDN : TELDNHandle; VAR eventMask : LONGINT): TELErr;
Description	hTELDN is a handle to a directory-number record. eventMask is a mask that indicates which call-appearance messages are supported by the telephone tool for hTELDN.
Result Codes	noErr, telBadCAErr

Placing and receiving calls

A typical telephone provides at least three services: placing calls, receiving calls, and releasing calls. Your application can place and receive calls by using the Telephone Manager routines described in this section. To release calls, your application can call the Telephone Manager routine `TELDrop`, described in the section "Using Drop, Hold, Transfer, Forward, and Conference," later in this chapter.

To place a call, your application must first prepare to place the call by calling `TELSetupCall`. If the number to be dialed is incomplete, your application can then complete the number and dial it by calling `TELDialDigits`. If the number to be dialed is complete, your application can dial it and establish the call by calling `TELConnect`.

Your application can accept an incoming call by calling `TELAcceptCall` or `TELAnswerCall`, reject the call by calling `TELRejectCall`, or deflect the call (as when transferring) by calling `TELDeflectCall`.

TELSetupCall

Setting up a call

`TELSetupCall` prepares to place an outbound call, either directly or for use with a feature such as Conference or Transfer. (For information on allocating directory-number records, refer to the section "Controlling Directory Numbers" later in this chapter.)

Function

```
TELSetupCall(hTELDN: TELDNHandle; VAR hTELCA: TELCAHandle; destDN,  
destName, destSubaddr, userUserInfo: Str255; bearerType, rate:  
INTEGER;): TELErr;
```

Description

`TELSetupCall` allocates an available call-appearance record for the directory number `hTELDN`, and returns it in `hTELCA`.

`hTELDN` is a handle to a directory-number record.

`hTELCA` is a handle to a call-appearance record on the directory number specified by `hTELDN`.

`destDN` specifies the destination phone number or network address. If `destDN` is specified completely, the call can be placed with `TELConnect`. But if `destDN` is `NIL` or is incomplete, the call cannot be placed until network address characters are completely sent with `TELDialDigits`. A call setup with `TELSetupCall` could, for example, emit a dial tone through the speaker either until `TELConnect` is called or until address characters are given with `TELDialDigits`.

`destName` is the name of the party associated with the `destDN`.

`destSubaddr` specifies a subaddress, as defined by the ISDN S-Bus. Use this parameter only if the network supports subaddressing. To find out whether the network supports subaddressing, call the routine `TELGetInfo` and check the field `hasSubaddress` in the telephone record. If the network does not support subaddressing, `destSubaddr` is ignored.

`userUserInfo` is user-to-user information, as defined by some telephone network switches. Use this parameter only on a network that supports user-to-user information. To find out whether the network supports user-to-user information, call the routine `TELGetInfo` to check the field `hasUserUserInfo` in the telephone record. If the network does not support user-to-user information, the information is ignored.

`bearerType` and `rate` indicate the type of call. Your application should set `bearerType` to 0 for voice calls and 1 for other calls. `rate` should be set to zero, which makes the tool use the default rate; all other values are reserved for use by Apple and should be considered positive.

Result Codes `noErr`, `telBadDNErr`, `telCAUnavail`

TELDialDigits

Dialing a call

`TELDialDigits` dials a string of network address characters for a call appearance.

Function `TELDialDigits(hTELCA: TELCAHandle; digits: Str255): TELErr;`

Description `hTELCA` is a handle to a call appearance set up by the routine `TELSetupCall`.
`digits` is a string of characters to be dialed.

`TELDialDigits` dials `digits` for `hTELCA`. When the telephone tool has received enough characters to complete a telephone call, it sends call-progress messages to the call-appearance message handler for `hTELCA`.

Result Codes `noErr`, `telBadCAErr`, `telBadDNErr`, `telBadTermErr`

TELConnect

Connecting a call

`TELConnect` establishes a connection for a call appearance.

Function `TELConnect(hTELCA: TELCAHandle): TELErr;`

Description `hTELCA` is a handle to a call appearance set up by the routine `TELSetupCall`.

Result Codes `noErr`, `telBadCAErr`

TELAcceptCall

Accepting a call

TELAcceptCall accepts a call appearance whose state is telCAOfferState.

Function TELAcceptCall(hTELCA: TELCAHandle): TELErr;

Description hTELCA is a handle to an incoming call appearance.

TELAcceptCall accepts call appearance hTELCA, which must be both acceptable and in the state telCAOfferState. (When the state of a call appearance is telCAOfferState, your application's call-appearance message handler receives a telCAOfferMsg message from the telephone tool. Whether the call is acceptable depends on the value of the acceptable field in the call-appearance record hTELCA.)

After a call appearance is accepted, its state changes to telCAAlertingState. It can then be answered with the routine TELAnswerCall.

Result Codes noErr, telBadCAErr, telCANotAcceptable, telFeatNotAvail, telFeatNotSub, telFeatNotSupp

TELAnswerCall

Answering a call

TELAnswerCall answers an incoming call appearance.

Function TELAnswerCall(hTELCA: TELCAHandle): TELErr;

Description hTELCA is a handle to a incoming call appearance.

TELAnswerCall answers the incoming call appearance hTELCA, whose state can be telCAOfferState or telCAAlertingState. When the call appearance is answered, your application receives a telCAActiveMsg message.

After an incoming call appearance is answered, its state changes to telCAActiveState and conversation can take place.

Result Codes noErr, telBadCAErr, telFeatNotAvail, telFeatNotSub, telFeatNotSupp

TELRejectCall

Rejecting a call

TELRejectCall rejects an incoming call appearance.

Function

TELReject(hTELCA: TELCAHandle; reason: INTEGER): TELErr;

Description

hTELCA is a handle to an incoming call appearance.

reason is reserved by Apple for future use.

TELRejectCall rejects call appearance hTELCA, whose state must be either telCAOfferState or telCAAlertingState. (When the state of a call appearance is telCAOfferState or telCAAlertingState, your application's call-appearance message handler receives a telCAOfferMsg or telCAAlertingMsg message from the telephone tool. Whether the call is rejectable depends on the value of the rejectable field in the call-appearance record hTELCA.)

After an incoming call appearance is rejected, its state changes to telCAIdleState, meaning that it can be disposed of by means of the routine TELDispose.

Result Codes

noErr, telBadCAErr, telCANotRejectable, telFeatNotAvail, telFeatNotSub, telFeatNotSupp

Deflecting a call

`TELDeflectCall` deflects an incoming call appearance, sending it to a remote directory number.

Function

`TELDeflect(hTELCA: TELCAHandle; rmtDN, rmtName, rmtSubaddress: Str255): TELErr;`

Description

`hTELCA` is a handle to an incoming call appearance.

`rmtDN` is the directory number to which the call is to be deflected.

`rmtName` is the name of the party associated with directory number `rmtDN`.

`rmtSubaddress` is a subaddress, as defined by the ISDN S-Bus.

`TELDeflectCall` deflects call appearance `hTELCA`, whose state must be either `telCAOfferState` or `telCAAlertingState`. (When the state of a call appearance is `telCAOfferState` or `telCAAlertingState`, your application's call-appearance message handler receives a `telCAOfferMsg` or `telCAAlertingMsg` message from the telephone tool. Whether the call is deflectable depends on the value of the field `deflectable` in the call-appearance record `hTELCA`.)

After an incoming call appearance is deflected, its state changes to `telCAIdleState`, meaning that it can be disposed of by means of the routine `TELDispose`.

Result Codes

`noErr`, `telBadCAErr`, `telCANotDeflectable`, `telFeatNotAvail`, `telFeatNotSub`, `telFeatNotSupp`

Using Drop, Hold, Transfer, Forward, and Conference

Many telephone users subscribe to the supplementary features Drop, Hold, Transfer, Forward, and Conference. If the user's terminal supports these features, your application can offer them by calling Telephone Manager routines. The routines described in this section let your application offer Drop, Hold, Transfer, Forward, and Conference. Routines for offering less-common supplementary features are described later in this chapter.

Not all supplementary features can be applied to all calls. For instance, if the telephone tool your application is using does not support the Conference feature, calls cannot be conferenced. Or if a given call is currently on hold, the Hold feature cannot be applied to it again.

The call-appearance record contains, for each supplementary feature, a set of flags showing whether the feature can be applied to that particular call. The names of these flags are typically `xxxSub`, `xxxAvail`, and `xxxActive`, where `xxx` is the name of the feature. For instance, the flags `holdSub`, `holdAvail`, and `holdActive` show whether the Hold feature is subscribed, whether it is available, and whether it is active. A feature is "subscribed" if the user's network line provides it and if the terminal supports it. A feature is "available" if the call appearance involved is in the state needed to use the feature. A feature is "active" if it is currently applied to the call appearance.

Here is an example, using the Hold feature, of how the `xxxSub`, `xxxAvail`, and `xxxActive` flags work. If the telephone record indicates that the user's terminal can put calls on hold, the Telephone Manager sets `holdSub` in each call-appearance record it allocates for the terminal. If an outgoing call can be held only after its call-appearance state is `telCAActiveState`, the Telephone Manager sets `holdAvail` only when the call reaches that state. Later, when the user puts the call on hold, the Telephone Manager sets the flag `holdActive`.

TELDrop

Dropping calls

`TELDrop` drops an incoming call.

Function

```
TELDrop(hTELCA: TELCAHandle; userUserInfo: Str255): TELErr;
```

△ **Important** `TELDrop` and `TELDNSelect` are the only Telephone Manager routines that drop calls. Routines that dispose of handles to Telephone Manager records do not drop the associated telephone call. △

Description

`hTELCA` is a handle to an incoming call appearance.

`userUserInfo` stores user-to-user information, as defined by some telephone network switches. Use this parameter only on a network that supports user-to-user information. To find out whether the network supports user-to-user information, call the routine `TELGetInfo` to check the field `hasUserUserInfo` in the telephone record. If the network does not support user-to-user information, the user-to-user information is ignored.

`TELDrop` drops the call appearance `hTELCA`, which can be in any state.

After a call appearance is dropped, its state changes to `telCAIdleState`.

Result Codes

`noErr`, `telBadCAErr`, `telFeatNotAvail`, `telFeatNotSub`, `telFeatNotSupp`

TELHold

Putting calls on hold

`TELHold` puts a call on hold.

Function	<code>TELHold(hTELCA: TELCAHandle): TELErr;</code>
Description	<code>hTELCA</code> is a handle to a call-appearance record. In that record, the <code>holdAvail</code> bit of the <code>FeatureFlag</code> field must be set. Otherwise, <code>TELHold</code> fails.
Result Codes	<code>noErr</code> , <code>telBadCAErr</code> , <code>telFeatActive</code> , <code>telFeatNotAvail</code> , <code>telFeatNotSub</code> , <code>telFeatNotSupp</code>

TELRetrieve

Retrieving held calls

`TELRetrieve` retrieves a held call.

Function	<code>TELRetrieve(hTELCA: TELCAHandle): TELErr;</code>
Description	<code>hTELCA</code> is a handle to a call-appearance record. In that record, the <code>holdActive</code> bit of the <code>FeatureFlag</code> field must be set. Otherwise, <code>TELRetrieve</code> fails.
Result Codes	<code>noErr</code> , <code>telBadCAErr</code> , <code>telFeatActive</code> , <code>telFeatNotAvail</code> , <code>telFeatNotSub</code> , <code>telFeatNotSupp</code>

TELTransferPrep

Preparing for a consult transfer

`TELTransferPrep` prepares a call for a consult transfer, in which the user consults the destination party before transferring the call. (For information on blind transferring, refer to the description of `TELTransferBlind`, later in this section.)

Function	<code>TELTransferPrep(hTELCA1, hTELCA2: TELCAHandle): TELErr;</code>
Description	<code>hTELCA1</code> is a handle to a call appearance whose state is either <code>telCAActiveState</code> or <code>telCAHeldState</code>. <code>hTELCA2</code> is a handle to a second call appearance, set up by <code>TELSetupCall</code> but not yet connected. Before calling <code>TELTransferPrep</code> your application must call <code>TELSetupCall</code> to set up <code>hTELCA2</code>. When <code>hTELCA2</code> becomes active, the user can, optionally, consult the destination party. Your application can then call <code>TELTransferEstablish</code> to transfer the call.

`TELTransferPrep` attempts to establish a connection for `hTELCA2`, and works much like `TELConnect`.

Result Codes `noErr`, `telBadCAErr`, `telBadDNErr`, `telFeatActive`, `telFeatNotAvail`,
`telFeatNotSub`, `telFeatNotSupp`

TELTransferEstablish

Completing a consult transfer

`TELTransferEstablish` completes the consult transfer of call previously set up with `TELTransferPrep`.

Function `TELTransferEstablish(hTELCA1, hTELCA2: TELCAHandle): TELErr;`

Description `hTELCA1` is a handle to a call appearance for the transferee, specified as `hTELCA1` in `TELTransferPrep`.

`hTELCA2` is the call appearance to which `hTELCA1` is to be transferred, specified as `hTELCA2` in `TELTransferPrep`.

Result Codes `noErr`, `telBadCAErr`, `telFeatActive`, `telFeatNotAvail`, `telFeatNotSub`,
`telFeatNotSupp`, `telTransRej`

TELTransferBlind

Blind-transferring a call

`TELTransferBlind` transfers a call without first letting the user consult the destination party. (For information on consult transferring, refer to the description of `TELTransferPrep`, earlier in this section.)

Function `TELTransferBlind(hTELCA1: TELCAHandle; rmtDN, rmtName, rmtSubaddress: Str255): TELErr;`

Description `hTELCA1` specifies the call appearance to be transferred.

`rmtDN` specifies the remote directory number to which the call will be transferred.

`rmtName` specifies the name of party whose telephone number is `rmtDN`.

`rmtSubaddress` specifies the network subaddress, if any, associated with `rmtDN`.

Result Codes `noErr`, `telBadCAErr`, `telBadDNErr`, `telFeatActive`, `telFeatNotAvail`,
`telFeatNotSub`, `telFeatNotSupp`, `telTransferReject`

TELForwardSet

Forwarding calls

TELForwardSet causes calls for one directory number to be forwarded to another.

Function `TELForwardSet(hTELDN: TELDNHandle; forwardDN, forwardPartyName, forwardSubaddr: Str255; forwardType, numRings: INTEGER): TELErr;`

Description `hTELDN` is a handle to the directory number whose calls are to be forwarded.

`forwardDN` is the directory number that will receive the forwarded calls.

`forwardPartyName` is the name of the party whose directory number is `hTELDN`.

`forwardSubaddr` is a subaddress, as defined by the ISDN S-Bus. Use this parameter only if the network supports subaddressing. (To find out whether the network supports subaddressing, check the bit `hasSubaddress` in the field `featureFlags` of the telephone record.) If the network does not support subaddressing, the tool ignores `forwardSubaddr`.

`forwardType` specifies the kind of forwarding being requested, and can have the following valid values:

```
CONST
    telForwardImmediate = 1; {forward all calls}
    telForwardBusy      = 2; {forward if busy}
    telForwardNoAnswer  = 3; {forward if no answer}
    telForwardBusyNA    = 4; {forward if busy or no answer}
```

`numRings` specifies how many rings occur before calls are forwarded to `forwardDN`.

Result Codes `noErr`, `telBadDNerr`, `telBadFwdType`, `telFeatActive`, `telFeatNotAvail`, `telFeatNotSub`, `telFeatNotSupp`, `telFwdTypeNotSupp`

TELForwardClear

Clearing call forwarding

TELForwardClear clears the call forwarding performed by **TELForwardSet**.

Function `TELForwardClear(hTELDN: TELDNHandle; forwardType : INTEGER): TELErr;`

Description `hTELDN` is a handle to the directory number whose calls are being forwarded.

`forwardType` specifies the kind of forwarding being cleared, as specified in **TELForwardSet**.

Result Codes `noErr`, `telBadDNerr`, `telBadFwdType`, `telFeatActive`, `telFeatNotAvail`, `telFeatNotSub`, `telFeatNotSupp`

Preparing for conferencing

TELConferencePrep prepares for one call to be conferenced with another by the routine TELConferenceEstablish (described later in this section). Your application must call TELConferencePrep each time a party is to be added to the conference.

Function

TELConferencePrep(hTELCA1, hTELCA2: TELCAHandle; numToConference :
INTEGER) : TELErr;

Description

hTELCA1 is a handle to the conference initiator—an active or held call appearance whose caRelated bit is set. All other call appearances conferenced with this one will reference hTELCA1 in the relatedCA field of their call-appearance records.

hTELCA2 is a handle to a call appearance set up by TELSetupCall but not yet connected. In call-appearance record hTELCA2, the field relatedCA references hTELCA1.

When call appearance hTELCA2 becomes active, the user of your application can consult the party associated with that call appearance. Your application can then call TELConferenceEstablish to establish a three-way conference. Or, if the user does not want to consult, your application can call TELConferenceEstablish immediately after call appearance hTELCA2 becomes active.

numToConference specifies how many calls, including the conference initiator, will be in this conference.

If the numToConferenceRequired bit is set in call-appearance record hTELCA1, your application must supply a value for numToConference. (If numToConference has a maximum value, it is specified by the field confLimit in the call-appearance record.) If the numToConferenceRequired bit is not set, numToConference can be left blank; the tool will ignore it.

TELConferencePrep attempts to establish a connection for hTELCA2, unless there is one already, and works much like the routine TELConnect.

Result Codes

noErr, telBadCAErr, telBadDNErr, telConfLimitErr, telConfNoLimit,
telConfLimitExceeded, telConfRej, telFeatActive, telFeatNotAvail,
telFeatNotSub, telFeatNotSupp

TELConferenceEstablish

Establishing a conference

TELConferenceEstablish conferences one call with another, and should be called only after you have prepared the conference with the routine TELConferencePrep.

Function	TELConferenceEstablish(hTELCA1, hTELCA2: TELCAHandle) : TELErr;
Description	hTELCA1 is a handle to the conference initiator—an active or held call that initiated the conference. hTELCA2 is a handle to another call-appearance record, whose caRelated field references hTELCA1.
Result Codes	noErr, telBadCAErr, telConfLimitErr, telBadDNErr, telConfLimitExceeded, telConfRej, telFeatActive, telFeatNotAvail, telFeatNotSub, telFeatNotSupp

TELConferenceSplit

Splitting a conference

TELConferenceSplit splits a call from a conference established by TELConferenceEstablish.

Function	TELConferenceSplit(hTELCA : TELCAHandle) : TELErr;
Description	hTELCA is a handle to a call appearance to be split from the conference. The call can be split from the conference only if, in call-appearance record hTELCA, the conferenceSplitAvail flag of the featureFlags field is set. After a split, the rest of the conference remains intact.
Result Codes	noErr, telBadCAErr, telConfRej, telFeatActive, telFeatNotAvail, telFeatNotSub, telFeatNotSupp

Using less-common supplementary features

This section describes routines for providing less-common supplementary features: Call Back, Do Not Disturb, Call Pickup, Call Park, Voice Mail, and Paging.

TELCallbackSet

Requesting a callback

`TELCallbackSet` requests a callback, which notifies the user's terminal when a destination number that was previously unavailable (busy or unanswered) becomes available.

◆ *Note:* Some switch vendors refer to the Call Back feature as the Ring Again feature.

Function	<code>TELCallbackSet(hTELCA: TELCAHandle; VAR callbackRef : INTEGER): TELErr;</code>
Description	<code>hTELCA</code> is a handle to the currently unavailable call appearance. <code>callbackRef</code> is an identifier that the tool provides to distinguish this callback from any others the user has requested. If <code>callbackRef</code> equals zero, your application should ignore it. When the destination number becomes available, <code>TELCallbackSet</code> causes your application's call-appearance message handler to receive a <code>telCACallbackMsg</code> message, with a value of <code>telCallbackNowAvail</code>. Some network systems require that all Call Back On No Answer requests be issued by the remote party (the person who did not answer). Usually, in such a system, the remote party will have received a <code>telCallbackDesired</code> message. The remote user can then call <code>TELCallbackNow</code>, passing <code>callbackRef</code> values that were in the message.
Result Codes	<code>noErr, telBadCAErr, telBadCBErr, telFeatActive, telFeatNotAvail, telFeatNotSub, telFeatNotSupp</code>

TELCallbackNow

Calling back

`TELCallbackNow` calls back a previously unavailable destination number, if your application has requested the callback through `TELCallbackSet`.

Function	<code>TELCallbackNow(hTELCA : TELCAHandle; callbackRef : INTEGER): TELErr;</code>
Description	<code>hTELCA</code> is a handle to a newly allocated call appearance. This handle will be used for the callback. <code>callbackRef</code> must have the same value as the <code>callbackRef</code> parameter returned from <code>TELCallbackSet</code>.
Result Codes	<code>noErr, telBadCAErr, telBadCBErr, telFeatActive, telFeatNotAvail, telFeatNotSub, telNoCallbackRef, telFeatNotSupp</code>

TELCallbackClear

Clearing pending callbacks

`TELCallbackClear` clears a callback requested through `TELCallbackSet`.

Function	<code>TELCallbackClear(hTEL : TELHandle; callbackRef : INTEGER): TELErr;</code>
Description	<code>hTEL</code> is a handle to the telephone record for the user's terminal. <code>callbackRef</code> must have the same value as the <code>callbackRef</code> parameter returned from <code>TELCallbackSet</code>.
Result Codes	<code>noErr, telBadCBErr, telBadTermErr, telFeatActive, telFeatNotAvail, telFeatNotSub, telFeatNotSupp, telNoCallbackRef</code>

TELDNDSet

Setting Do Not Disturb

`TELDNDSet` sets the Do Not Disturb feature (as implemented by the telephone network switch) on a specified directory number.

Function	<code>TELDNDSet(hTELDN: TELDNHandle; dndType : INTEGER): TELErr;</code>
Description	<code>hTELDN</code> is a handle to the directory number for which Do Not Disturb is being set. <code>dndType</code> specifies the kind of Do Not Disturb, and can have the following valid values:

```

CONST
    telDNDIntExt      = 0;
                        {do not disturb on internal and external calls}
    telDNDExternal    = 1;
                        {do not disturb on external calls only}
    telDNDInternal    = 2;
                        {do not disturb on internal calls only}
    telDNDNonIntercom = 3;
                        {do not disturb on all calls except intercom}

```

Your application can provide more complex Do Not Disturb capabilities than those of `TELDNDSet`. To do so, inspect the alerting pattern on each incoming call, rejecting unwanted calls with the routine `TELReject`.

Result Codes `noErr`, `telBadDNDType`, `telBadDNErr`, `telDNDTypeNotSupp`, `telFeatActive`,
 `telFeatNotAvail`, `telFeatNotSub`, `telFeatNotSupp`

TELDNDClear

Clearing Do Not Disturb

`TELDNDClear` clears the Do Not Disturb feature on a directory number, as implemented by the telephone network switch.

Function `TELDNDClear(hTELDN: TELDNHandle; dndType : INTEGER): TELErr;`

Description `hTELDN` is a handle to the directory number for which Do Not Disturb is being cleared.
 `dndType` specifies the kind of Do Not Disturb being cleared.

Result Codes `noErr`, `telBadDNDType`, `telBadDNErr`, `telDNDTypeNotSupp`, `telFeatActive`,
 `telFeatNotAvail`, `telFeatNotSub`, `telFeatNotSupp`

TELCallPickup

Picking up calls

`TELCallPickup` lets the user's terminal answer a call that is alerting at another terminal.

Function

```
TELCallPickup(hTELCA: TELCAHandle; pickupDN : StringPtr;  
pickupGroupID : INTEGER): TELErr;
```

Description

`hTELCA` is a handle to a newly allocated call appearance used to pick up the call.

`pickupDN` specifies the remote directory number where the call is alerting. If `pickupDN` is `NIL`, the user can pick up an alerting call from any directory number within a pickup group predefined by the telephone network switch.

`pickupGroupID` specifies the pickup group to which `pickupDN` belongs. Your application must pass this value if the telephone network switch requires a pickup ID for Call Pickup requests. (To find out whether the network switch requires a pickup ID, check the `numPickupIDs` field of the directory-number record associated with `hTELCA`. If the value of this field is greater than zero, your system requires a pickup ID.)

When your application calls `TELCallPickup`, the call-appearance message handler receives a `telCAActiveMsg` message, as when a normal alerting call is answered.

Result Codes

`noErr`, `telBadDNErr`, `telBadPickupGroupID`, `telFeatActive`, `telFeatNotAvail`, `telFeatNotSub`, `telFeatNotSupp`

TELParkCall

Parking calls

`TELParkCall` parks a call, making it available for retrieval at another directory number.

Function

```
TELParkCall(hTELCA: TELCAHandle; VAR parkRetrieveID : Str255; parkID  
: Str255) : TELErr;
```

Description

`hTELCA` specifies the call appearance being parked.

`parkRetrieveID` specifies an identifier that the user must enter at another directory number to retrieve call appearance `hTELCA`. Your application is responsible for disposing of the string referenced by `parkRetrieveID`.

`parkID` specifies an ID, such as directory number or code, that the user enters to park the call.

When parking calls, some systems return a `parkRetrieveID`, by which the call can be retrieved at any directory number in the system. In contrast, other systems require that the user retrieve the call with a specific ID, `parkID`. Still other systems require both kinds of identifiers. To find out which method your system uses, check the field `featureFlags` in call-appearance record `hTELCA`. If the `parkRetrieveWithID` flag is set, the system returns a `parkRetrieveID`; if the `parkWithID` flag is set, the system requires a `parkID`. If `parkRetrieveWithID` and `parkWithID` are both set, the system requires both a `parkID` and a `parkRetrieveID`.

Result Codes `noErr`, `telBadCAErr`, `telBadParkID`, `telFeatActive`, `telFeatNotAvail`,
`telFeatNotSub`, `telFeatNotSupp`

TELRetrieveParkedCall

Retrieving parked calls

`TELRetrieveParkedCall` retrieves a parked call.

Function `TELRetrieveParkedCall(hTELCA: TELCAHandle; parkRetrieveID : Str255) :`
`TELErr;`

Description `hTELCA` specifies a newly allocated call appearance on which the call is being retrieved.

`parkRetrieveID` is an identifier required by systems that return a park-retrieve ID when parking calls.

When your application calls `TELRetrieveParkedCall`, the call-appearance message handler receives the message `telCAActiveMsg`, as when answering a normal alerting call.

Result Codes `noErr`, `telBadCAErr`, `telBadParkID`, `telFeatActive`, `telFeatNotAvail`,
`telFeatNotSub`, `telFeatNotSupp`

Accessing voice-mail systems

`TELVoiceMailAccess` accesses the voice-mail system of the user's private branch exchange (PBX).

Function `TELVoiceMailAccess(hTELCA: TELCAHandle) : TELErr;`

Description `hTELCA` specifies a newly allocated call appearance on which the voice-mail system will be accessed.

`TELVoiceMailAccess` requests access to a voice-mail system by passing to it `hTELCA`, a handle to the call appearance on which the voice-mail system will be accessed. Your application can obtain `hTELCA` either from the routine `TELSetupCall` or from the message `telCAOutgoingMsg`.

`TELVoiceMailAccess` does not provide voice-mail management; it provides only the ability to access a voice-mail system. Your application might call `TELVoiceMailAccess` after, for example, receiving the message `telDNVoiceMailMsg` or when the user wants to check saved voice-mail messages.

Result Codes `noErr, telBadDNErr, telFeatActive, telFeatNotAvail, telFeatNotSub, telFeatNotSupp`

TELPaging

Accessing paging equipment

`TELPaging` accesses a specified paging function predefined by the network system.

Function `TELPaging(hTELCA: TELCAHandle; pageID : INTEGER) : TELErr;`

Description `hTELCA` specifies a newly allocated call appearance to be used for the page.

`pageID` is an identifier required if the user's telephone has multiple paging features. To find out whether your system requires a `pageID`, check the `NumPageID` field of directory-number record associated with `hTELCA`. If the value of this field is greater than zero, your system requires a `pageID`. Note that if `NumPageID` equals, for example, 5, then `pageID` must have a value between 1 and 5—not between 0 and 4.

If your telephone network switch provides a special telephone number for paging, your application should not use `TELPaging`. Instead, it should use the routines `TELSetup` and `TELConnect`. In other words, on systems where users initiate paging by pressing a dedicated "paging button", your application should invoke `TELPaging`. But on systems where users initiate paging by placing calls to a dedicated extension—such as 3460, which might correspond to an overhead loudspeaker—your application should invoke `TELSetup` and `TELConnect` to place a call to extension 3460.

Result Codes noErr, telBadDNErr, telBadPageID, telFeatActive, telFeatNotAvail,
telFeatNotSub, telFeatNotSupp

TELIntercom

Using intercom

TELIntercom accesses a specified intercom function predefined by the network system.

Function TELIntercom(hTELCA: TELCAHandle; intercomID : INTEGER) : TELerr;

Description hTELCA specifies a newly allocated call appearance to be used for the intercom function. If a system does not require a call appearance when placing intercom calls, hTELCA equals zero.

intercomID is an identifier required if the user's telephone has multiple intercom features. To find out whether your system requires an intercomID, check the numIntercomIDs field of the directory-number record associated with hTELCA. This field tells how many intercom "keys" the terminal supports. If its value is greater than zero, your system requires an intercomID. Note that if numIntercomIDs equals, for example, 5, then intercomID must have a value between 1 and 5—not between 0 and 4.

Result Codes noErr, telBadDNErr, telBadIntercomID, telFeatActive, telFeatNotAvail,
telFeatNotSub, telFeatNotSupp

Accessing special features of switches and tools

This section describes routines that let your application access features for which there is no specific Telephone Manager routine.

If your application is meant to be tool-independent, you can use `TELOtherFeatureList` to list network-switch features that do not require specific parameter values. Your application can then display items from this list as user options. To access any of the listed features, use `TELOtherFeatureImplement`.

If your application is tool-specific and can send a parameter block in the format the tool requires, `TELOtherFunction` lets you access tool-specific features.

TELOtherFeatureList

Listing special switch features

`TELOtherFeatureList` lists the network-switch features that your application can access without passing specific parameters.

Function `TELOtherFeatureList(hTEL: TELHandle; VAR fList : FeatureListPtr): TELErr;`

Description `hTEL` is a handle to a telephone record.

`FeatList` is a pointer to a linked list, each item of which describes a feature and its attributes. The list has the following format:

```
TYPE
  FeatureListPtr    =    ^FeatureList
  FeatureList
    featureID       :    INTEGER;
    featureName     :    StringPtr;
    handleType      :    INTEGER;
                    {values are: 0 = telHandleType
                               1 = telDNHandleType
                               2 = telCAHandleType}
    nextFeature     :    FeatureListPtr;
END
```

In each item of `fList`, `featureID` is a tool-defined value that identifies the feature, and `featureName` points to the name of the feature. `handleType` specifies the kind of record for which your application must pass a handle when accessing the feature through `TELOtherFeatureImplement`. If `handleType` is zero (0), your application must pass a handle to a telephone record; if 1, a handle to a directory-number record; or if 2, a handle to a call-appearance record. Your application is not responsible for disposing of the list to which `fList` points.

`nextFeature` points to the next item in the list of features, or is `NIL` if there are no more items.

Result Codes `noErr, telBadTermErr`

TELOtherFeatureImplement

Implementing special switch features

`TELOtherFeatureImplement` accesses a network-switch feature returned by `TELOtherFeatureList`.

Function `TELOtherFeatureImplement(hTEL : TELHandle; theHandle: Handle;
featureID : INTEGER): TELErr;`

Description `hTEL` is a handle to a telephone record.

`theHandle` is a handle to a telephone record, directory-number record, or call-appearance record, as specified by `TELOtherFeatureList`.

`featureID` is the feature ID returned by `TELOtherFeatureList`.

Result Codes `noErr, telBadCAErr, telBadDNerr, telBadFeatID, telBadHandErr,
telBadTermErr`

TELOtherFunction

Implementing tool-specific features

`TELOtherFunction` accesses a tool-specific feature by passing a parameter block defined by the tool.

△ **Important** For information about third-party tool features, consult the developer of the tool. Such features are neither supported nor documented by Apple Computer. △

Function `TELOtherFunction(hTEL: TELHandle; paramblock : Ptr; size : LONGINT):
TELErr;`

Description `hTEL` is a handle to a telephone record.

`paramblock` is a parameter block defined by the specific tool.

`size` is the size (in bytes) of `paramblock`.

Result Codes `noErr, telBadTermErr`

Finding out which routines a tool supports

`TELToolFunctions` finds out whether a tool supports a specified Telephone Manager routine.

Function `TELToolFunctions(hTEL: TELHandle; msgcode : INTEGER; VAR supportsIt : BOOLEAN): TELErr;`

Description `hTEL` is a handle to a telephone record.

`msgcode` is a message code that specifies a particular Telephone Manager routine. For example, the message code `telResetTermMsg` specifies the routine `TELResetTerm`, and the message code `telCallPickupMsg` specifies the routine `TELCallPickup`. Most other message codes follow this naming convention. Refer to the file `Telephones.h` (C) or `Telephones.p` (Pascal) for a complete list of these message codes.

`supportsIt` is `TRUE` if the tool supports `msgcode`, and `FALSE` if it does not.

Result Codes `noErr, telBadTermErr`

Localizing configuration strings

The Telephone Manager provides two routines that make it easier to localize configuration strings.

TELIntlToEnglish

Translating into English

`TELIntlToEnglish` converts a configuration string, which is pointed to by `inputPtr`, to an American English configuration string pointed to by `outputPtr`.

Function `TELIntlToEnglish(hTEL: TELHandle; inputPtr: Ptr; VAR outputPtr: Ptr; language: INTEGER): OSErr;`

Description The function returns an operating-system error code if any internal errors occur.

`hTEL` is a handle to a telephone record.

`inputPtr` is a pointer to the C-style configuration string to be translated.

`outputPtr` is a pointer to the translation, an American English C-style configuration string.

The telephone tool allocates space for `outputPtr`. Your application is responsible for disposing of the pointer with `DisposPtr` when done with it.

`language` specifies the language from which the string is to be converted. Valid values for this field are shown in the description of the Script Manager in *Inside Macintosh*, Volume V. If the language specified is not supported, this routine returns `noErr`, but `outputPtr` is `NIL`.

Translating from English

TELEnglishToIntl converts an American English configuration string, which is pointed to by `inputPtr`, to a configuration string pointed to by `outputPtr`.

Function `TELEnglishToIntl(hTEL: TELHandle; inputPtr: Ptr; VAR outputPtr: Ptr; language: INTEGER): OSErr;`

Description The function returns an operating-system error code if any internal errors occur.

`hTEL` is a handle to a telephone record.

`inputPtr` is a pointer to an American English C-style configuration string to be translated.

`outputPtr` is a pointer to the translation, a C-style configuration string.

The telephone tool allocates space for `outputPtr`; your application is responsible for disposing of the pointer with `DisposePtr` when done with it.

`language` specifies the language to which the string is to be converted. Valid values for this field are shown in the description of the Script Manager in *Inside Macintosh*, Volume V. If the language specified is not supported, `noErr` is still returned, but `outputPtr` is `NIL`.

Monitoring and controlling the terminal

The Telephone Manager lets your application monitor and control the physical components associated with a terminal, such as a hookswitch, display, speaker, or microphone. To monitor and control these components, use the routines described in this section.

TELGetHooksw

Monitoring a hookswitch

TELGetHooksw finds out whether a hookswitch of the specified device is physically on-hook or off-hook.

Function `TELGetHooksw(hTEL: TELHandle; htype: INTEGER; VAR onHook: BOOLEAN) : TELErr;`

Description `hTEL` is a handle to a telephone record.

`htype` specifies the type of hookswitch, and has one of the following valid values:

CONST

```
telHandset      = 1; {hand-set hookswitch}
telSpeakerphone = 2; {speakerphone "ON" switch}
{Values 3 through 255 are reserved for Apple}
{Values 256 through 32768 are reserved for use by tools}
```

`onHook` equals `devOnHook` if the hookswitch of device `htype` is on, or `devOffHook` if it is off.

Result Codes `noErr`, `telBadHTypeErr`, `telBadTermErr`, `telHTypeNotSupp`

Setting a hookswitch

TELSetHooksw sets the physical state of a device's hookswitch, if the device supports this capability.

Function TELSetHooksw(hTEL: TELHandle; htype: INTEGER; onHook: BOOLEAN) : TELErr;

Description hTEL is a handle to a telephone record.
htype specifies the type of hookswitch, and has one of the following valid values:

```
CONST
    telHandset          = 1; {hand-set hookswitch}
    telSpeakerphone     = 2; {speakerphone "ON" switch}
    {Values 3 through 255 are reserved for Apple}
    {Values 256 through 32768 are reserved for use by tools}
```

onHook specifies the state to which the hookswitch will be set: on-hook if onHook equals devOnHook, or off-hook if onHook equals devOffHook.

- ◆ *Note:* If the user takes the telephone off-hook to place a call, the Telephone Manager relays a telCAOutgoingMsg message, and assigns a handle (type TELCAHandle) to a call-appearance record. If the user takes the telephone off-hook to receive a call, the Telephone Manager sends a telCAActiveMsg message.

Result Codes noErr, telBadHTypeErr, telBadTermErr, telHTypeNotSupp

Monitoring the volume level and device state

TELGetVolume finds out the current volume level and state of a specified device.

Function TELGetVolume(hTEL: TELHandle; volType: INTEGER; VAR level: INTEGER; VAR volState: INTEGER) : TELErr;

Description hTEL is a handle to a telephone record.
volType specifies the type of device volume being monitored. Valid values are as follows:

CONST

```
telHandsetSpeakerVol = 1;
                        {volume of handset speaker}
telHandsetMicVol      = 2;
                        {sensitivity of handset microphone}
telSpeakerphoneVol    = 3;
                        {volume of speakerphone}
telSpeakerphoneMicVol = 4;
                        {sensitivity of speakerphone
                        microphone}
telRingerVol          = 5;
                        {volume of handset ringer}
{Values 6 through 255 are reserved for the Telephone Manager}
{Values 256 through 32768 are available for use by tools}
```

level is the current volume level of the device, expressed as positive integer between 1 (lowest volume) and 100 (highest volume).

volState is the current state of the device, and has one of the following values:

CONST

```
telVolStateSame      = 0;
                        {volume left in same state,
                        on or off}
telVolStateOn         = 1;
                        {volume on at previously
                        specified level}
telVolStateOff        = 2;
                        {volume muted but volume
                        level unchanged}
{Values 3 through 255 are reserved for the Telephone Manager}
{Values 256 through 32768 are available for use by tools}
```

Result Codes

noErr, telBadLevelErr, telBadStateErr, telBadTermErr, telBadVTypeErr,
telStateNotSupp, telVTypeNotSupp

Setting the volume level and device state

`TELSetVolume` sets the volume level of a specified device and, optionally, mutes the device.

Function `TELSetVolume(hTEL: TELHandle; volType: INTEGER; VAR level: INTEGER; volState: INTEGER) : TELErr;`

Description `hTEL` is a handle to a telephone record.

`volType` specifies the type of device volume being set. Valid values are as follows:

```
CONST
    telHandsetSpeakerVol = 1; {volume of handset speaker}
    telHandsetMicVol     = 2; {sensitivity of
                               handset microphone}
    telSpeakerphoneVol   = 3; {volume of speakerphone}
    telSpeakerphoneMicVol = 4; {sensitivity of speakerphone
                               microphone}
    telRingerVol         = 5; {volume of handset ringer}
    {Values 6 through 255 are reserved for the Telephone Manager}
    {Values 256 through 32768 are available for use by tools}
```

`level` is the volume level your application is requesting, expressed as an integer between 0 and 100. After the volume is set, the Telephone Manager returns the actual volume level in `level`. Valid values for `level` are as follows:

```
CONST
    telVolSame    = 0;
    telVolMin     = 1;
    telVolMax     = 100;
```

Setting `level` to `telVolSame` leaves the volume level at its previous setting. Setting `level` to `telVolMin` sets the volume level to the minimum allowed by the device. Setting `level` to `telVolMax` sets the volume to the highest level the Telephone Manager allows.

Different telephone tools support different maximum values for `level`. To find the maximum `level` value for a particular tool, your application can set `level` to `telVolMax` before calling `TELSetVolume`. When `TELSetVolume` returns, the value of `level` is the maximum value support by the telephone tool.

- ◆ **Note:** Tools that do not support a `level` of `telVolMax` should instead set the volume level to the maximum allowed by the tool and return that volume level in `level`.

`volState` is the state to which your application is setting the device. Valid values are as follows:

CONST

```
telVolStateSame = 0;
                  {leave volume in current state, on or off}
telVolStateOn   = 1;
                  {turn on device at previous volume level}
telVolStateOff  = 2;
                  {mute volume; leave level setting unchanged}
{Values 3 through 255 are reserved for the Telephone Manager}
{Values 256 through 32768 are available for use by tools}
```

Result Codes

noErr, telBadLevelErr, telBadStateErr, telBadTermErr, telBadVTypeErr,
telStateNotSupp, telVTypeNotSupp

Making the terminal “ring”

TELAlert makes the terminal emit a specified alerting pattern. This routine controls the ringer of the terminal only, not those provided by applications.

Function `TELAlert(hTEL: TELHandle; VAR level: INTEGER; alertPattern: INTEGER)
: TELErr;`

Description `hTEL` is a handle to a telephone record.

`level` specifies the volume level at which the alerting pattern is to sound, expressed as an integer between 0 and 100. After the volume level is set, the Telephone Manager returns the actual volume level in `level`. Valid values for `level` are as follows:

```
CONST
    telVolSame      = 0;
    telVolMin       = 1;
    telVolMax       = 100;
```

Setting `level` to `telVolSame` leaves the volume level at its previous setting. Setting `level` to `telVolMin` sets the volume level to the minimum allowed by the device. Setting `level` to `telVolMax` sets the volume to the highest level the Telephone Manager allows.

- ◆ *Note:* Tools that do not support a `level` of `telVolMax` should instead set the volume level to the maximum allowed by the tool and return that volume level in `level`.

`alertPattern` is the alerting pattern the terminal is to emit. Valid values are as follows:

```
CONST
    telPattern0 = 0; {normal alerting pattern}
    telPattern1 = 1; {alerting pattern 1}
    telPattern2 = 2; {alerting pattern 2}
    telPattern3 = 3; {alerting pattern 3}
    telPattern4 = 4; {alerting pattern 4}
    telPattern5 = 5; {alerting pattern 5}
    telPattern6 = 6; {alerting pattern 6}
    telPattern7 = 7; {alerting pattern 7}
    telPatternOff = 15; {alerting pattern undefined}
    {Values 8 through 14 and 16 through 255 are
     reserved for the Telephone Manager}
    {Values 256 through 32768 are available for use by tools}
```

To find out which alerting patterns and volume levels the terminal supports, check the fields `ringerTypes` and `hasvolctl` in telephone record `hTEL`.

Result Codes `noErr`, `telAPattNotSupp`, `telBadAPattErr`, `telBadLevelErr`, `telBadTermErr`

Monitoring the display text

TELGetDisplay returns the current display text of the user's telephone, if the terminal stores this text.

Function

```
TELGetDisplay(hTEL: TELHandle; index: INTEGER; VAR displayMode:
INTEGER; VAR text: StringPtr) : TELErr;
```

Description

hTEL is a handle to a telephone record.

index specifies which item of the display text TELGetDisplay is to return. Valid values are as follows:

CONST

```
telEntireDisplay      =      0; {entire display}
{Values 1 through 255 are reserved for the Telephone Manager
and are otherwise invalid.}
{Values 256 through 32768 are available for use by tools.}
```

displayMode indicates the display mode. Valid values are as follows:

CONST

```
telNormalDisplayMode  =      1; {Normal mode}
telInspectMode        =      2; {Inspect mode}
telMiscMode           =      3; {Miscellaneous mode}
telRetrieveMode        =      4; {Message Retrieval mode}
telDirectoryQueryMode =      5; {Electronic Directory mode}
{Values 6 through 255 are reserved for the Telephone Manager
and are invalid}
{Values 256 through 32768 are available for use by tools.}
```

text points to the text currently showing on the display of the telephone. Your application is responsible for disposing of the string to which text points.

Result Codes

noErr, telBadIndex, telBadTermErr, telIndexNotSupp

Setting the display text

`TELSetDisplay` sets the text or display mode of the status display on the user's telephone.

Function `TELSetDisplay(hTEL: TELHandle; index, displayMode: INTEGER; text: Str255) : TELErr;`

Description `hTEL` is a handle to a telephone record.
`index` specifies which item of the display text `TELSetDisplay` is to set. Valid values are as follows:

```
CONST
    telEntireDisplay      =      0; {entire display}
    {Values 1 through 255 are reserved for the Telephone Manager
     and are invalid}
    {Values 256 through 32768 are available for use by tools}
```

`displayMode` specifies the display mode to which the status display will be set.

`text` is the text that will replace item `index` on the display. Your application is responsible for disposing of the string to which `text` points.

`TELSetDisplay` sets the display in mode `displayMode`.

Result Codes `noErr`, `telBadIndex`, `telBadTermErr`, `telIndexNotSupp`, `telModeNotSupp`

Controlling directory numbers

The Telephone Manager lets your application control each directory number associated with the user's terminal. To control directory numbers, use the routines described in this section.

TELCountDNs

Counting directory numbers

`TELCountDNs` returns the number of directory numbers associated with the user's terminal. The routine counts either all directory numbers or only those of a specified type.

Function

```
TELCountDNs(hTEL: TELHandle; dnType: INTEGER; physical: BOOLEAN) :  
INTEGER;
```

Description

`hTEL` is a handle to a telephone record.

`dnType` specifies the type of directory numbers the routine is to count. Valid values are as follows:

CONST

```
telAllDNs          = 0;  
                    {All DNs}  
telInternalDNs     = 1;  
                    {DNs connected to a PBX  
                    or nonpublic switch}  
telInternalDNsOnly = 2;  
                    {DNs connected to a PBX  
                    or nonpublic switch, and  
                    capable of placing internal  
                    calls only}  
telExternalDNsOnly = 3; {DNs connected to a  
                    public network}  
{Values 4 through 255 are reserved for the Telephone Manager  
and are otherwise invalid}  
{Values 256 through 32768 are reserved for use by tools}
```

`physical` specifies whether `TELCountDNs` will count all directory numbers of type `dnType` or only those to which commands can be sent, as indicated by the field `dnAccessible` of the directory-number record.

Result Codes

`noErr`, `telBadDNType`, `telBadTermErr`

Finding directory numbers by index

`TELDNLookupByIndex` returns a handle to the n th directory number of the user's terminal. Your application must call this routine once, typically after calling `TELCountDNs`, to get a handle to the directory-number record associated with a particular directory number.

Function

```
TELDNLookupByIndex(hTEL: TELHandle; dnType: INTEGER; physical:
BOOLEAN; index: INTEGER; VAR hTELDN: TELDNHandle) : TELErr;
```

Description

`hTEL` is a handle to a telephone record.

`dnType` specifies the type of directory numbers the routine will consider when finding the n th one. Valid values are as follows:

```
CONST
    telAllDNs           = 0;
                        {All DNs}
    telInternalDNs      = 1;
                        {DNs connected to a PBX
                        or nonpublic switch}
    telInternalDNsOnly  = 2;
                        {DNs connected to a PBX or
                        nonpublic switch, and capable
                        of placing internal calls only}
    telExternalDNsOnly  = 3;
                        {DNs connected to a
                        public network}
    telDNTypeUnknown    = 4;
                        {DN type unknown}
{Values 5 through 255 are reserved for the Telephone Manager
 and are otherwise invalid}
{Values 256 through 32768 are reserved for use by tools}
```

`physical` specifies whether `TELDNLookupByIndex` will consider all directory numbers of type `dnType` when finding directory number n , or consider only those to which commands can be sent.

`index` specifies which directory number `TELDNLookupByIndex` is to look up. The maximum value of `index` is the one that `TELCountDNs` returns when its selection criteria are the same as those of `TELDNLookupByIndex`—that is, when both calls have the same values for `dnType` and `physical`. Note that if `TELCountDNs` returns, for example, the integer 3, then the valid values of `index` are the integers 1 through 3—not 0 through 2.

`hTELDN` is a handle to the directory-number record for the n th directory number returned, or is `NIL` if no such handle is found.

Result Codes

```
noErr, telBadDNType, telBadIndex, telBadTermErr, telDNTypeNotSupp,
telIndexNotSupp
```

TELDNLookupByName

Finding directory numbers by name

TELDNLookupByName returns a handle to the directory number having a specified name—for example, (408) 555-1212.

Function

```
TELDNLookupByName(hTEL: TELHandle; dn: Str255; VAR hTELDN:
TELDNHandle) : TELErr;
```

Description

hTEL is a handle to a telephone record.

dn is the name of the directory number that **TELDNLookupByName** is to look up.

hTELDN is a handle to the directory-number record for **dn**. If the directory-number record is not found, **hTELDN** is zero, and **TELDNLookupByName** returns **telBadTermErr**.

Result Codes

noErr, **telBadDNErr**, **telBadTermErr**

TELGetDNInfo

Getting information about a directory number

TELGetDNInfo makes the telephone tool update a specified directory-number record. This updated record reflects the current state and current capabilities of the associated directory number.

Function

```
TELGetDNInfo(hTELDN: TELDNHandle) : TELErr;
```

Description

hTELDN is a handle to the directory-number record for the directory number whose information is requested.

The information returned by **TELGetDNInfo** is a “snapshot”; it reflects the state of the directory number as of the time the information was retrieved.

Result Codes

noErr, **telBadDNErr**

Finding the state of directory-number features

`TELGetDNFlags` updates the fields `featureFlags` and `forwardFlags` in a specified directory-number record and passes back the updated values of the fields. (For a description of the fields, refer to the section "Directory-Number Data Structure," earlier in this chapter.)

Function	<code>TELGetDNFlags(hTELDN: TELDNHandle; VAR dnFeatureFlags: LONGINT; VAR dnForwardFlags: LONGINT) : TELErr;</code>
Description	<code>hTELDN</code> is a handle to the directory-number record whose fields you wish to update. <code>dnFeatureFlags</code> stores the updated value of the <code>featureFlags</code> field of directory number <code>hTELDN</code>. <code>dnForwardFlags</code> stores the updated value of the <code>forwardFlags</code> field of directory number <code>hTELDN</code>.
Result Codes	<code>noErr, telBadDNerr</code>

TELDNSelect

Selecting a directory number

`TELDNSelect` selects or deselects a directory number on a system that has such a feature.

Function	<code>TELDNSelect(hTELDN: TELDNHandle; select: BOOLEAN) : TELErr;</code>
Description	<code>hTELDN</code> is a handle to the directory-number record to be selected. <code>select</code> indicates whether <code>TELDNSelect</code> is to select directory number <code>hTELDN</code> or deselect it. If <code>select</code> is set to <code>TRUE</code>, the directory number is selected; otherwise, it is deselected. Selecting a directory number puts on hold all calls of the previously selected directory number. Deselecting a directory number drops all call appearances associated with <code>hTELDN</code>.
Result Codes	<code>noErr, telBadDNerr, telBadSelect</code>

Disposing of a directory-number handle

TELDNDispose disposes of the handle to a specified directory-number record. Your application should call this routine when it no longer needs any information about the directory number associated with the record.

Note that disposing of the directory-number handle does not drop call appearances on that number. To drop a call appearance, use the routine **TELDrop**.

Function

TELDNDispose(hTELDN: TELDNHandle) : TELErr;

Description

hTELDN is a handle to a directory-number record.

TELDNDispose disposes of **hTELDN** and all handles to call-appearance records associated with **hTELDN**. In addition, the routine removes any directory-number message handlers or call-appearance message handlers registered for the directory number associated with **hTELDN**.

Result Codes

noErr, telBadDNErr

Controlling call appearances

The Telephone Manager lets your application control each call appearance associated with a particular directory number. To control call appearances, use the routines described in this section.

TELCountCAs

Counting call appearances

TELCountCAs returns the number of non-idle call appearances currently associated with a particular directory number. The routine counts either all such call appearances or only those of a specified type.

Function `TELCountCAs(hTELDN: TELDNHandle; intExt: INTEGER) : INTEGER;`

Description `hTELDN` is a handle to a directory-number record.

`intExt` specifies the type of call appearances the routine is to count. Valid values are as follows:

```
CONST
    telAllCallOrigins      = 0; {All call appearances}
    telInternalCall        = 1; {Internal call appearances only}
    telExternalCall        = 2; {External call appearances only}
    {Values 3 through 255 are reserved for the Telephone Manager
     and are otherwise invalid}
    {Values 256 through 32768 are reserved for use by tools}
```

Result Codes `noErr, telBadDNerr, telBadIntExt, telIntExtNotSupp`

TELCALookup

Finding call appearances

TELCALookup returns a handle to the *n*th call appearance currently associated with a specified directory number.

Function `TELCALookup(hTELDN: TELDNHandle; intExt: INTEGER; index: INTEGER; VAR hTELCA: TELCAHandle) : TELErr;`

Description `hTELDN` is a handle to a directory-number record.

`intExt` specifies the types of call appearance the routine will count when finding the *n*th one. Valid values are as follows:

CONST

```
telAllCallOrigins      = 0; {All call appearances}
telInternalCall        = 1; {Internal call appearances only}
telExternalCall        = 2; {External call appearances only}
{Values 3 through 255 are reserved for the Telephone Manager
 and are otherwise invalid}
{Values 256 through 32768 are reserved for use by tools}
```

`index` specifies which call appearance `TELCALookup` is to find. The maximum value of `index` is the one that `TELCountCAs` returns when its selection criteria are the same as those of `TELCALookup`—that is, when both calls have the same values for `hTELDN` and `intExt`. Note that if `TELCountCAs` returns, for example, the integer 3, then the valid values of `index` are the integers 1 through 3—not 0 through 2.

`hTELCA` is a handle to the call-appearance record for the *n*th call appearance returned.

In general, your application is notified of each valid call-appearance handle, either because it placed the call or because it received a Telephone Manager message such as `CAOffer` or `CAAlerting`. Thus, in general, your application need not call `TELCALookup` to obtain a call-appearance handle. But it is possible that some call appearances will already be in use when your application starts. `TELCALookup` lets you obtain handles for those call appearances.

Result Codes `noErr`, `telBadDNErr`, `telBadIndex`, `telBadIntExt`, `telIntExtNotSupp`

TELGetCAInfo

Getting information about a call appearance

`TELGetCAInfo` updates a specified call-appearance record. This updated record reflects the current state and current capabilities of the call appearance.

Function `TELGetCAInfo(hTELCA: TELCAHandle) : TELErr;`

Description `hTELCA` is a handle to the call-appearance record being updated.

The information in the updated call-appearance record is a “snapshot”; it reflects the state of the call appearance as of the time the information was retrieved.

Result Codes `noErr`, `telBadCAErr`

Finding the state of a call appearance

`TELGetCAState` requests that the telephone tool update the `caState` field of a particular call-appearance record and pass back the updated value of the field.

Function `TELGetCAState(hTELCA: TELCAHandle; VAR state: INTEGER) : TELErr;`

Description `hTELCA` is a handle to a call-appearance record.

`state` stores the updated value of the `caState` field of call-appearance record `hTELCA`. (For a list of call-appearance states, refer to the description of `caState` in the section "Call-Appearance Data Structure," earlier in this chapter.)

Result Codes `noErr`, `telBadCAErr`

Finding the state of call-appearance features

`TELGetCAFlags` updates the fields `featureFlags` and `otherFeatures` of a specified call-appearance record and passes back the updated values of the fields. (For a description of the fields, refer to the section "Call-Appearance Data Structure," earlier in this chapter.)

Function `TELGetCAFlags(hTELCA: TELCAHandle; VAR caFeatureFlags: LONGINT; VAR caOtherFeatures: LONGINT) : TELErr;`

Description `hTELCA` is a handle to the call-appearance record whose fields you wish to update.

`caFeatureFlags` stores the updated value of the `featureFlags` field of call appearance record `hTELCA`.

`caOtherFeatures` stores the updated value of the `otherFeatures` field of call appearance record `hTELCA`.

Result Codes `noErr`, `telBadCAErr`

Disposing of a call-appearance handle

`TELCADispose` disposes of the handle to a specified call-appearance record. Your application should call this routine when it no longer needs any information about the call appearance associated with the record.

Note that disposing of the call-appearance handle does not drop the associated call. To drop a call, use the routine `TELDrop`.

Function	<code>TELCADispose(hTELCA: TELCAHandle) : TELErr;</code>
Description	<code>hTELCA</code> is a handle to a call-appearance record. <code>TELCADispose</code> disposes of <code>hTELCA</code>.
Result Codes	<code>noErr</code> , <code>telBadCAErr</code>

Miscellaneous routines

The routines described in this section perform a variety of tasks.

TELGetToolName

Getting the name of a tool

`TELGetToolName` returns in `name` the name of the tool specified by `procID`.

Procedure `TELGetToolName(procID: INTEGER; VAR name: Str255);`

Description If `procID` references a telephone tool that does not exist, the Telephone Manager sets `name` to `NIL`.

TELGetVersion

Getting 'vers' resource information

`TELGetVersion` returns a handle to a relocatable block, which contains the information in the telephone tool's 'vers' resource with ID=1. Your application is responsible for disposing of the handle when done with it.

◆ *Note:* The handle returned is *not* a resource handle.

Function `TELGetVersion(hTEL: TELHandle): Handle;`

Description `hTEL` is a handle to a telephone record.

TELGetTELVersion

Getting the Telephone Manager version number

`TELGetTELVersion` returns the version number of the Telephone Manager.

Function `TELGetTELVersion: INTEGER;`

Description The version number of the Telephone Manager described in this document is as follows:

```
CONST
    curTELVersion = 1;
```

Routines your application must provide

To use the Telephone Manager, your application must be able to receive messages that the Telephone Manager relays from tools. These messages are of three types: terminal messages, directory-number messages, and call-appearance messages. These messages are listed in Appendix B. Your application must provide a message-handling routine for each message type, and must register the routines with the Telephone Manager by calling `TELTermMsgHand`, `TELDNMsgHand`, or `TELCAMsgHand`.

This section provides three routine templates—`MyTermMsgHandler`, `MyDNMsgHandler`, and `MyCAMsgHandler`—one for each of the message handlers your application must provide. These templates show only the interface your message handlers must have. You provide the actual routines, which can have any name. Refer to the section “Handling Messages,” earlier in this chapter, for information on registering your message handlers with the Telephone Manager.

`MyTermMsgHandler`

Template for terminal message handlers

`MyTermMsgHandler` is a routine you must write to handle messages from an application user's terminal.

Procedure

```
MyTermMsgHandler(hTEL: TELHandle; msg: LONGINT; mtype, value:
INTEGER; globals: LONGINT);
```

Description

`hTEL` is a handle to the telephone record for the user's terminal.

`msg` specifies the type of terminal message sent. Refer to Appendix B for a list of valid values for `msg`.

`mtype` and `value`, if used, specify attributes of the message. Their exact meaning and use vary, depending on the value of `msg`.

`globals` is a pointer to a location in memory—for example, to your application's globals (register A5). Your application must previously have passed the value of `globals` when calling `TELTermMsgHand` to register `MyTermMsgHandler`.

Template for directory-number message handlers

`MyDNMsgHandler` is a routine you must write to handle messages about a particular directory number.

Procedure `MyDNMsgHandler(hTELDN: TELDNHandle; msg: LONGINT; mtype, value: INTEGER; rmtDN, rmtName, rmtSubaddress : StringPtr; globals: LONGINT);`

Description `hTELDN` is a handle to a directory-number record.

`msg` a constant showing the type of directory-number message sent. Refer to Appendix B for a list of valid values for `msg`.

`mtype` and `value`, if used, specify attributes of the message. Their exact meaning varies, depending on the value of `msg`.

`rmtDN`, `rmtName`, and `rmtSubaddress` specify the directory number, party, and subaddress being called, if such information is applicable to the call and is available from the telephone network switch.

`globals` is a pointer to a location in memory—for example, to your application's globals (register A5). Your application must previously have passed the value of `globals` when calling `TELDNMsgHand` to register `MyDNMsgHandler`.

Template for call-appearance message handlers

`MyCAMsgHandler` is a routine you must write to handle messages about a particular call appearance.

Procedure `MyCAMsgHandler(hTELCA: TELCAHandle; msg: LONGINT; mtype, value: INTEGER; msgInfo : Ptr; globals: LONGINT);`

Description `hTELCA` is a handle to a call-appearance record.

`msg` specifies the type of call-appearance message sent. Refer to Appendix B for a list and description of valid values for `msg`.

`mtype` and `value`, if used, specify attributes of the message. Their exact meaning varies, depending on the value of `msg`.

`msgInfo` specifies the call appearance and person being called, if such information is applicable to the call and is available from the telephone network switch.

`globals` is a pointer to a location in memory—for example, to your application's globals (register A5). Your application must previously have passed the value of `globals` when calling `TELCAMsgHand` to register `MyCAMsgHandler`.

Chapter 3 Writing Telephone Tools

T H I S C H A P T E R provides information about writing a telephone tool. It first discusses general concepts relevant to writing a tool and then describes the six resources that are an essential part of a telephone tool. These six resources are exactly analogous to the six resources that must be in any communications tool intended for the Communications Toolbox. One of these resources, the main code resource, is far more complex than the others. For this reason, although it is introduced in this chapter, details about it are given in a separate chapter, Chapter 4. For information about writing your main code resource, see that chapter.

To write your own telephone tool, you need to be familiar with the Telephone Manager, with which your tool will interface. See Chapters 1 and 2 for information about the Telephone Manager. You should also be familiar with the Apple's guidelines for communications tools, which are discussed in *Inside the Macintosh Communications Toolbox*.

About writing a telephone tool

The Telephone Manager interacts with tools in much the same way that other Communications Toolbox managers do. The application calls a routine, which the Telephone Manager handles by sending a message to a tool. For example, when an application requires a service, such as the creation of a new telephone record, it calls the `TELNew` routine. The Telephone Manager passes this request on by issuing a message, `telNewMsg`, to the main code resource of the appropriate tool.

△ **Important** Telephone tools differ from other tools in one important way. The main code resource of a telephone tool not only receives messages from the Telephone Manager, but also sends messages back—for example, to relay information from the network switch. The messages that your main code resource can send are described in this chapter. The Telephone Manager routines associated with these messages are described in Chapter 2. △

The six tool resources

You need to create six resources to make your own telephone tool. All of these resources are described in this chapter, except the main code resource, which is described in detail in Chapter 4.

There is one tool-related resource, which is optional:

'vbnd' The bundle resource contains the name of the tool and information about what resources belong to the tool.

You also need to write five code resources, which *must* be part of your tool:

'vdef' The main code resource performs the basic telephone functions, such as `TELNew`.

'vval' The validation code resource validates telephone records with `TELValidate`, and also fills in configuration record default values with `TELDefault`.

'vset' The setup-definition code resource supports the custom tool-settings dialog box, which allows users to configure telephone tools.

'vscr' The scripting language interface code resource handles the interface between a scripting language and the tool.

'vloc' The localization code resource handles localization of configuration strings.

The bundle resource

The tool bundle contains the master list of resources that are associated with your telephone tool. Besides the six standard resources, the tool bundle can contain references to any additional resources that your tool requires, such as dialog boxes or menus. Although your tool will work without a bundle resource, including one is good programming practice. The bundle resource allows you to change resource IDs when conflicts arise, without having to recompile your code.

Your telephone tool can refer to resources with local IDs that the Communications Resource Manager can map to actual resource IDs. (Your tool should use the Communications Resource Manager routines `CRMLocalToRealID` and `CRMRealToLocalID`.) The telephone bundle resource, shown here, provides a data structure to accommodate this mapping.

```
type 'vbnd' { /**/
    integer = $$CountOf(ToArray) - 1;
    array TypeArray {
        literal longint;      /* Type */
        integer = $$CountOf(IDArray) - 1;
        wide array IDArray {
            integer;          /* Local ID */
            integer;          /* Actual ID */
        };
    };
};
```

The validation code resource

The validation code resource parses two possible messages from the Telephone Manager: `telValidateMsg` and `telDefaultMsg`. An application or tool will request one of these services when it requires your tool to check the values in the telephone record or to reset the telephone record to its default values. Your telephone tool should contain the default values for the telephone record.

The validation code resource should be a resource of type `'vval'`. It should be able to accept the messages shown in this example:

```
FUNCTION vval(hTEL: TELHandle; msg: INTEGER; p1, p2, p3: LONGINT): LONGINT;
VAR
    pConfig:    ConfigPtr;

BEGIN
    CASE msg OF
        telValidateMsg:    { hTEL is valid here }
            BEGIN
                vval = DoValidate(hTEL);
            END;
        telDefaultMsg:    { hTEL is not valid here }
            BEGIN
                { p1 is a pointer to the configPtr }
                { p2 is allocate or not }
                { p3 is zero }

                IF p2 = TRUE THEN
                    BEGIN
                        pConfig := ConfigPtr(NewPtr(SIZEOF(ConfigRecord)));
                        ConfigHandle(p1)^ := pConfig;
                        { real programmers check errors here }
                    END
                ELSE
                    BEGIN
                        ConfigHandle(p1)^ := pConfig;
                        := ;
                    END;
                DoDefault(pConfig);
            END;
    END; { case }
END;
```

The messages accepted by the validation code resource and their associated values are as follows:

```
CONST
{ validation code resource messages }
    telValidateMsg      =      0;
    telDefaultMsg       =      1;
```

For each of the messages defined here, `p1`, `p2`, and `p3` take on different meanings. These meanings are discussed in the message descriptions that follow. If your tool receives a message other than those shown, it should return `telNotSupported`.

telValidateMsg

Your tool will receive `telValidateMsg` when the application requires your tool to validate the fields in the telephone record. Your tool should compare the values in this record with the values specified in the tool.

The example code given here shows how your tool can respond to `telValidateMsg`.

After executing the code necessary to respond to `telValidateMsg`, your code should pass back 0 if there were no errors, or 1 if the configuration record had to be rebuilt by your tool. `p1`, `p2`, and `p3` should be ignored.

```
{ perform validate here }
FUNCTION DoValidate(hTEL: TELHandle): LONGINT;
VAR
    pPrivate:    PrivatePtr;
    pConfig:     ConfigPtr;

BEGIN
    pConfig := ConfigPtr(hTEL^.config);
    pPrivate := PrivatePtr(hTEL^.private);

    IF pConfig^.foobar = 0 THEN
        DoValidate := 0      { okey dokey }
    ELSE
        DoValidate := 1;     { uh-oh }
END;
```

telDefaultMsg

Your tool will receive `telDefaultMsg` when the application requires your tool to fill in the fields of a telephone record. Default values should be specified in your tool.

After executing the code necessary to respond to `telDefaultMsg`, `p1` should pass back a pointer to the configuration record pointer. If `p2` contained 1 when `TELDefault` was called, your tool should allocate the configuration record and return the pointer in `p1`. If `p2` was 0, then your tool should simply use the configuration pointer obtained by dereferencing `p1`.

The setup-definition code resource

Applications can present users with a custom dialog box containing tool-specific items that allows them to configure their own telephones or select a telephone tool. The Telephone Manager routines `TELSetupPreflight`, `TELSetupSetup`, `TELSetupItem`, `TELSetupFilter`, and `TELSetupCleanup` make this possible.

The telephone tool setup code resource should be a function called `'vset'`, and should be able to handle the following parameters:

```
{ main entry point for vset resource }
FUNCTION vset(pSetup: TELSetupPtr; msg: INTEGER;
  p1, p2, p3: LONGINT): LONGINT;
TYPE
  LocalHandle = ^LocalPtr;
  LocalPtr = ^LocalRecord;
  LocalRecord = RECORD { private tool setup context }
    foobar: LONGINT;
  END;

  IntPtr = ^INTEGER;
  EventPtr = ^EventRecord;

BEGIN
  CASE msg OF
    telSpreflightMsg:
      BEGIN
        theCookie := CookiePtr(NewPtr(SIZEOF(CookieRecord)));
        CookieHandle(p3)^ := theCookie;      { send back theCookie }
        vset := Preflight(pSetup, theCookie);
      END;
    telSsetupMsg:
      BEGIN
        theCookie := CookieHandle(p3)^;      { get the magic cookie }
        Setup(pSetup);                       { do the setup }
      END;
    telSitemMsg:
      BEGIN
        theCookie := CookieHandle(p3)^;      { get the magic cookie }
        Item(pSetup, theCookie, IntPtr(p1)); { process the items hit }
      END;
```

```

telSfilterMsg:
    BEGIN
        theCookie := CookieHandle(p3)^;           { get the magic cookie }
        vset := Filter(pSetup, theCookie, EventPtr(p1), IntPtr(p2));
    END;
telScleanupMsg:
    BEGIN
        theCookie := CookieHandle(p3)^;           { get the magic cookie }
        DisposPtr(Ptr(theCookie));                { and get rid of it }
    END;
END; { case }
END;

```

In the preceding code sample, `magic cookie` is intended to store the private data structure of your telephone tool. Note that there is no message `TELSpstflightMsg`, because the corresponding Telephone Manager routine, `TELSetupPostflight`, requires no action from your tool. `TELSetupPostflight` releases the 'vset' resource from memory.

Valid values for `msg` are as follows:

```

CONST
    telSpreflightMsg    =    0;
    telSsetupMsg        =    1;
    telSitemMsg         =    2;
    telSfilterMsg       =    3;
    telScleanupMsg      =    4;

```

For each of the messages just shown, `p1`, `p2`, and `p3` take on different meanings. These meanings are discussed in the message descriptions that follow. If your tool receives a message other than those shown, it should return `telNotSupported`. When your tool handles these routines, it will use a `TELSetupStruct` data structure.

```

TYPE
    TELSetupPtr    =    ^TELSetupStruct;
    TELSetupStruct =    RECORD
        theDialog    :    DialogPtr;
        count        :    INTEGER; {dialog item number of first appended item}
        theConfig    :    Ptr;
        procID       :    INTEGER
    END;

```

telSpreflightMsg

Your setup-definition code resource should perform a function similar to that shown in the example code when it receives `telSpreflightMsg` from the Telephone Manager. This is where your tool retrieves the 'DITL' resource, to be appended to the `TELchoose` dialog.

When passed to your telephone tool, `p3` will be a pointer to a `LONGINT` that gets passed to the other routines during setup definition. `p3` should serve as `magicCookie` if the setup-definition procedure requires some private context.

After executing the code necessary to respond to `telSpreflightMsg`, your telephone tool should return a handle to a dialog item list. This handle should then be disposed of by the caller of this function.

```
FUNCTION Preflight(pSetup: TELSetupPtr; theCookie: LocalPtr): LONGINT;
CONST
    localID = 1;                { we want DITL local ID 1 }

VAR
    hDITL: Handle;
    theID: INTEGER;
    oldRF: INTEGER;

BEGIN
    theCookie^.foobar := 0;      { setup theCookie }

    theID := CRMLocalToRealID(ClassTEL, pSetup^.procID, 'DITL', localID);
    IF theID = -1 THEN
        Preflight := 0          { no DITL found }
    ELSE
        BEGIN
            oldRF := CurResFile;
            UseResFile(pSetup^.procID); { procID is the tool refnum }
            hDITL := GetResource('DITL', theID);
            UseResFile(oldRF);

            IF hDITL <> NIL THEN
                DetachResource(hDITL);    { got it so detach it }

                Preflight := LONGINT(hDITL);
        END;
    END;
END;
```

telSsetupMsg

Your setup-definition code resource should perform a function similar to that shown in the example code when it receives `telSsetupMsg` from the Telephone Manager. This is where your tool initializes all items in the 'DITL' resource retrieved by `TELSetupPreflight`.

When passed to your telephone tool, `p3` will be a pointer to `magicCookie`, which is a `LONGINT`.

```
PROCEDURE Setup(pSetup: TELSetupPtr);
CONST
    myFirstItem = 1;
    mySecondItem = 2;

VAR
    first: INTEGER;                { first item appended (0-based) }
    pConfig: ConfigPtr;

BEGIN
    WITH pSetup^ DO
```

```

BEGIN
    first := count - 1;           { count is 1-based }
    pConfig := ConfigPtr(theConfig); { get the config ptr }

    GetDItem(theDialog, first+myFirstItem, itemKind, itemHandle,
              itemRect);
    SetCtlValue(ControlHandle(itemHandle), pConfig^.foobar);

    GetDItem(theDialog, first+mySecondItem, itemKind, itemHandle,
              itemRect);
    SetCtlValue(ControlHandle(itemHandle), 1-pConfig^.foobar);

    END; {with}
END;

```

telSitemMsg

Your setup-definition code resource should perform a function similar to that shown in the example code when it receives `telSitemMsg` from the Telephone Manager. Your tool receives `telSitemMsg` when an item belonging to your tool is hit.

When passed to your telephone tool, `p1` points to an item that was selected from the dialog box item list, and `p3` contains a pointer to `magicCookie`. Your tool can change the selected item by modifying the item number to which `p1` points.

```

PROCEDURE Item(pSetup: TELSetupPtr; pItem: IntPtr);
CONST
    myFirstItem      = 1;
    mySecondItem     = 2;

VAR
    first            : INTEGER;      { first item appended (0-based) }
    pConfig           : ConfigPtr;
    value             : INTEGER;

BEGIN
    WITH pSetup^ DO
    BEGIN
        first := count - 1;           { count is 1-based }
        pConfig := ConfigPtr(theConfig); { get the config ptr }

        CASE pItem^ -first OF
        myFirstItem:
            BEGIN
                GetDItem(theDialog, first+myFirstItem, itemKind,
                          itemHandle, itemRect);
                value := GetCtlValue(ControlHandle(itemHandle));
                value := 1 - value;
                pConfig^.foobar := value; { stick into config record }
                SetCtlValue(ControlHandle(itemHandle), value); { update control }
            END;

```

```

        mySecondItem:
            BEGIN
                SysBeep(5);
                FlashMenuBar(0);
            END;
        END; { case }
    END; { with }
END;

```

telSfilterMsg

Your setup-definition code resource should perform a function similar to that shown in the example code when it receives `telSfilterMsg` from the Telephone Manager.

When passed to your telephone tool, `p1` will contain a pointer to an event record, `p2` will contain a pointer to an item clicked in the dialog box list, and `p3` will contain a pointer to `magicCookie`.

If the event that was passed to this function was handled, your telephone tool should return `TRUE`; otherwise, it should return `FALSE`.

```

FUNCTION Filter(pSetup: TELSetupPtr; theCookie: LocalPtr;
               pEvent: EventPtr; pItem: IntPtr): LONGINT;
BEGIN
    Filter := 0;                                { not hungry }

    IF pEvent^.what = keyDown THEN              { eat all keyDowns }
    BEGIN
        SysBeep(5);
        Filter := 1;                            { processed }
    END;
END;

```

telScleanupMsg

Your setup-definition code resource should perform a function similar to the one shown in the example code when it receives `telScleanupMsg` from the Telephone Manager. This is where your tool should dispose of any private storage allocated during the creation of the custom tool-settings dialog box.

When passed to your telephone tool, `p3` will contain a pointer to `magicCookie`.

```

PROCEDURE myCleanup(p3: LONGINT);
BEGIN
    DisposPtr( Ptr(p3) );                      { dispose of magicCookie }
    p3 := 0;
END;

```

The scripting language interface code resource

Your telephone tool's scripting language interface code resource is responsible for handling the interface between your tool and a scripting language. Also, it must provide complete configuration information for saving and opening documents.

Your scripting interface code resource must handle two messages: `telMgetMsg` and `telMsetMsg`. It should be a resource of type `'vscr'`, and should be able to handle the parameters that are shown in this example:

```
FUNCTION vscr(hTEL: TELHandle; msg: INTEGER; p1, p2, p3: LONGINT): LONGINT;
VAR
    pConfig:      ConfigPtr;

BEGIN
    vscr := 0;      { for now }

    CASE msg OF
        telMgetMsg:
            vscr := LONGINT(GetConfig(hTEL));
        telMsetMsg:
            vscr := SetConfig(hTEL, Ptr(p1));
    END; { case }
END;
```

Valid values for `msg` are as follows:

```
CONST
    telMgetMsg    =    0;
    telMsetMsg    =    1;
```

For each of the messages defined here, `p1`, `p2`, and `p3` take on different meanings. These meanings are discussed in the message descriptions that follow. If your tool receives a message other than those shown, it should return `telNotSupported`.

telMgetMsg

Your tool will receive `telMgetMsg` from the Telephone Manager when the application requires a string that describes the telephone record. The sample code shows how your application can handle `telMgetMsg`.

After executing the code necessary to respond to `telMgetMsg`, your telephone tool should return `NIL` if there was a problem constructing the configuration string. Otherwise, it should return a pointer to a null-terminated string that contains American English tokens representing the configuration record pointed to by `config` in the telephone record.

```
FUNCTION GetConfig(hTEL: TELHandle): Ptr;
VAR
    thePtr:      Ptr;
    pConfig:     ConfigPtr;
    theString,
    string2:     STR255;
```

```

BEGIN
    pConfig := ConfigPtr(hTEL^.config);      { get the config record }
    theString := 'FOOBAR ';                 { attribute name is FOOBAR }
    NumToString(pConfig^.foobar, string2);   { get the attribute value }
    theString := CONCAT(string, string2);     { make the config string }
    thePtr := NewPtr(SIZEOF(LENGTH(theString)+1));

    IF thePtr <> NIL THEN
    BEGIN
        BlockMove(Ptr(LONGINT(@theString)+1),
            thePtr, LENGTH(theString)); { copy it }
        Ptr(LONGINT(thePtr)+LENGTH(theString))^ := 0;      { 0 terminate it }
    END;

    GetConfig := thePtr;                      { bye bye }
END;

```

telMsetMsg

Your tool will receive `telMsetMsg` from the Telephone Manager when the application requires your tool to set the fields of the telephone record to values that are specified in a string. The Telephone Manager will pass a pointer to this string as a parameter to this call. The sample code shows how your tool can handle `telMsetMsg`.

When passed to your telephone tool's scripting interface code resource, `p1` will be a pointer to an American English null-terminated string that contains tokens representing a configuration record.

Your tool should return one of the following values: a number less than -1 to indicate an `osErr`, -1 to indicate a generic error, 0 if there was no problem with the string, or a positive number to indicate the character position where parsing was stopped.

The Telephone Manager automatically calls `TELvalidate` after your tool has responded to `telMsetMsg`.

```

FUNCTION SetConfig(hTEL: TELHandle; theSource: Ptr): INTEGER;
VAR
    pConfig      : ConfigPtr;      { tool specific config record }
    paramStr,
    valueStr      : Str255;         { parameter and value strings }
    outOfTokens   : BOOLEAN;        { end of the line? }
    returnVal     : INTEGER;        { what to return }

```

```

BEGIN
    { Init some stuff }
    pConfig := ConfigPtr(hTEL^.config);
    returnVal := noErr;

    IF (theSource^ = CHR(0)) THEN
        outOfTokens := TRUE
    ELSE
        outOfTokens := FALSE;

```

```

WHILE NOT outOfTokens DO BEGIN
    (* Build the first token and put it into paramStr *)

    IF (paramStr = 'FOOBAR') THEN BEGIN
        (* Build the next token and put it into valueStr *)

        pConfig^.foobar := valueStr;

    END
    ELSE BEGIN
        (* returnVal = location of the paramStr *)
        LEAVE;
    END;

    (* index to next token *)

END; { while }

SetConfig := returnVal;
END;

```

The localization code resource

Your telephone tool's localization code resource is responsible for providing the services necessary to localize your tool. It must handle two messages, `telL2English` and `telL2Intl`.

Your localization code resource should be a resource of type `'vloc'`. It should be able to handle the parameters shown in the example code.

```

FUNCTION      vloc(hTEL: TELHandle; msg: INTEGER; p1, p2, p3: LONGINT) : LONGINT;

```

Valid values for `msg` are as follows:

```

CONST
    telL2English  =    0;
    telL2Intl     =    1;

```

For each of the messages defined here, `p1`, `p2`, and `p3` take on different meanings. These meanings are discussed in the message descriptions that follow.

telL2English and telL2Intl

Your tool will receive `telL2English` from the Telephone Manager when the application requires your tool to localize a string to English. When the parameters `p1`, `p2`, and `p3` are passed to your tool, `p1` will contain a pointer to a localized null-terminated string that contains tokens representing a configuration record; `p2` will contain a pointer that points to a second pointer. Your tool will have to allocate space for this pointer (by calling `NewPtr`), which contains the American English null-terminated configuration string. `p3` will contain a language identifier, which is defined in the discussion of the Script Manager in *Inside Macintosh*, Volume V.

Your tool will receive `tell2Int1` from the Telephone Manager when the application requires your tool to localize a string to a language other than English. When the parameters `p1`, `p2`, and `p3` are passed to your tool, `p1` will contain a pointer to an American English null-terminated string that contains tokens representing a configuration record; `p2` will contain a pointer to a second pointer. Your tool will have to allocate space for this pointer, which contains the localized configuration string. `p3` will contain a language identifier, which is defined in the Script Manager in *Inside Macintosh*, Volume V. The next code example shows how your tool can handle both `tell2English` and `tell2Int1`.

After executing the code necessary to respond to `tell2English` or `tell2Int1`, your routine should return `NIL` if there was a Memory Manager error or if the language requested is not available. It should also return any appropriate error code in the status field of the telephone record.

```
{ main entry point for vloc resource }
FUNCTION vloc(hTEL: TELHandle; msg: INTEGER; p1, p2, p3: LONGINT): LONGINT;
TYPE
    PtrPtr = ^Ptr;

VAR
    outPtr: Ptr;
    procID: INTEGER;

BEGIN
    outPtr := PtrPtr(p2)^;           { get output pointer }
    case msg of
        tell2English:
            vloc := Translate( Ptr(p1),outPtr,p3,verUS);
        tell2Int1:
            vloc := Translate( Ptr(p1),outPtr,verUS,p3);
    end; {case}
    PtrPtr(p2)^ := outPtr;           { return output pointer }
END; { mytsrDEF }

{ Translates an input config string from one language to another }
{ returns 0 if no problem, non zero if there is a problem }
{ This routine needs to allocate outputStr }
{ if language is not supported, return 0 but leave outputStr NIL }

function Translate( inputStr: Ptr; var outputStr: Ptr;
                    fromLanguage,toLanguage: longint): longint;

BEGIN
END; { Translate }
```

config: the configuration record

An application using your tool may save and restore the contents of a configuration record to set the state of the telephone at any time. The configuration record, therefore, should be self-contained and should not contain any pointers or handles to other data structures. Your tool allocates this record in response to `telDefaultMsg`. The Telephone Manager, not the tool, deallocates the configuration record when the application calls `TELDispose`.

Chapter 4 **Writing Your Tool's Main Code Resource**

THIS CHAPTER tells you how to write the main code resource for a telephone tool. There are at least five other code resources that you need to include as part of your tool; they are described in Chapter 3. You should read that chapter, as well as Chapters 1 and 2, before reading this chapter.

This chapter describes the messages, parameters, and data structures that the Telephone Manager passes to your tool's main code resource. It also describes the messages that your main code resource can send back to the Telephone Manager.

The main code resource

The main code resource of your tool serves two purposes. The first is to parse messages from the Telephone Manager and then to branch to a routine that can handle each message. The second purpose is to send messages to the master message handlers of the Telephone Manager, in response to activity on the telephone network.

The main code resource should be a resource of type 'vdef' and should be able to accept the parameters shown here:

```
FUNCTION      vdef(hTEL: TELHandle; pTELTerm: TELTermPtr;
                  hDN: TELDNHandle; hCA: TELCAHandle;
                  msg: INTEGER; p1, p2, p3: LONGINT) : INTEGER;
```

For each of the messages defined here, the first five parameters 'vdef' returns, namely hTEL, pTELTerm, hDN, hCA, and msg, have the following meanings:

hTEL is a handle to a telephone record for the user's terminal.

pTELTerm points to a record of type TELTermRecord, part of the telephone record hTEL.

hDN is a handle to a directory-number record assigned to terminal hTEL.

hCA is a handle to a call-appearance record on directory number hDN. msg identifies the message.

The remaining three parameters 'vdef' returns, namely p1, p2, and p3, take on different meanings in each message. These parameters are described in the message descriptions later in this chapter.

Your tool can respond to messages by returning either an operating-system error code or one of the following four codes: telFeatNotSupp, telFeatNotSub, telFeatNotAvail, or telFeatActive.

telFeatNotSupp means that your tool does not understand the message it received.

telFeatNotSub means that your tool supports the requested feature, but the feature is not subscribed.

telFeatNotAvail means that, although your tool supports the requested feature and the feature is subscribed, the feature was not available when requested.

telFeatActive means that your tool supports the requested feature, but the feature is already active.

When relaying requests from applications to your tool, the Telephone Manager checks any handles passed in the requests, making sure they are on a list of seemingly valid handles. (This list is maintained by the Telephone Manager.) But the Telephone Manager can sometimes relay an invalid handle—one for a terminal, directory number, or call appearance that your tool no longer supports.

If your tool receives an invalid handle, it should send one of the following three result codes: telBadTermErr (for terminals), telBadDNErr (for directory numbers), or telBadCAErr (for call appearances). In addition, your tool should set to -1 the field tRef (of the telephone record), dnRef (of the directory-number record), or caRef (of the call-appearance record), depending on the type of the invalid handle.

Messages that the main code resource accepts

The messages accepted by the main code resource, and their associated values, are as follows. For a listing of these messages in numerical order, see the header file `TelephoneTools.p` (Pascal) or `TelephoneTools.h` (C).

CONST

<code>telAcceptCallMsg</code>	<code>=</code>	<code>206;</code>
<code>telActivateMsg</code>	<code>=</code>	<code>6;</code>
<code>telAlertMsg</code>	<code>=</code>	<code>74;</code>
<code>telAnswerCallMsg</code>	<code>=</code>	<code>209;</code>
<code>telCADisposeMsg</code>	<code>=</code>	<code>200;</code>
<code>telCAEventsSuppMsg</code>	<code>=</code>	<code>114;</code>
<code>telCallbackClearMsg</code>	<code>=</code>	<code>63;</code>
<code>telCallbackNowMsg</code>	<code>=</code>	<code>216;</code>
<code>telCallbackSetMsg</code>	<code>=</code>	<code>215;</code>
<code>telCallPickupMsg</code>	<code>=</code>	<code>217;</code>
<code>telCALookupMsg</code>	<code>=</code>	<code>111;</code>
<code>telCAMsgHandMsg</code>	<code>=</code>	<code>112;</code>
<code>telCloseTermMsg</code>	<code>=</code>	<code>53;</code>
<code>telClrCAMsgHandMsg</code>	<code>=</code>	<code>113;</code>
<code>telClrDNMsgHandMsg</code>	<code>=</code>	<code>105;</code>
<code>telClrTermMsgHandMsg</code>	<code>=</code>	<code>55;</code>
<code>telConfEstMsg</code>	<code>=</code>	<code>231;</code>
<code>telConfPrepMsg</code>	<code>=</code>	<code>230;</code>
<code>telConfSplitMsg</code>	<code>=</code>	<code>213;</code>
<code>telConnectMsg</code>	<code>=</code>	<code>204;</code>
<code>telCountCAsMsg</code>	<code>=</code>	<code>110;</code>
<code>telCountDNsMsg</code>	<code>=</code>	<code>60;</code>
<code>telDeactivateMsg</code>	<code>=</code>	<code>7;</code>
<code>telDeflectCallMsg</code>	<code>=</code>	<code>208;</code>
<code>telDialDigitsMsg</code>	<code>=</code>	<code>205;</code>
<code>telDisposeMsg</code>	<code>=</code>	<code>1;</code>
<code>telDNDClearMsg</code>	<code>=</code>	<code>123;</code>
<code>telDNDDisposeMsg</code>	<code>=</code>	<code>101;</code>
<code>telDNDSetsMsg</code>	<code>=</code>	<code>122;</code>
<code>telDNEventsSuppMsg</code>	<code>=</code>	<code>106;</code>
<code>telDNLookupByIndexMsg</code>	<code>=</code>	<code>61;</code>
<code>telDNLookupByNameMsg</code>	<code>=</code>	<code>62;</code>
<code>telDNMsgHandMsg</code>	<code>=</code>	<code>104;</code>
<code>telDNSelectMsg</code>	<code>=</code>	<code>100;</code>
<code>telDropMsg</code>	<code>=</code>	<code>210;</code>

telEventMsg	=	5;
telForwardClearMsg	=	121;
telForwardSetMsg	=	120;
telGetCAFlagsMsg	=	202;
telGetCAInfoMsg	=	203;
telGetCAStateMsg	=	201;
telGetDisplayMsg	=	75;
telGetDNFlagsMsg	=	103;
telGetDNInfoMsg	=	102;
telGetHookswMsg	=	70;
telGetInfoMsg	=	57;
telGetVolumeMsg	=	72;
telHoldMsg	=	211;
telIdleMsg	=	50;
telIntercomMsg	=	222;
telMenuMsg	=	4;
telNewMsg	=	0;
telOpenTermMsg	=	51;
telOtherFeatImplMsg	=	65;
telOtherFeatListMsg	=	64;
telOtherFunctionMsg	=	67;
telPagingMsg	=	221;
telParkCallMsg	=	218;
telRejectCallMsg	=	207;
telResetTermMsg	=	52;
telResumeMsg	=	3;
telRetrieveMsg	=	212;
telRetrieveParkedCallMsg	=	219;
telSetDisplayMsg	=	76;
telSetHookswMsg	=	71;
telSetupCallMsg	=	115;
telSetVolumeMsg	=	73;
telSuspendMsg	=	2;

<code>telTermEventsSuppMsg</code>	<code>=</code>	<code>56;</code>
<code>telTermMsgHandMsg</code>	<code>=</code>	<code>54;</code>
<code>telToolFunctionsMsg</code>	<code>=</code>	<code>66;</code>
<code>telTransfBlindMsg</code>	<code>=</code>	<code>214;</code>
<code>telTransfEstMsg</code>	<code>=</code>	<code>233;</code>
<code>telTransfPrepMsg</code>	<code>=</code>	<code>232;</code>
<code>telVoiceMailAccessMsg</code>	<code>=</code>	<code>220;</code>

The rest of this section describes each of the messages your main code resource should accept.

telAcceptCallMsg

The Telephone Manager will send `telAcceptCallMsg` when an application requests that your tool accept the incoming call appearance `hCA`, which must be in the state `telCAOfferState`.

`telAcceptCallMsg` is intended for phone systems that “offer” incoming calls instead of, or in addition to, causing them to alert (“ring”) immediately, as in most systems.

`hTEL` is a handle to a telephone record for the user’s terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1`, `p2`, and `p3` are unused.

When done, your tool should return `noErr` if the request was handled.

telActivateMsg

The Telephone Manager will pass `telActivateMsg` or `telDeactivateMsg` to your tool when the application requires your tool to perform an action, such as installing or removing a menu from the menu bar in response to an activate or deactivate message.

`hTEL` is a handle to a telephone record for the user’s terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`hDN`, `hCA`, `p1`, `p2`, and `p3` are unused.

telAlertMsg

The Telephone Manager will send `telAlertMsg` when an application requests that your tool make a ringer attached to the terminal ring—for instance, so that the user can set the volume level of the ringer.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is a pointer to a 2-byte field; it specifies the volume level to which your tool should set the ringer. Valid values for `p1` are 0 through 100. If `p1` equals zero, your tool should leave the volume level unchanged. If the value of `p1` exceeds the maximum volume supported by your tool, set the volume level to your tool's maximum volume value, and return that value in the field referenced by `p1`.

`p2` specifies the alerting pattern of the ringer. Your tool should return `telAPattNotSupp` if it does not support the requested alerting pattern.

`hDN`, `hCA`, and `p3` are unused.

When done, your tool should return `noErr` if the request was handled.

telAnswerCallMsg

The Telephone Manager will send `telAnswerCallMsg` when an application requests that your tool answer the alerting or offered call appearance `hCA`. If the `castate` field of the call appearance equals `telCAofferState`, your tool should both accept the call appearance and answer it.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telCADisposeMsg

Your tool will receive `telCADisposeMsg` when an application requests that the Telephone Manager dispose of a handle to a call-appearance record.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1`, `p2`, and `p3` are unused.

When done, your tool should perform any necessary cleanup. However, it should not dispose of `hCA`, nor should it dispose any string pointers in the call-appearance record referenced by `hCA`. The Telephone Manager disposes of these.

△ **Important** Your tool should not drop a call appearance unless the tool receives the message `telDropMsg` or `telDNSelectMsg`. Refer to the descriptions of these messages for more information. △

telCAEventsSuppMsg

Your tool will receive `telCAEventsSuppMsg` when an application inquires which types of call appearance messages your tool supports.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` points to a field containing a 4-byte mask filled in by your tool. This mask indicates which types of call appearance messages your tool supports.

`hCA`, `p2`, and `p3` are unused.

The sample code provides a basic template into which you can code your tool's response to `telCAEventsSuppMsg`. When done, your tool should return 0 if the request was handled.

```
FUNCTION myTELCAEventsSupp (hDN : TELDNHandle; VAR eventMask : LONGINT) : INTEGER;
VAR
    err      :   OSErr;
BEGIN
    myTELCAEventsSupp:= noErr;
    { somewhere earlier you did this...
      myDNPrivates^.mycaMessagesSupported := telCAActiveMsg + telCAProgressMsg +
      telCAConferenceMsg + etc. }
    eventMask := myDNPrivates^.mycaMessagesSupported;
END;
```

telCallbackClearMsg

Your tool will receive `telCallbackClearMsg` when an application requests that a previous callback be cleared from the user's terminal.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is a callback reference value.

`hCA`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled. If no callback is set, your tool should return `telCBErr`. If your tool requires a callback reference, and `p1` is invalid, your tool should return `telNoCallbackRef`.

telCallbackNowMsg

The Telephone Manager will send `telCallbackNowMsg` when an application requests that your tool call back a remote terminal, regardless of who set the callback or why. Your tool is responsible for checking whether the callback request is valid.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1` is a callback reference value. If your tool has more than one outstanding callback, `p1` should specify which destination is to be called back. (Your tool specified this value previously, when processing the corresponding message `telCallbackSetMsg`.)

`p2` and `p3` are unused.

When done, your tool should return 0 if the request was handled. If no callback is set, your tool should return `telCBErr`. If your tool requires a callback reference, and `p1` is invalid, your tool should return `telNoCallbackRef`.

The series of occurrences that precede a `telCallbackNowMsg` message differ depending on whether the terminal called was busy or unanswered. If one terminal (A) called another (B), and Terminal B was busy, the following steps preceded `telCallbackNowMsg`. First, the application on Terminal A called the routine `telCallbackSet`, causing your tool to receive the message `telCallbackSetMsg`. Next, when Terminal B was no longer busy, your tool sent the message `callbackNowMsg`, which the Telephone Manager then relayed to the application. Finally, the application called the routine `telCallbackNow`, causing your tool to receive `telCallbackNowMsg`. Your tool should now call back Terminal B.

On some systems, a "call back on no answer" works as follows: First, the application on Terminal A called the routine `telCallbackSet`, causing your tool to receive the message `telCallbackSetMsg`. Then, when the network switch notified Terminal B that a callback had been set, your tool received the message `telCallbackNow`, which the Telephone Manager relayed to the application. Finally, the application called the routine `telCallbackNow`, causing your tool on Terminal B to receive `telCallbackNowMsg`. Your tool on Terminal B should now call back Terminal A.

telCallbackSetMsg

The Telephone Manager will send `telCallbackSetMsg` when an application requests that your tool set a callback against the remote directory number specified in call appearance `hca`. When done, your tool should return 0 if the request was handled.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hca` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1` points to a 2-byte callback reference value. If the telephone network switch returns a callback reference, you should return it in the field referenced by `p1`.

`p2` and `p3` are unused.

telCallPickupMsg

The Telephone Manager will send `telCallPickupMsg` when an application requests that your tool pick up a call alerting either in a predefined pickup group (`p2`) or at a specified directory number (`p1`). Your tool should support `p1` or `p2`, depending on the kind of network switch your tool supports.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` points to a string of type `str255` that stores the remote directory number where the call to be picked up is alerting.

`p2` points to a string of type `str255` that stores a pickup group ID.

`hca` and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telCALookupMsg

Your tool will receive `telCALookupMsg` when the application wants to obtain a handle to the n th call appearance of a specified type.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hca` is a handle to the call-appearance record for the n th call appearance of type `p1`.

`msg` identifies the message.

`p1` specifies the type of call appearance—internal only (0), external only (1), or all (2).

`p2` is the index: it specifies the n th call appearance. If `telCountCAsMsg` returned the value 5, then valid index values are 1, 2, 3, 4, 5, for identical values of `p1` and `p2`.

`p3` is unused.

When done, your tool should return 0 if the request was handled. If `p2` is invalid, your tool should return `telBadIndex`.

If the index is valid, your tool should fill in the following fields: `caState`, `intExt`, `callType`, `dialType`, `bearerType`, `rate`, `confLimit`, `featureFlags`, `otherFeatures`, and `telCAPrivate`. Your tool should not, however, fill in fields of type `StringPtr`. Such fields are updated by the Telephone Manager when your tool sends `telCAProgressMsg` messages of type `telCAPUpdate` and `telCAPRouted`. Your tool should not change the value of `caRef`, except as warranted by error conditions.

For any error condition, the tool should place -1 in the `caRef` field. In addition, if an invalid index value was passed, the tool should return `telBadIndexErr`. Or, if your tool does not support the call-appearance type specified by `p1`, it should return `telBadCAtype` or `telIntExtNotSupp`.

telCAMsgHandMsg

The Telephone Manager will send `telCAMsgHandMsg` each time an application requests that your tool start sending messages on activity related to call appearances on a specified directory number.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hca` is unused.

`msg` identifies the message.

`p1` is the mask of events for which the tool should send messages.

`p2` is the address of the Telephone Manager's master call-appearance message handler.

`p3` points to the globals required by the master message handler `p2`.

The Telephone Manager distributes each tool message to all application message handlers registered for that type of message.

The message mask passed in `p1` is a master mask. It is equal to the result of performing a logical OR operation on the event masks of all registered call-appearance message handlers. Although your tool can ignore the mask passed in `p1`, the Telephone Manager runs more efficiently if your tool sends only the message types specified in this mask.

The sample code provides a basic template into which you can code your tool's response to `telCMsgHandMsg`. When done, your tool should return `noErr` if the request was handled.

```
FUNCTION myTELCMsgHand(hDN : TELDNHandle; eventMask : LONGINT;
                      msgHandler: ProcPtr; globals: LONGINT) : INTEGER;
VAR
    err      :      OSErr;
BEGIN
    myTELCMsgHand:= noErr;

    {note that call-appearance message handlers are registered per directory number}
    myDNPrivates^.caEventMask := eventMask;
    myDNPrivates^.caMsgHandler := msgHandler;
    myDNPrivates^.caGlobals := globals;
END;
```

telCloseTermMsg

The Telephone Manager will send `telCloseTermMsg` when an application requests that your tool close the terminal drivers of the user's terminal. Your tool can update the fields of the telephone record when handling this request, but should not disconnect any calls.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`hDN`, `hCA`, `p1`, `p2`, and `p3` are unused.

The sample code provides a basic template into which you can code your tool's response to `telCloseTermMsg`. When done, your tool should return 0 if the request was handled. When the terminal is closed, you should also send a terminal message of type `telTermCloseMsg`. Your tool should keep a `usecount`, incrementing it each time `telOpenTermMsg` is received. If no one else is using the driver, it can be closed.

If your tool encounters errors, it can send a generic `telTermErrorMsg` and a specific `telTermOtherMsg` to aid in debugging. (For information about handling terminal messages, refer to the sections "Handling Messages" and "Routines Your Application Must Provide" in Chapter 2.)

```
FUNCTION myTELCloseTerm(hTEL: TELHandle, pTELTerm: TELTermPtr)
VAR
    err      :      OSErr;
BEGIN
    myTELCloseTerm:= noErr;
    myTermPrivates^.usecount := myTermPrivates^.usecount -1;
    if (myTermPrivates^.usecount > 0)
    BEGIN
        myTELCloseTerm := telTermStillNeeded;
        EXIT(myTELCloseTerm);
    END;
```

```

myTermPrivates^.buffer^.csCode := CloseTerminal;
err := PBControl(@myTermPrivates^.buffer, FALSE);
if (err <> noErr)
    myTELCloseTerm:= telBadTermErr;
END;

```

telClrCAMsgHandMsg

The Telephone Manager will send `telClrCAMsgHandMsg` when an application requests that your tool stop sending messages about call appearances for a particular directory number. This request affects only the requesting application. Other applications using the Telephone Manager will continue to receive messages about these call appearances.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` is the new mask of events desired by the Telephone Manager. If there are no more call-appearance message handlers for your tool, the event mask will be zero.

`p2` is the address of the Telephone Manager's master call-appearance message handler. `p2` may equal zero if `p1` equals zero.

`hCA` and `p3` are unused.

The sample code provides a basic template into which you can code your tool's response to `telClrCAMsgHandMsg`. When done, your tool should return 0 if the request was handled.

```

FUNCTION myTELClrCAMsgHand(hDN : TELDNHandle; eventMask : LONGINT;
                           msgHandler: ProcPtr) : INTEGER;
VAR
    err      : OSErr;
BEGIN
    {This is where the tool updates its
     private storage regarding which messages
     to send--through eventMask--and how to
     send them--through msgHandler }
    myTELClrCAMsgHand:= 0;
    myDNPrivates^.caEventMask := eventMask;
    myDNPrivates^.caMsgHandler := msgHandler;
END;

```

telClrDNMsgHandMsg

The Telephone Manager will send `telClrDNMsgHandMsg` to your tool when an application requests your tool to stop sending messages on activity related to this directory number. This request affects only the requesting application. Other applications using the Telephone Manager will continue to receive messages about this directory number.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` is the new mask of events desired by the Telephone Manager. If there are no more directory-number message handlers for your tool, the event mask will be zero.

`p2` is the address of the master directory-number message handler. `p2` can equal zero if `p1` equals zero.

`hca` and `p3` are unused.

The sample code provides a basic template into which you can code your tool's response to `telClrDNMsgHandMsg`. When done, your tool should return 0 if the request was handled.

```
FUNCTION myTELClrDNMsgHand(hDN : TELHandle; eventMask : LONGINT;
                           msgHandler: ProcPtr) : INTEGER;

VAR
    err      :      OSErr;
BEGIN
    myTELClrDNMsgHand:= noErr;
    myDNPrivates^.DNEventMask := eventMask;
    myDNPrivates^.DNMsgHandler := msgHandler;
END;
```

telClrTermMsgHandMsg

The Telephone Manager will send `telClrTermMsgHandMsg` to your tool when the application requests your tool to stop sending messages on activity related to this terminal. This request affects only the requesting application. Other applications using the Telephone Manager will continue to receive messages about the terminal.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is the new mask of events desired by the Telephone Manager. If there are no more terminal message handlers for your tool, the event mask will be zero.

`p2` is the address of the master terminal message handler. `p2` can equal zero if `p1` equals zero.

`hDN`, `hca`, and `p3` are unused.

The sample code provides a basic template into which you can code your tool's response to `telClrTermMsgHandMsg`. When done, your tool should return 0 if the request was handled.

```
FUNCTION myTELClrTermMsgHand(hTEL : TELHandle; eventMask : LONGINT;
                             msgHandler: ProcPtr) : INTEGER;

VAR
    err      :      OSErr;
BEGIN
    myTELClrTermMsgHand:= 0;
    myTermPrivates^.termEventMask := eventMask;
    myTermPrivates^.termMsgHandler := msgHandler;
END;
```

telConfEstMsg

Your tool will receive `telConfEstMsg` when an application requests that one call appearance (`hca`) be conferenced with another (`p1`).

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.
`hDN` is a handle to a directory-number record associated with terminal `hTEL`.
`hCA` is a handle to a call-appearance record (for the conference initiator) on directory number `hDN`.
`msg` identifies the message.
`p1` is a handle to a second call-appearance record, for the call being added to the conference.
`p2` and `p3` are unused.

Your tool should check that this `TELConfEstMsg` message was preceded by the message `TELConfPrepMsg` for call appearances `hCA` and `p1`. If so, your tool can unite the two call appearances in a conference. Otherwise, your tool can return an error message.

When done, your tool should return 0 if the request was handled. If `hCA1` was not properly prepared, your tool should return `telConfRej`.

telConfPrepMsg

The Telephone Manager will send `telConfPrepMsg` when your tool should prepare one call appearance to be conferenced with another specified by `p1`.

`hTEL` is a handle to a telephone record for the user's terminal.
`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.
`hDN` is a handle to a directory-number record associated with terminal `hTEL`.
`hCA` is a handle to a call-appearance record (for the conference initiator) on directory number `hDN`.
`msg` identifies the message.

`p1` is a handle to a second call-appearance record, for the call being added to the conference. (Your tool should previously have received the message `TELSetupCallMsg` to set up this call.)
`p2` is the total number of calls that will make up an entire conference. If the network switch requires that this number be sent explicitly, your tool should already have set the `numToConferenceRequired` bit of the `otherFeatures` field in call-appearance record `hCA`. If the switch does not require this number, your tool should ignore `p2`.

`p3` is unused.

If the second call appearance is not in an active state, the tool should proceed as with `TELConnectMsg` to get the call appearance to an active state.

When done, your tool should return 0 if the request was handled.

telConfSplitMsg

Your tool will receive `telConfSplitMsg` when an application requests that a call appearance be removed from a current conference. Your tool should not drop this call appearance, but should only remove it from the conference, allowing it to exist on its own.

`hTEL` is a handle to a telephone record for the user's terminal.
`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.
`hDN` is a handle to a directory-number record associated with terminal `hTEL`.
`hCA` is a handle to a call-appearance record (for the call to be removed) on directory number `hDN`.
`msg` identifies the message.
`p1`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telConnectMsg

Your tool will receive `telConnectMsg` when the application requests that a specified call appearance proceed to an active state (`telCAActiveState`), if possible. Your tool should use the information in call-appearance record `hCA` to place the outgoing call.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1`, `p2`, and `p3` are unused.

Your tool must allow directory numbers that include the following characters: the digits 0 through 9, the number sign (#), the asterisk (*), the comma (,), and the exclamation point (!). The exclamation point should be treated as a flash-hook. All other characters should be ignored. For example, your tool should dial the directory number 1 (408) 555-1212 as 4085551212.

If the destination directory number includes a comma, your tool should pause for one second before dialing subsequent characters. An exclamation point in the destination directory number signifies flashhook.

When done, your tool should return 0 if the request was handled.

telCountCAsMsg

Your tool will receive `telCountCAsMsg` when the application inquires how many call appearances are associated with a particular directory number. Your tool should count only call appearances whose state is not `telCAIdleState`.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` specifies which call appearances should be counted—internal only (0), external only (1), or all (2).

`hCA`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled. If your tool does not support the value of `p1`, it should return the result code `telIntExtNotSupp`.

telCountDNsMsg

Your tool will receive `telCountDNsMsg` when the application inquires how many directory numbers are associated with the user's terminal.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` specifies which directory numbers should be counted—internal only (0), external only (1), or all (2).

`p2` specifies whether logical directory numbers are to be counted. If `p2` equals zero, all directory numbers (logical and physical) of type `p1` should be counted. If `p2` equals 1, only physical directory numbers of type `p1` should be counted.

`hDN`, `hCA`, and `p3` are unused.

When done, your tool should return 0 if the request was handled. If your tool does not support the value in `p1`, it should return the result code `telBadDNType`.

telDeactivateMsg

Refer to the description of `telActivateMsg`.

telDeflectCallMsg

Your tool will receive `telDeflectCallMsg` when the application requests that an incoming call appearance on one directory number be deflected to another directory number. The state of the incoming call appearance must be either `telCAOfferState` or `telCAAlertingState`.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1` points to a string of type `str255` storing the destination directory number, to which the call appearance will be deflected.

`p2` points to a string of type `str255` storing the subaddress of the destination directory number. If your tool does not support subaddressing, it should ignore `p2`.

`p3` points to a string of type `str255` storing the name of the party associated with directory number `p1`.

Your tool should save the values of `p1`, `p2`, and `p3`, and should pass them back in the message `telCADeflectMsg`.

When done, your tool should return 0 if the request was handled.

If your telephone network switch does not support a Deflect feature, your tool can mimic this feature by performing a blind transfer. To do so, your tool should first answer the call and then immediately transfer it to directory number `p1`.

telDialDigitsMsg

The Telephone Manager will send `telDialDigitsMsg` when an application requests that your tool transmit specified network characters over a call appearance. (For a list of valid network characters, refer to the description of `telConnectMsg`.)

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1` points to a string of type `str255` storing the network characters to be transmitted.

`p2` and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telDisposeMsg

Your tool will receive `telDisposeMsg` when an application requests that the Telephone Manager dispose of the handle to a telephone record. Your tool cannot stop the Telephone Manager from disposing of the handle, but should perform any necessary cleanup now.

△ **Important** Unless your tool receives the message `telDropMsg` or `telDNSelectMsg`, it should not drop a call appearance. Refer to the descriptions of those messages for more information. △

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.
`msg` identifies the message.

`hDN`, `hCA`, `p1`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telDNDClearMsg

Your tool will receive `telDNDClearMsg` when an application requests that an active Do Not Disturb feature be cleared on a specified directory number.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` specifies the type of Do Not Disturb feature being cleared.

`hCA`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled, or `telDNDDTypeNotSupp` if it does not support the value in `p1`.

telDNDisposeMsg

Your tool will receive `telDNDisposeMsg` when the application has requested that the Telephone Manager dispose of a handle to a directory-number record. Your tool cannot stop the Telephone Manager from disposing of the handle, but should perform any necessary cleanup now. Also at this time, your tool should dispose of the string pointers `rmtDN`, `rmtSubaddress`, and `rmtPartyName`, but no others.

△ **Important** Unless your tool receives the message `telDropMsg` or `telDNSelectMsg`, it should not drop a call appearance. Refer to the descriptions of those messages for more information. △

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`hCA`, `p1`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telDNDSetMsg

The Telephone Manager will send `telDNDSetMsg` when your tool should set (activate) a specified Do Not Disturb feature on a particular directory number.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` specifies the kind of Do Not Disturb feature to be set.

`hCA`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled, or `telDNDTypeNotSupp` if it does not support the value in `p1`.

telDNEventsSuppMsg

The Telephone Manager will send `telDNEventsSuppMsg` when an application inquires what type of directory-number messages your tool supports.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` points to a 4-byte mask of message types, to be filled in by your tool.

`hDN`, `hCA`, `p2`, and `p3` are unused.

The sample code provides a basic template into which you can code your tool's response to `telDNEventsSuppMsg`. When done, your tool should return 0 if the request was handled.

```
FUNCTION myTELDNEventsSupp (hDN : TELDNHandle; VAR eventMask : LONGINT) : INTEGER;
VAR
    err      : OSErr;
BEGIN
    myTELDNEventsSupp := 0;
    { somewhere earlier you did this...
    myDNPrivates^.myDNMessagesSupported := telDNForwardMsg + DNDMsg;
      + etc }
    eventMask := myDNPrivates^.myDNMessagesSupported;
END;
```

telDNLookupByIndexMsg

Your tool will receive `telDNLookupByIndexMsg` when the application requests a handle to the *n*th directory number of the user's terminal.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` specifies the type of directory numbers to be counted—internal only (0), external only (1), or all (2).

`p2` specifies whether logical directory numbers are to be counted. If `p2` is zero, directory numbers, logical and physical, of type `p1` are counted. If `p2` is 1, only physical directory numbers of type `p1` are counted.

`p3` is the index. It specifies the *n*th call directory number. If `telCountDNsMsg` returned the value 5, then valid index values are 1, 2, 3, 4, 5, for identical values of `p1` and `p2`.

If the index is valid, your tool should fill in the following fields referenced by `hDN`: `dn`, `dnRef`, `dnPartyName`, `dnSubaddress`, `maxAllocCA`, `curAllocCA`, `dnType`, `featureFlags`, `forwarddFlags`, `numPageIDs`, `numIntercomIDs`, `numPickupIDs`, and `telDNPrivate`.

After setting the value of `dnRef`, your tool should not change it. Also, your tool should not update the fields relating to forwarding numbers, subaddresses, or party names. These fields are updated each time the Telephone Manager receives the message `telDNForwardMsg` from your tool.

`hCA` is unused.

When done, your tool should return 0 if the request was handled. If an invalid index value was passed, the tool should return `telBadIndexErr`. If the tool does not support the `dnType` specified by `p1`, it should return `telDNTypeNotSupp`. For any error condition, the tool should place -1 in the `dnRef` field of directory-number record `hDN`. The Telephone Manager will then dispose of `hDN`.

telDNLookupByNameMsg

Your tool will receive `telDNLookupByNameMsg` when the application requests a handle to the directory number specified by `name`.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` points to a string of type `str255` storing the directory number to be looked up. If the directory number is invalid, the tool should set the value in `dnRef` to -1 and return `telBadDNerr`.

If the value of `p1` is valid, the tool should fill in the following fields referenced by `hDN`: `dn`, `dnRef`, `dnPartyName`, `dnSubaddress`, `maxAllocCA`, `curAllocCA`, `dnType`, `featureFlags`, `forwarddFlags`, `numPageIDs`, `numIntercomIDs`, `numPickupIDs`, and `telDNPrivate`.

After setting the value of `dnRef`, your tool should not change it. Also, your tool should not update the fields relating to forwarding numbers, subaddresses, or party names. These fields are updated each time the Telephone Manager receives the message `telDNForwardMsg` from your tool.

`hCA`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telDNMsgHandMsg

The Telephone Manager will send `telDNMsgHandMsg` when an application requests that your tool start sending messages on activity related to this directory number.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` stores the mask of events for which the tool should send messages.

`p2` stores the address of the Telephone Manager's master directory-number message handler.

`p3` stores the globals required by the Telephone Manager's message handler.

`hca` is unused.

Please note that the address of the message handler installed by the application is not the same one passed by the application when calling `TELDNMsgHand`. All tools send all messages to master message handlers that are part of the Telephone Manager. The Telephone Manager takes your message and distributes it to all message handlers that have registered for that message.

The message mask passed in `p1` is a master mask. It is equal to the result of performing a logical OR operation on the event masks of all registered directory-number message handlers. Although your tool can ignore the mask passed in `p1`, the Telephone Manager runs more efficiently if your tool sends only the message types specified in this mask.

The sample code shows a template into which you can code your tool's response to `telDNMsgHandMsg`. When done, your tool should return 0 if the request was handled.

```
FUNCTION myTELDNMsgHand(hDN : TELDNHandle; eventMask : LONGINT;
                        msgHandler: ProcPtr; globals: LONGINT) : INTEGER;
VAR
    err      :      OSErr;
BEGIN
    myTELDNMsgHand:= noErr;
    myDNPrivates^.dnEventMask := eventMask;
    myDNPrivates^.dnMsgHandler := msgHandler;
    myDNPrivates^.dnGlobals := globals;
END;
```

telDNSelectMsg

The Telephone Manager will send `telDNSelectMsg` when an application requests that your tool select or deselect a particular directory number for use.

All tools should handle this message.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` specifies whether the directory number is to be selected or deselected. If `p1` equals zero, the directory number is deselected; if `p1` equals 1, the number is selected.

`hca`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

Selecting a directory number is the programmatic equivalent of pushing a directory-number button on a physical telephone set having multiple lines. If your tool receives a second or subsequent `telDNSelectMsg` message for a different directory number, it should put on hold any call appearances on the currently selected directory number and then select the new directory number.

Deselecting a directory number is the equivalent of dropping (hanging up) all call appearances on that directory number. After deselecting a number, your tool should not automatically select another number.

telDropMsg

Your tool will receive `telDropMsg` when an application requests that a call appearance be dropped (hung up).

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1` points to a string of type `Str255` storing any user-to-user information to be passed at the time of the drop. If your tool does not support user-to-user information, this field should be ignored.

`p2` and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telEventMsg

The Telephone Manager will pass `telEventMsg` to your tool when an event occurs in a window associated with the telephone tool. The sample code shows a template into which you can code your tool's response to `telEventMsg`.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` points to the event record.

`hDN`, `hCA`, `p2`, and `p3` are unused.

The reference constant field of the window record will contain the handle to the telephone record.

```
PROCEDURE myEvent(hTEL : TELHandle; theEvent : EventRecord);
CONST
    CancelButton = 2;
VAR
    theDialog      : DialogPtr;
    theItem        : INTEGER;
BEGIN
    { Check if it is a dialog-related event }
    if IsDialogEvent(theEvent) then
        begin
            { get the item hit }
            if DialogSelect(theEvent, theDialog, theItem) then
                begin
                    if theItem = CancelButton then
                        { Cancel the connection }
                    end;
                end
            else
                { Handle the keyDown, updateEvt, mouseDown and any other event here }
            end;
        end;
END;
```

telForwardClearMsg

Your tool will receive `telForwardClearMsg` when the application requests that a particular Forward feature be cleared on a specified directory number.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p2` is an integer specifying the type of forwarding to be cleared—for instance, Forward On No Answer.

`hCA`, `p1`, and `p3` are unused.

When done, your tool should return 0 if the request was handled, or `telFwdTypeNotSupp` if it does not support the value in `p2`. Your tool should not clear the strings referenced by directory-number record `hDN`. The Telephone Manager clears these.

telForwardSetMsg

Your tool will receive `telForwardSetMsg` when an application requests that a specified directory number be forwarded to another number.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`p1` points to the following parameter block:

```
TELForwardPB          =      RECORD
    forwardDN          :      StringPtr;
    forwardPartyName   :      StringPtr;
    forwardSubaddress  :      StringPtr;
    forwardType        :      INTEGER;
    numRings           :      INTEGER;
    END;
```

In this block, `forwardDN` is the directory number to which calls will be forwarded.

`forwardPartyName` is the name of the party associated with `forwardDN`. `forwardSubaddress` is the subaddress associated with `forwardDN`. `forwardType` is the type forwarding requested—for example, Forward On No Answer. `numRings` is used only if `forwardType` specifies Forward On No Answer; it is the number of times the telephone should ring before forwarding occurs.

`hCA`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled, or `telFwdTypeNotSupp` if your tool does not support the value in `forwardType`. Your tool should not update the fields in `hDN` that relate to forwarding. These fields are updated each time the Telephone Manager receives the message `telDNForwardMsg` from your tool.

telGetCAFlagsMsg

The Telephone Manager will send `telGetCAFlagsMsg` when an application requests that your tool update the fields `featureFlags` and `otherFeatures` in a specified call-appearance record and that your tool return the updated values of those fields.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`. This is the call appearance whose `featureFlags` and `otherFeatures` fields are to be updated.

`msg` identifies the message.

`p1` points to a 4-byte field that stores the updated value of the `featureFlags` field of call-appearance record `hCA`. Your tool should update this field.

`p2` points to a 4-byte field that stores the updated value of the `otherFeatures` field of call-appearance record `hCA`. Your tool should update this field.

`p3` is unused.

When done, your tool should return 0 if the request was handled. If call-appearance record `hCA` is invalid, your tool should return `telBadCAErr`.

telGetCAInfoMsg

The Telephone Manager will send `telGetCAInfoMsg` when an application requests that your tool update the values in a specified call-appearance record.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1`, `p2`, and `p3` are unused.

Your tool should update the following fields: `caState`, `intExt`, `callType`, `dialType`, `bearerType`, `rate`, `confLimit`, `featureFlags`, `otherFeatures`, and `telCAPrivate`.

Your tool should keep the information for each call-appearance record in the tool's private storage. By doing so, your tool maintains information about the call appearance. Your tool should update the call-appearance record only at the application's request.

The sample code provides a basic template into which you can code your tool's response to `telGetCAInfoMsg`. When done, your tool should return 0 if the request was handled. If the call appearance has become idle, the tool should put -1 in the `caRef` field of call-appearance record `hCA` and return `telBadCAErr`.

```
PROCEDURE myTELGetCAInfo(hCA: TELCAHandle)
```

```
BEGIN
```

```
  WITH CH^^ DO
```

```
  BEGIN
```

```
    caRef := myCAprivates^.reference;
```

```
    featureFlags := conferenceSub + transferSub + holdSub;
```

```
    otherFeatures := whatever;
```

```
    { tool should also fill in }
```

```
  END;
```

```
END;
```

telGetCAStateMsg

The Telephone Manager will send `telGetCAStateMsg` when an application requests that your tool update the `caState` field in a specified call-appearance record and return the updated value of that field.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1` points to a 2-byte field that stores the updated value of the `caState` field of call-appearance record `hCA`. Your tool should update this field.

`p2` and `p3` are unused.

A tool should update the `caState` field of call-appearance record `hCA` only at the application's request.

The sample code provides a basic template into which you can code your tool's response to `telGetCAStateMsg`. When done, your tool should return 0 if the request was handled. If the `hCA` references an idle call appearance, the `caRef` should be set to -1 and return `telBadCAErr`.

```
PROCEDURE myTELGetCAState(hCA: TELCAHandle; VAR p1: INTEGER)
```

```
BEGIN
```

```
    hCA^.caState := myprivates^.lateststate;
```

```
    p1 := myprivates^.lateststate;
```

```
END;
```

telGetDisplayMsg

Your tool will receive `telGetDisplayMsg` when an application requests the current display text of the terminal.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is the index. It specifies which item of the display is to be retrieved.

`p2` is a pointer to a text string allocated by your tool.

`p3` is a pointer to a 2-byte value that is the display mode. Your tool should set `p3` to the current display mode.

`hDN` and `hCA` are unused.

When done, your tool should return 0 if the request was handled, or `telIndexNotSupp` if your tool does not support the value in `p1`.

telGetDNFlagsMsg

The Telephone Manager will send `telGetDNFlagsMsg` when an application requests that your tool update the fields `featureFlags` and `forwardFlags` in a specified directory-number record and that your tool return the updated values of those fields.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`. This is the directory-number record whose `featureFlags` and `forwardFlags` fields are to be updated.
`msg` identifies the message.

`p1` points to a 4-byte field that stores the updated value of the `featureFlags` field of directory-number record `hDN`. Your tool should update this field.

`p2` points to a 4-byte field that stores the updated value of the `forwardFlags` field of directory-number record `hDN`. Your tool should update this field.

`hCA` and `p3` are unused.

When done, your tool should return 0 if the request was handled. If directory-number record `hDN` is invalid, your tool should return `telBaddNErr`.

telGetDNInfoMsg

The Telephone Manager will send `telGetDNInfoMsg` when an application requests that your tool update the values in the record referenced by `hDN`.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`msg` identifies the message.

`hCA`, `p1`, `p2`, and `p3` are unused.

Your tool should fill in the following fields referenced by `hDN`: `dn`, `dnRef`, `dnPartyName`, `dnSubaddress`, `maxAllocCA`, `curAllocCA`, `dnType`, `featureFlags`, `forwarddFlags`, `numPageIDs`, `numIntercomIDs`, `numPickupIDs`, and `telDNPrivate`.

Your tool should keep the information for each directory-number record in the tool's private storage. By doing so, your tool maintains information about the directory number. Your tool should update the directory-number record only at the application's request.

The sample code provides a basic template into which you can code your tool's response to `telGetDNInfoMsg`. When done, your tool should return 0 if the request was handled. If the directory number is invalid, the tool should set the `dnRef` field to -1 and return `telBaddNErr`.

```
PROCEDURE myTELGetDNInfo(hDN: TELDNHandle)
```

```
BEGIN
```

```
    WITH hDN^^ DO
```

```
    BEGIN
```

```
        dnRef := myDNprivates^.reference;
```

```
        directory number := myDNprivates^.dnName;
```

```
        featureFlags := dndSub + intercomSub + dnSelectSub;
```

```
        forwardFlags := immediateForwardSub + busyForwardSub;
```

```
        { tool should also fill in partyName, maxAllocCA, curAllocCA,
          dnType, numPageIDs, numIntercomIDs }
```

```
    END;
```

```
END;
```

telGetHookswMsg

Your tool will receive `telGetHookswMsg` when an application inquires about the current physical hook state of a device attached to the terminal managed by your tool.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is the hookswitch type.

`p2` is pointer to a Boolean variable that specifies whether the device is on- or off-hook. Your tool updates this value.

`hDN`, `hCA`, and `p3` are unused.

`p1` specifies the device. You should return the state in the field referenced by `p2`.

When done, your tool should return 0 if the request was handled.

telGetInfoMsg

The Telephone Manager will send `telGetInfoMsg` when an application requests that your tool update the values in the record referenced by `hTEL`.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`hDN`, `hCA`, `p1`, `p2`, and `p3` are unused.

Your tool should fill in the field `telPrivate` and all the fields of `pTELTerm`.

Your tool should keep the information for each telephone record in the tool's private storage, and should update the record only at the application's request.

The sample code provides a basic template into which you can code your tool's response to `telGetInfoMsg`. When done, your tool should return 0 if the request was handled. If the handle references an invalid terminal, the tool should set the `tRef` field to -1 and return `telBadTermErr`.

```
PROCEDURE myTELGetInfo(hTEL: TELHandle, pTELTerm: TELTermPtr)
```

```
BEGIN
```

```
  th^.version := curTELVersion;
```

```
  { other fields are all filled in by Telephone Manager or app! }
```

```
  { but there's some stuff to fill in the TELTermRecord }
```

```
  WITH pTELTerm^ DO
```

```
    BEGIN
```

```
      tRef := slotnum;
```

```
      featureFlags := hasHandset + hasRinger + canOnHkDial;
```

```
      handsetSpeakerVol := myHandsetVolMax;
```

```
      ringerVol := myRingerVolMax;
```

```
      numDNs := myDNsConfigured;
```

```
      { etc., etc. }
```

```
    END;
```

```
END;
```

telGetVolumeMsg

The Telephone Manager will send `telGetVolumeMsg` when an application inquires the current volume setting of a device attached to the terminal managed by your tool.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is the volume type.

`p2` points to a 2-byte field that specifies the current volume level. Your tool updates this value.

`p3` points to a 2-byte field that specifies the current state of the device. Your tool updates this value.

`hDN` and `hCA` are unused.

Your tool should check the value of `p1`. If the value is supported, your tool should place the volume level in the field referenced by `p2` and place the device state in the field referenced by `p3`.

When done, your tool should return 0 if the request was handled, or `telVolTypeNotSupp` if your tool does not support the value in `p1`.

telHoldMsg

Your tool will receive `telHoldMsg` when the application requests that a specified call appearance (`hCA`) be put on hold.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telIdleMsg

The Telephone Manager will send `telIdleMsg` when an application is giving your tool time to do idle processing. This time should be used to check on the progress of current calls and to check for the presence of new incoming calls. If progress occurs or calls come in, messages should be sent to the master message handler of the Telephone Manager.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`hDN`, `hCA`, `p1`, `p2`, and `p3` are unused.

When done your tool should return 0 or, if it detects a problem, the message

`telTermErrorMsg`.

telIntercomMsg

Your tool will receive `telIntercomMsg` when the application requests that an intercom function be activated.

`hTEL` is a handle to a telephone record for the user's terminal.

pTELterm points to a record of type **TELtermRecord**, part of the telephone record **hTEL**.
hDN is a handle to a directory-number record associated with terminal **hTEL**.
hCA is a handle to a call-appearance record on directory number **hDN**.
msg identifies the message.
p1 is an intercom ID that specifies which intercom function the tool should activate. (The application is responsible for labeling the intercom keys.)
p2 and **p3** are unused.
 If your tool supports multiple intercom functions, you should specify how many in the **numIntercomIDs** field of directory-number record **hDN**.
 When done, your tool should return 0 if the request was handled. If your tool requires an intercom ID and the value in **p1** is invalid, your tool should return **telBadIntercomID**.

telMenuMsg

Your tool will receive **telMenuMsg** when a menu event has occurred in the application.

hTEL is a handle to a telephone record for the user's terminal.
pTELterm points to a record of type **TELtermRecord**, part of the telephone record **hTEL**.
msg identifies the message.
p1 contains the menu ID.
p2 contains the menu item.
hDN, **hCA**, and **p3** are unused.
 The sample code provides a basic template into which you can code your tool's response to **telMenuMsg**. When done, your tool should return 0 if the menu event was not handled, and 1 if it was.

```

FUNCTION myMenu(hTEL : TELHandle; mID : INTEGER; mItem: INTEGER) : LONGINT;
BEGIN
    myMenu := 0;
    { if mine then
      begin
        myMenu := 1;
        Process the menu command.
      end;
    }
END;
```

telNewMsg

Your tool will receive **telNewMsg** when the application requests that the Telephone Manager create a new telephone record.

hTEL is a handle to a telephone record for the user's terminal.
pTELterm points to a record of type **TELtermRecord**, part of the telephone record **hTEL**.
msg identifies the message.
p1 is the tool ID, a unique identifier that the Telephone Manager assigns to your tool. Your tool should pass the tool ID when sending messages to the Telephone Manager.
hDN, **hCA**, **p2**, and **p3** are unused.

Your tool should fill in any fields that it can in the telephone record specified by `hTEL` and `pTELTerm`. Your tool may need to open the terminal driver now, before receiving the message `telOpenTermMsg`, to get the configuration information needed to fill in the telephone record.

When done, your tool should return 0 if the request was handled.

It is possible to receive `telNewMsg` more than once—for instance, if more than one application is using the Telephone Manager. As a convenience, in all `telNewMsg` messages except the first, the Telephone Manager places a pointer to your tool's private storage in the `telPrivate` field of telephone record `hTEL`. But in the first `telNewMsg` message, `telPrivate` equals zero.

Thus, instead of initializing private storage for each `telNewMsg` message, your tool can, optionally, check the `telPrivate` field and initialize the private storage only if `telPrivate` equals zero.

telOpenTermMsg

The Telephone Manager will send `telOpenTermMsg` when an application requests that your tool open any underlying terminal drivers and ready the terminal for use.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`. `msg` identifies the message.

`hDN`, `hCA`, `p1`, `p2`, and `p3` are unused.

Your tool can update the information in `pTELTerm` at this time if any values have changed.

The sample code provides a basic template into which you can code your tool's response to `telOpenTermMsg`. When done, your tool should return 0 if the request was handled. Also, when the terminal is open, your tool should send terminal messages of type `telTermOpenMsg` and `telTermEnableMsg`, and should increment a use-count variable for the terminal.

If you encounter errors, you may choose to send a generic `telTermErrorMsg` and a specific `telTermOtherMsg` to aid in debugging. (For information about handling terminal messages, refer to the sections "Handling Messages" and "Routines Your Application Must Provide" in Chapter 2.)

```
PROCEDURE myTELOpenTerm(hTEL: TELHandle, pTELTerm: TELTermPtr)
VAR
    err      :      OSErr;
BEGIN
    myTELOpenTerm := noErr;
    err := OpenDriver(".COOLPHONE", @myTermPrivates^.driverRefNum);
    if (err <> noErr)
    BEGIN
        myTELOpenTerm := telBadTermErr;
        myTermPrivates^.usecount := myTermPrivates^.usecount + 1;
    END;
END;
```

telOtherFeatImplMsg

Your tool will receive `telOtherFeatImplMsg` when an application requests that a feature returned by a call to `TELOtherFeatureList` is to be executed.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is a handle to the Telephone Manager data structure (for instance, a call-appearance record) needed to implement the feature. Your tool should check that `p1` is a valid handle.

`p2` is the feature ID of the requested feature.

`hDN`, `hCA`, and `p3` are unused.

When done, your tool should return 0 if the request was handled. Because the requested feature is specific to your tool, your tool is responsible for all processing of the request. If an error occurs, your tool should return a Telephone Manager result code.

telOtherFeatListMsg

The Telephone Manager will send `telOtherFeatListMsg` when the application requests a list of the features supported by your tool but for which there is no specific Telephone Manager message. Your tool should provide the list as a linked list of type `FeatureListPtr`. The application can then display the list items to the user.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is pointer of type `FeatureListPtr`. It points to a list of features.

`hDN`, `hCA`, `p2`, and `p3` are unused.

The features in the list should be simple ones that require only a feature descriptor and a handle to a telephone record, directory-number record, or call-appearance record.

The advantage of this message over `telOtherFunctionMsg` is that any application can display your tool's supplementary features.

When done, your tool should return 0 if the request was handled. If an error occurs, your tool should return a Telephone Manager result code.

telOtherFunctionMsg

The Telephone Manager will send `telOtherFunctionMsg` when the application requests that a tool-specific feature be invoked.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELterm` points to a record of type `TELtermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is a pointer to a parameter block.

`p2` is the size of the parameter block (in bytes).

`hDN`, `hCA`, and `p3` are unused.

Note that you must document the parameter block passed in `p1` and the value passed in `p2`, as well as any result codes.

When done, your tool should return 0 if the request was handled.

telPagingMsg

Your tool will receive `telPagingMsg` when the application requests that a paging function be activated.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1` specifies which page ID the tool should implement. (The application is responsible for labeling the paging keys.)

`p2` and `p3` are unused.

If your tool supports multiple page functions, it should specify how many such functions in the `numPageIDs` field of the directory-number record `hDN`.

When done, your tool should return 0 if the request was handled. If the application specified an invalid page ID, your tool should return `telBadPageID`.

telParkCallMsg

Your tool will receive `telParkCallMsg` when the application requests that a specified call appearance be parked.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to the active call appearance that will be parked.

`msg` identifies the message.

`p1` points to a string of type `str255`; it stores the ID used to park the call. This value is filled in by your tool.

`p2` points to a string of type `str255`; it stores the directory number at which the call will be parked.

`p3` is unused.

When done, your tool should return 0 if the request was handled.

The Telephone Manager supports two types of parking. In the first, the application provides a park ID, usually a directory number, in `p2`; and the call is parked with that ID. The application can then retrieve the call from that directory number by means of the Telephone Manager routine `TELRetrieveParkedCall`. When using this method, the tool should set the `parkWithID` bit in the call-appearance record `hCA`.

In the second method, your tool parks the call and returns `parkRetrieveID`. The application can then use the `parkRetrieveID` to retrieve the call from any telephone connected to the network switch. When using this method, the tool should set the `parkRetrieveWithID` bit in the `otherFeatures` field of the call-appearance record.

In the third method, both a `parkID` and a `parkRetrieveID` are required. Typically, the user must enter a one- to four-digit code (the `parkID`) and retrieve it with the same ID.

telRejectCallMsg

Your tool will receive `telRejectCallMsg` when an application requests that your tool reject a specified call appearance. This call appearance must be in either the state `telCAAlertingState` or `telCAOfferState`.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.
`hCA` is a handle to a call-appearance record on directory number `hDN`. In this call-appearance record, your tool should set the `rejectable` bit if it can reject calls.

`msg` identifies the message.

`p1`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled. If the call appearance is not in an alerting state, your tool should not reject the call, but instead should return the result code `telCANotRejectable`.

telResetTermMsg

The Telephone Manager will send `telResetTermMsg` when an application requests that your tool reset (close and then open) any underlying terminal drivers. When handling this request, your tool can, optionally, soft-boot any underlying hardware. Your tool can also update the information in the telephone record if any values have changed.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`hDN`, `hCA`, `p1`, `p2`, and `p3` are unused.

By handling this request, your tool should clear the state of the terminal, its directory numbers, and their call appearances.

The sample code provides a basic template into which you can code your tool's response to `telResetTermMsg`. When done, your tool should return 0 if the request was handled. When the terminal is reset, your tool should also send a terminal message of type `telTermResetMsg`.

If you encounter errors, you may choose to send a generic `telTermErrorMsg` and a specific `telTermOtherMsg` to aid in debugging. (For information about handling terminal messages, refer to the sections "Handling Messages" and "Routines Your Application Must Provide" in Chapter 2.)

```
FUNCTION myTELResetTerm(hTEL: TELHandle, pTELTerm: TELTermPtr)
```

```
VAR
```

```
    err      :      OSErr;
```

```
BEGIN
```

```
    myTELResetTerm := noErr;
```

```
    myTermPrivates^.buffer.csCode := ResetTerminal;
```

```
    err := PBControl(@myTermPrivates^.buffer, FALSE);
```

```
    if (err <> noErr)
```

```
        myTELResetTerm := telBadTermErr;
```

```
END;
```

telResumeMsg

Refer to the description of `telSuspendMsg`.

telRetrieveMsg

Your tool will receive `telRetrieveMsg` when the application requests that the state of a specified call appearance be changed from held to active.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.
`hDN` is a handle to a directory-number record associated with terminal `hTEL`.
`hCA` is a handle to a call-appearance record on directory number `hDN`.
`msg` identifies the message.
`p1`, `p2`, and `p3` are unused.
When done, your tool should return 0 if the request was handled.

telRetrieveParkedCallMsg

The Telephone Manager will send `telRetrieveParkedCallMsg` when an application requests that your tool retrieve a parked call. When an application calls the Telephone Manager routine `telRetrieveParkedCall`, the Telephone Manager sends `telSetupCallMsg` to your tool before sending `telRetrieveParkedCallMsg`.

`hTEL` is a handle to a telephone record for the user's terminal.
`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.
`hDN` is a handle to a directory-number record associated with terminal `hTEL`.
`msg` identifies the message.
`p1` points to a string of type `Str255` storing a `parkRetrieveID`. If your tool requires a `parkRetrieveID`, it should set the `parkRetrieveID` bit of the `otherFeatures` field in the call-appearance record. Otherwise, your tool should ignore `p1`.
`hCA`, `p2`, and `p3` are unused.
When done, your tool should return 0 if the request was handled.

telSetDisplayMsg

Your tool will receive `telSetDisplayMsg` when an application requests that the display of the user's terminal be set.

`hTEL` is a handle to a telephone record for the user's terminal.
`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.
`msg` identifies the message.
`p1` is the index; it specifies which item of the display will be set.
`p2` is a pointer to a string of type `Str255` that contains the new text.
`p3` is the display mode to be set.
`hDN` and `hCA` are unused.
When done, your tool should return 0 if the request was handled. When the display changes, your tool should send a terminal message of type `telTermDisplayMsg`. If your tool does not support the value in `p1`, it should return `telIndexNotSupp`. If your tool does not support the value in `p3`, it should return `telDisplayModeNotSupp`.

telSetHookswMsg

The Telephone Manager will send `telSetHookswMsg` when an application wants to set the physical hook state of a device attached to the user's terminal.

`hTEL` is a handle to a telephone record for the user's terminal.
`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.
`msg` identifies the message.

`p1` is the device type.

`p2` specifies the desired hook state as `telDeviceOnHook` (on-hook) or `telDeviceOffHook` (off-hook).

`hDN`, `hCA`, and `p3` are unused.

When done, your tool should return 0 if the request was handled. When the hook state changes, your tool should send a terminal message of type `telTermHookswMsg`. If your tool does not support the value in `p1`, it should return `telHTypeNotSupp`.

Your tool can support `telSetHookswMsg` only if it can change the physical status of the specified device (for instance, if it can turn on a speakerphone).

telSetupCallMsg

The Telephone Manager will send `telSetupCallMsg` when an application requests that your tool set up a call appearance for later use.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`. In this record, the destination directory number, name, and subaddress have been filled in by the Telephone Manager. Your tool should fill in the following fields: `caState`, `intExt`, `callType`, `dialType`, `bearerType`, `rate`, `confLimit`, `featureFlags`, `otherFeatures`, and `telCAPrivate`.

`msg` identifies the message.

`p1` is a pointer to any user-to-user information. If your tool does not support user-to-user information, it should ignore this field.

`p2` is the bearer type.

`p3` is the rate. If your tool is using version 1.0 of the Telephone Manager, it must set the bearer type and the rate to zero.

When done, your tool should return 0 if the request was handled. If any error occurs, your tool should place -1 in the `caRef` field of call-appearance record `hCA`, to make the Telephone Manager dispose of `hCA`.

telSetVolumeMsg

The Telephone Manager will send `telSetVolumeMsg` when an application wants to set the volume of a device attached to the user's terminal. Your tool can support `telSetVolumeMsg` only if it can change the physical volume setting of the specified device.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is the volume type (handset volume, ringer volume, and so on).

`p2` points to a 2-byte field storing the desired volume level. If this level is outside the range your tool allows, your tool should store your tool's maximum volume level in the field referenced by `p2`.

`p3` is the desired state.

`hDN` and `hCA` are unused.

When done, your tool should return 0 if the request was handled. If your tool does not support the value in `p1`, it should return `telVTypeNotSupp`.

telSuspendMsg

The Telephone Manager will pass `telSuspendMsg` or `telResumeMsg` when an application requires your tool to perform an action, such as installing or removing a menu from the menu bar in response to a suspend or resume event.

telTermEventsSuppMsg

The Telephone Manager will send `telTermEventsSuppMsg` when an application inquires what type of terminal messages your tool supports.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` points to a field containing a 4-byte mask, to be filled in by your tool. This mask indicates which types of terminal messages your tool supports.

`hDN`, `hCA`, `p2`, and `p3` are unused.

The sample code provides a basic template into which you can code your tool's response to `telTermEventsSuppMsg`. When done, your tool should return 0 if the request was handled.

```
FUNCTION myTELTermEventsSupp (hTEL : TELHandle; VAR eventMask : LONGINT) : INTEGER;
VAR
    err      :      OSErr;
BEGIN
    myTELTermEventsSupp:= noErr;
    { somewhere earlier you did this...
      myTermPrivates^.myTermMessagesSupported := telTermHookMsg + telTermVolMsg +
        telTermOpenMsg + etc. }
END;
```

telTermMsgHandMsg

The Telephone Manager will send `telTermMsgHandMsg` when an application requests that your tool start sending messages about activity related to this terminal.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies the message.

`p1` is the mask of events for which the tool should send messages. This is a master mask equal to the result of performing a logical OR operation on all event masks for registered terminal message handlers.

`p2` is the address of the Telephone Manager's master terminal message handler.

`p3` specifies the globals required by the Telephone Manager's message handler.

`hDN` and `hCA` are unused.

The sample code provides a basic template into which you can code your tool's response to `telTermMsgHandMsg`. When done, your tool should return `noErr` if the request was handled.

```
FUNCTION myTELTermMsgHand(hTEL : TELHandle; eventMask : LONGINT;
                           msgHandler: ProcPtr; globals: LONGINT) : INTEGER;
VAR
    err      :      OSErr;
```

```

BEGIN
    myTELTermMsgHand:= noErr;
    myTermPrivates^.termEventMask := eventMask;
    myTermPrivates^.termMsgHandler := msgHandler;
    myTermPrivates^.termGlobals := globals;
END;

```

telToolFunctionsMsg

The Telephone Manager will send `telToolFunctionsMsg` when an application inquires whether your tool supports a specified Telephone Manager message.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`msg` identifies this message (`telToolFunctionsMsg`).

`p1` specifies the Telephone Manager message about which the application is inquiring.

`p2` points to a Boolean field that specifies whether your tool supports the Telephone Manager message specified in `p1`. This Boolean field should be set to `TRUE` if your tool supports the message, or `FALSE` if it does not.

`hDN`, `hCA`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telTransfBlindMsg

Your tool will receive `TELTransferBlind` when an application requests that a call appearance be transferred immediately (without consultation) to a specified directory number.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1` points to a string of type `Str255` storing the directory number to which the call will be transferred.

`p2` points to a string of type `Str255` storing the subaddress, if any, of directory number `p1`. Unless your tool supports subaddressing, it should ignore `p2`.

`p3` points to a string of type `Str255` storing the name of the party associated with directory number `p1`.

Your tool should save the value of `p1`, `p2`, and `p3`, and send them in the message `telCATransferMsg`.

When done, your tool should return 0 if the request was handled.

telTransfEstMsg

Your tool will receive `telTransfEstMsg` when an application requests that one call appearance (`hCA`) be transferred to a second call appearance (`p1`). The user will have consulted with the party at `p1`. At this time, your tool should transfer `hCA` to `p1`.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`. This is the call appearance for the person being transferred.

`msg` identifies the message.

`p1` is a handle to a second call-appearance record. (The application should have already prepared call appearance `hCA` by calling the Telephone Manager routine `TELTransferPrep`.) This is the call appearance for the person to whom call appearance `hCA` is being transferred.

`p2` and `p3` are unused.

When done, your tool should return 0 if the request was handled.

telTransfPrepMsg

Your tool will receive `telTransfPrepMsg` when an application requests that an active call appearance be prepared for transfer to a second call appearance.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record for an active call on directory number `hDN`. This the call appearance to be transferred.

`msg` identifies the message.

`p1` is a handle to a second call-appearance record. (This call appearance was set up earlier, when the application called the Telephone Manager routine `TELSetupCall`.)

`p2` and `p3` are unused.

If the second call appearance is idle, your tool should proceed as with `TELConnectMsg`, to make that call appearance active. The user can then consult with the party associated with `p1`.

When done, your tool should return 0 if the request was handled.

telVoiceMailAccessMsg

Your tool will receive `telVoiceMailAccessMsg` when an application requests that your tool access the voice-mail system of the telephone network switch.

`hTEL` is a handle to a telephone record for the user's terminal.

`pTELTerm` points to a record of type `TELTermRecord`, part of the telephone record `hTEL`.

`hDN` is a handle to a directory-number record associated with terminal `hTEL`.

`hCA` is a handle to a call-appearance record on directory number `hDN`.

`msg` identifies the message.

`p1`, `p2`, and `p3` are unused.

When done, your tool should return 0 if the request was handled.

Messages that the main code resource sends

Telephone tools, like other tools of the Communications Toolbox, accept messages from their corresponding manager, the Telephone Manager. In addition, telephone tools send messages to the Telephone Manager. Specifically, the main code resource of your tool should send messages to the master message handlers of the Telephone Manager. This section describes each of the messages that your tool can send.

△ **Important** Your tool sends messages by calling the master message handlers of the Telephone Manager. They, in turn, relay your information to application message handlers. (For descriptions of the messages that applications receive from the Telephone Manager, refer to Appendix B.) △

The Telephone Manager maintains three master message handlers, one each for the main message types your tool can send: terminal messages, directory-number messages, and call-appearance messages. Before sending messages of a particular type, your tool must receive from the Telephone Manager a message that contains a procedure pointer to the master message handler for that type. For terminal messages, your tool must receive `telTermMsgHandMsg`; for directory-number messages, `telDNMsgHandMsg`; and for call-appearance messages, `telCAMsgHandMsg`. Each time the Telephone Manager sends one of these messages, your tool should save the procedure pointer contained in the message. In addition, your tool should save the event mask and the pointer to the Telephone Manager globals, also contained in the message. Your tool should send only messages whose types match those specified in the event mask.

To send a message of a particular type, your tool should call the corresponding master message handler. For example, to send a directory-number message, your tool should call the directory-number master message handler. When calling any master message handler, your tool must pass a pointer to a parameter block, which differs depending on the type of message being sent. This parameter block is described later in this chapter.

The messages that can be sent by the main code resource, and their associated values, are as follows:

```
CONST
    telCAActiveMsg          = $00000020
    telCAAlertingMsg        = $00000001

    telCACallbackMsg        = $00000800
    telCAConferenceDropMsg  = $00010000
    telCAConferenceMsg      = $00000040
    telCAConferenceSplitMsg = $00008000

    telCADeflectMsg         = $00002000
    telCADigitsMsg          = $00000200
    telCADisconnectMsg      = $00000010

    telCAFaxToneMsg         = $00800000
    telCAForwardMsg         = $00004000
```

telCAHoldMsg	=	\$00000100
telCAIdleMsg	=	\$01000000
telCAIntercomMsg	=	\$00200000
telCAInUseMsg	=	\$00040000
telCAModemToneMsg	=	\$00400000
telCAOfferMsg	=	\$00000002
telCAOtherMsg	=	\$80000000
telCAOutgoingMsg	=	\$00000008
telCAPagingMsg	=	\$00100000
telCACallParkMsg	=	\$00000400
telCACallPickupMsg	=	\$00080000
telCAProgressMsg	=	\$00000004
telCAQueuedMsg	=	\$00020000
telCAREjectMsg	=	\$00001000
telCASuccessiveAlertMsg	=	\$02000000
telCATransferMsg	=	\$00000080
telCAUserUserInfoMsg	=	\$04000000
telDNDNDMsg	=	\$00000002
telDNForwardMsg	=	\$00000001
telDNOtherMsg	=	\$00008000
telDNSelectedMsg	=	\$00000008
telDNVoiceMailMsg	=	\$00000004
telTermCloseMsg	=	\$00000040
telTermDisplayMsg	=	\$00000008
telTermEnableMsg	=	\$00000010
telTermErrorMsg	=	\$00000100
telTermHookMsg	=	\$00000001
telTermKeyMsg	=	\$00000002
telTermOpenMsg	=	\$00000020
telTermOtherMsg	=	\$00000200
telTermResetMsg	=	\$00000080
telTermVolMsg	=	\$00000004

General call-appearance messages

The call-appearance messages your tool can send are of several types—for instance, messages about conference calls and messages about calls being disconnected. Each type requires that your tool pass a different parameter block.

This section describes the general call-appearance messages: those not requiring an extended parameter block. The other types of call-appearance messages are described later in this chapter.

Here is a list of the general call-appearance messages:

<code>telCAActiveMsg</code>	<code>telCACallbackMsg</code>
<code>telCADeflectMsg</code>	<code>telCADigitsMsg</code>
<code>telCADisconnectMsg</code>	<code>telCAFaxToneMsg</code>
<code>telCAForwardMsg</code>	<code>telCAHoldMsg</code>
<code>telCAIdleMsg</code>	<code>telCAIntercomMsg</code>
<code>telCAInUseMsg</code>	<code>telCAModemToneMsg</code>
<code>telCAPagingMsg</code>	<code>telCACallParkMsg</code>
<code>telCACallPickupMsg</code>	<code>telCAProgressMsg</code>
<code>telCAQueuedMsg</code>	<code>telCAREjectMsg</code>
<code>telCASuccessiveAlertMsg</code>	

To send a general call-appearance message, your tool must call the call-appearance master message handler passed in the Telephone Manager message `telCAMsgHandMsg`. When calling this message handler, your tool must pass a pointer to the following parameter block:

```
TELCAGenericMsgPB    = RECORD
    toolID              : INTEGER;
    tRef                : INTEGER;
    dnRef               : INTEGER;
    caRef               : INTEGER;
    msg                 : LONGINT;
    mtype               : INTEGER;
    value               : INTEGER;
    rmtDN               : StringPtr;
    rmtName              : StringPtr;
    rmtSubaddress       : StringPtr;
    dialType            : INTEGER;
END;
```

`toolID` is a unique identifier that the Telephone Manager assigns to your tool when sending the message `telNewMsg`. Your tool should save `toolID` in private storage.

`tRef` is the reference number that your tool assigns to this particular terminal. Do not change the value of `tRef` while the terminal is open.

`dnRef` is the reference number that your tool assigns to this particular directory number. Do not change the value of `dnRef` while the directory number is valid.

`caRef` is the reference number that your tool assigns to this particular call appearance. Do not change the value of `caRef` while the call appearance is valid.

`msg` identifies the message that your tool is sending.

`mtype` and `value` vary in meaning, depending on the message being sent, and are described in the description of each message.

`rmtDN` points to a string of type `str255` that stores a remote directory number—for instance, the directory number being called.

`rmtName` points to a string of type `str255` storing a name associated with `rmtDN`.

`rmtSubaddress` points to a string of type `str255` storing the subaddress, if any, associated with `rmtDN`.

`dialType` specifies the type of directory number and name contained in `rmtDN` and `rmtName` (10-digit North American, non-dialable, and so on).

The rest of this section describes each of the general call-appearance messages that your tool can send.

telCAActiveMsg

Your tool should send `telCAActiveMsg` each time an outgoing or incoming call appearance becomes active. `mtype`, `value`, `rmtDN`, `rmtName`, `rmtSubaddress`, and `dialType` are unused and set to zero.

telCACallbackMsg

Your tool should send `telCACallbackMsg` each time Call Back activity occurs on a particular call appearance.

`mtype` returns a callback reference value, if there is one; otherwise, `mtype` returns 0. `value` specifies whether the callback was set, established, failed, and so on. `rmtDN` and `rmtName` specify the remote directory number associated with the callback. Set `rmtDN`, `rmtName`, `rmtSubaddress`, and `dialType` to zero if directory-number information is unavailable or not supported by your tool.

telCADeflectMsg

Your tool should send `telCADeflectMsg` each time Call Deflect activity occurs on a particular call appearance. Your tool should also send `telCADeflectMsg` each time a call is automatically deflected as a result of forwarding, if your system provides this information.

`mtype` is unused and set to zero. `value` specifies whether the attempt to deflect the call succeeded. `rmtDN`, `rmtName`, `rmtSubaddress`, and `dialType` specify the directory number, party name, subaddress, and dial type to the call that was deflected.

telCADigitsMsg

Your tool should send `telCADigitsMsg` to indicate that one or more keys were pressed at the remote directory number associated with this call appearance.

`mtype` specifies whether the key pressed corresponds to an audible dual-tone multiple frequency (DTMF) tone. `value` specifies the ASCII value of the keys pressed. `rmtDN`, `rmtName`, `rmtSubaddress`, and `dialType` are unused and set to zero.

telCADisconnectMsg

Your tool should send **telCADisconnectMsg** each time an active, held, or conferenced call appearance is dropped (hung up) by the local party or the remote party.

rmtDN, **rmtName**, **rmtSubaddress**, and **dialType** are unused and set to zero.

telCAFaxToneMsg

Your tool should send **telCAFaxToneMsg** to indicate that a fax tone has been detected on a particular call appearance.

mtype is unused and set to zero. **value** specifies whether the tone is still present. **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** are unused and set to zero.

telCAForwardMsg

Your tool should send **telCAForwardMsg** to indicate that an outgoing call has been forwarded to a new destination.

mtype is unused and set to zero. **value** specifies the type of forwarding that occurred. **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** specify the directory number, party name, subaddress, and dial type to which the call was forwarded.

telCAHoldMsg

Your tool should send **telCAHoldMsg** each time Hold activity occurs for a call appearance.

mtype is unused and should be set to zero. **value** specifies whether the call was held or retrieved, or an attempt to hold a call failed. **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** are unused and set to zero.

telCAIdleMsg

Your tool should send **telCAIdleMsg** each time a call appearance becomes idle.

mtype, **value**, **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** are unused and set to zero.

telCAIntercomMsg

Your tool should send **telCAIntercomMsg** each time Intercom activity occurs on a particular call appearance.

mtype is unused and set to zero. **value** specifies whether the attempted Intercom operation succeeded. **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** are unused and set to zero.

telCAInUseMsg

Your tool should send **telCAInUseMsg** to indicate that a particular call appearance is in use, but at another terminal. This message is applicable only to multiple-access directory numbers (MADNs).

mtype, **value**, **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** are unused and set to zero.

telCAModemToneMsg

Your tool should send **telCAModemToneMsg** to indicate that a modem tone has been detected on a particular call appearance.

mtype is unused and set to zero. **value** specifies whether the tone is still present. **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** are unused and set to zero.

telCAPagingMsg

Your tool should send **telCAPagingMsg** each time that there is paging activity for a call appearance.

mtype is unused and set to zero. **value** specifies whether the attempt to page succeeded. **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** are unused and set to zero.

telCACallParkMsg

Your tool should send **telCACallParkMsg** each time Call Park activity occurs on a particular call appearance.

mtype is unused and should be set to zero. **value** specifies whether the call was parked, retrieved, or recalled, or failed to be parked or retrieved. **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** specify the directory number, party name, subaddress, and dial type against which the call was parked, if this information is available. If Call Park is not available or is not supported by your tool, **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** should be set to zero.

telCACallPickupMsg

Your tool should send **telCACallPickupMsg** each time Call Pickup activity occurs on a particular call appearance.

mtype is unused and set to zero. **value** specifies whether the pickup was successful or failed. Messages are also sent to originators if the call was picked up at a directory number other than the one dialed. **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** specify the directory number whose call was picked up successfully or unsuccessfully. In the case of a message sent to the originator, **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** specify the directory number that picked up the call.

telCAProgressMsg

Your tool should send **telCAProgressMsg** to indicate each change in the state of an outgoing call appearance.

mtype is unused and should be set to zero. **value** specifies the progress indicator. If **value** is **telCAPUpdate** or **telCAPRouted**, **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** specify the updated directory number, party name, subaddress, and dial type. If **value** is neither **telCAPUpdate** nor **telCAPRouted**, **rmtDN**, **rmtName**, **rmtSubaddress**, and **dialType** are set to zero.

telCAQueuedMsg

Your tool should send `telCAQueuedMsg` each time an incoming call is queued for this terminal.

`mtype`, `value`, `rmtDN`, `rmtName`, `rmtSubaddress`, and `dialType` are unused and set to zero.

telCARejectMsg

Your tool should send `telCARejectMsg` each time Call Reject activity occurs on a particular call appearance. This message should also be sent if an outgoing call was rejected by the destination.

`mtype` is unused and set to zero. `value` specifies whether the reject succeeded or failed. `rmtDN`, `rmtName`, `rmtSubaddress`, and `dialType` are unused and set to zero.

telCASuccessiveAlertMsg

Your tool should send `telCASuccessiveAlertMsg` each time the attached telephone set “rings.” The application can thereby count rings, answering on the *n*th ring.

`mtype`, `value`, `rmtDN`, `rmtName`, `rmtSubaddress`, and `dialType` are unused and set to zero.

telCAUserUserInfoMsg

Your tool should send `telCAUserUserInfoMsg` each time user-to-user information arrives for a call appearance.

`rmtDN` stores the user-to-user information. `mtype`, `value`, `rmtName`, `rmtSubaddress`, and `dialType` are unused and set to zero.

Incoming/outgoing call-appearance messages

This section describes incoming/outgoing call-appearance messages: messages specific to a call appearance that is currently coming in or going out. The incoming/outgoing call-appearance messages are as follows:

`telCAAlertingMsg`

`telCAOfferMsg`

`telCAOutgoingMsg`

Each time your tool sends one of these messages, the Telephone Manager creates a handle to the specified call-appearance record and passes this handle to the appropriate applications. If an application has already created a call-appearance handle for a particular outgoing call (using the routine `TELSetupCall`), the Telephone Manager creates call-appearance handles for any other applications monitoring that call.

To send an incoming/outgoing call-appearance message, your tool must call the call-appearance master message handler passed in the Telephone Manager message `telCMsgHandlerMsg`. When calling this message handler, your tool must pass a pointer to the following parameter block:

```

TELCAInOutMsgPB      = RECORD
    toolID             : INTEGER;
    tRef               : INTEGER;
    dnRef              : INTEGER;
    caRef              : INTEGER;
    msg                : LONGINT;
    mtype              : INTEGER;
    value              : INTEGER;
    rmtDN              : StringPtr;
    rmtName            : StringPtr;
    rmtSubaddress      : StringPtr;
    callType           : INTEGER;
    dialType           : INTEGER;
    bearerType         : INTEGER;
    rate               : INTEGER;
    routeDN            : StringPtr;
    routeName          : StringPtr;
    routeSubaddress    : StringPtr;
    featureFlags       : LONGINT;
    otherFeatures      : LONGINT;
    telCAPrivate       : LONGINT;
END;

```

`toolID` is a unique identifier that the Telephone Manager assigns to your tool when sending the message `telNewMsg`. Your tool should save `toolID` in private storage.

`tRef` is the reference number that your tool assigns to this particular terminal. Do not change the value of `tRef` while the terminal is open.

`dnRef` is the reference number that your tool assigns to this particular directory number. Do not change the value of `dnRef` while the directory number is valid.

`caRef` is the reference number that your tool assigns to this particular call appearance. Do not change the value of `caRef` while the call appearance is valid.

`msg` identifies the message that your tool is sending.

`mtype` is the message type, if any.

`value` is the message value, if any.

`rmtDN` points to a string of type `str255` that stores a remote directory number—for instance, the directory number being called.

`rmtName` points to a string of type `str255` storing a name associated with `rmtDN`.

`rmtSubaddress` points to a string of type `str255` that stores the subaddress, if any, associated with `rmtDN`.

`callType` specifies whether the call appearance is direct-inbound, transferred-inbound, recalled, and so on.

`dialType` specifies the type of directory number and name contained in `rmtDN` and `rmtName` (10-digit North American, non-dialable, and so on).

`bearerType` and `rate` are unused but are reserved by Apple for future use. Set these fields to zero.

`routedDN` points to a string of type `Str255` that stores a remote directory number.

`routeName` points to a string of type `Str255` that stores a name associated with `routedDN`.

`routeSubaddress` points to a string of type `Str255` that stores the subaddress, if any, associated with `routedDN`.

`featureFlags` indicates the initial state of the features in the field `featureFlags` of the call-appearance record for `caRef`.

`otherFeatures` indicates the initial state of the features in the field `otherFeatures` of the call-appearance record for `caRef`.

`telCAPrivate` is private data that your tool wants stored in the call-appearance record for `caRef`.

The rest of this section describes each of the incoming/outgoing call-appearance messages that your tool can send.

telCAAlertingMsg

Your tool should send `telCAAlertingMsg` to indicate that an incoming call is in the state `telCAAlertingState`.

`mtype` is unused and should be set to zero.

`value` is the alerting pattern. It has one of the following values:

<code>telPattern0</code>	<code>= 0;</code>	<code>{ Normal Alerting Pattern }</code>
<code>telPattern1</code>	<code>= 1;</code>	<code>{ Alerting Pattern - type 1 }</code>
<code>telPattern2</code>	<code>= 2;</code>	<code>{ Alerting Pattern - type 2 }</code>
<code>telPattern3</code>	<code>= 3;</code>	<code>{ Alerting Pattern - type 3 }</code>
<code>telPattern4</code>	<code>= 4;</code>	<code>{ Alerting Pattern - type 4 }</code>
<code>telPattern5</code>	<code>= 5;</code>	<code>{ Alerting Pattern - type 5 }</code>
<code>telPattern6</code>	<code>= 6;</code>	<code>{ Alerting Pattern - type 6 }</code>
<code>telPattern7</code>	<code>= 7;</code>	<code>{ Alerting Pattern - type 7 }</code>
<code>telPatternOff</code>	<code>= 8;</code>	<code>{ Alerting Pattern - turned off }</code>
<code>telPatternUndefined</code>	<code>= 15;</code>	<code>{ Alerting Pattern undefined }</code>

telCAOfferMsg

Your tool should send `telCAOfferMsg` to indicate that an incoming call is in the state `telCAOfferState`.

`mtype` and `value` are unused and should be set to zero.

telCAOutgoingMsg

Your tool should send `telCAOutgoingMsg` to indicate that an outgoing call has been initiated from the user's terminal.

`mtype` is unused and should be set to zero. `value` is one of the following values:

```
telPhysical      =      0;
                  { User lifted handset and initiated call }
telProgrammatic  =      1;
                  { Outgoing call initiated programmatically }
```

Call-appearance message for transferring calls

This section describes `telCATransferMsg`, a message that your tool should send each time activity occurs on the Call Transfer feature of a call appearance.

To send `telCATransferMsg`, your tool must call the call-appearance master message handler passed in the Telephone Manager message `telCAMsgHandMsg`. When calling this message handler, your tool must pass a pointer to the following parameter block:

```
TELCATransfMsgPB      = RECORD
    toolID              : INTEGER;
    tRef                : INTEGER;
    dnRef               : INTEGER;
    caRef               : INTEGER;
    msg                 : LONGINT;
    mtype               : INTEGER;
    value               : INTEGER;
    rmtDN               : StringPtr;
    rmtName             : StringPtr;
    rmtSubaddress       : StringPtr;
    dialType            : INTEGER;
END;
```

`toolID` is a unique identifier that the Telephone Manager assigns to your tool when sending the message `telNewMsg`. Your tool should save `toolID` in private storage.

`tRef` is the reference number that your tool assigns to this particular terminal. Do not change the value of `tRef` while the terminal is open.

`dnRef` is the reference number that your tool assigns to this particular directory number. Do not change the value of `dnRef` while the directory number is valid.

`caRef` is the reference number that your tool assigns to this particular call appearance. Do not change the value of `caRef` while the call appearance is valid.

`msg` identifies the message that your tool is sending.

`mtype` is the message type, if any.

`value` is the message value, if any.

`rmtDN` points to a string of type `str255` that stores the remote directory number to which the call will be transferred.

`rmtName` points to a string of type `str255` storing a name associated with `rmtDN`.

`rmtSubaddress` points to a string of type `str255` storing the subaddress, if any, associated with `rmtDN`.

Call-appearance messages for conference calls

This section describes call-appearance messages that apply to calls being united in a conference:

```
telCAConferenceMsg
telCAConferenceSplitMsg
telCAConferenceDropMsg
```

To send any of these messages, your tool must call the call-appearance master message handler passed in the Telephone Manager message `telCAMsgHandlerMsg`. When calling this message handler, your tool must pass a pointer to the following parameter block:

```
TELCAGenericPB      = RECORD
    toolID            : INTEGER;
    tRef              : INTEGER;
    dnRef             : INTEGER;
    caRef             : INTEGER;
    msg               : LONGINT;
    mtype             : INTEGER;
    value             : INTEGER;
END;
```

`toolID` is a unique identifier that the Telephone Manager assigns to your tool when sending the message `telNewMsg`. Your tool should save `toolID` in private storage.

`tRef` is the reference number that your tool assigns to this particular terminal. Do not change the value of `tRef` while the terminal is open.

`dnRef` is the reference number that your tool assigns to this particular directory number. Do not change the value of `dnRef` while the directory number is valid.

`caRef` is the reference number that your tool assigns to this particular call appearance. Do not change the value of `caRef` while the call appearance is valid.

`msg` identifies the message that your tool is sending.

`mtype` is the message type, if any.

`value` is the message value, if any.

The rest of this section describes each of the call-appearance messages for conference calls.

telCAConferenceDropMsg

Your tool should send `telCAConferenceDropMsg` to indicate that a specified call appearance has been dropped, not merely split from a conference.

`mtype` is unused and should be set to zero.

`value` should be set to one of the following values:

```
telConferenceDropFailed = 0; { CA could not be dropped }
telConferenceDropped    = 1; { CA dropped successfully }
ConfDropByInitiator    = 2; { CA dropped because initiator was dropped }
```

telCAConferenceMsg

Your tool should send `telCAConferenceMsg` to indicate activity—other than the splitting or dropping of call—on the Conference feature of a particular call appearance.

`mtype` is unused and should be set to zero.

`value` should be set to one of the following values:

```
telConferencePrepFailed = 0; { conference could not be prepared }
telConferencePending    = 1; { conference prepared successfully }
telConferenceEstFailed  = 2; { conference could not be established }
telConferenceEst        = 3; { conference established }
```

telCAConferenceSplitMsg

Your tool should send `telCAConferenceSplitMsg` to indicate that a specified call appearance has been split from a conference (but not dropped).

`mtype` is unused and should be set to zero.

`value` should be set to one of the following values:

```
telConferenceDropFailed = 0; { CA could not be dropped }
telConferenceDropped    = 1; { CA dropped successfully }
```

Directory-number messages

This section describes directory-number messages: messages your tool can send regarding a particular directory number. To send a directory-number message, your tool must call the call-appearance master message handler passed in the Telephone Manager message `telDNMsgHandlerMsg`. When calling this message handler, your tool must pass a pointer to the following parameter block:

```

TELDNMsgPB          = RECORD
    toolID           : INTEGER;
    tRef              : INTEGER;
    dnRef             : INTEGER;
    msg               : LONGINT;
    mtype             : INTEGER;
    value             : INTEGER;
    rmtDN             : StringPtr;
    rmtName           : StringPtr;
    rmtSubaddress     : StringPtr;
END;

```

`toolID` is a unique identifier that the Telephone Manager assigns to your tool when sending the message `telNewMsg`. Your tool should save `toolID` in private storage.

`tRef` is the reference number that your tool assigns to this particular terminal. Do not change the value of `tRef` while the terminal is open.

`dnRef` is the reference number that your tool assigns to this particular directory number. Do not change the value of `dnRef` while the directory number is valid.

`msg` identifies the directory-number message your tool is sending.

`mtype` and `value` vary in meaning according to the message being sent, and are described in the description of each message.

`rmtDN` points to a string of type `str255` that stores a remote directory number—for instance, the directory number being called.

`rmtName` points to a string of type `str255` storing a name associated with `rmtDN`.

`rmtSubaddress` points to a string of type `str255` storing the subaddress, if any, associated with `rmtDN`.

The rest of this section describes each of the directory-number messages that your tool can send.

telDNDNDMsg

Your tool should send `telDNDNDMsg` to indicate a change in the status of the Do Not Disturb feature on particular directory number.

`value` specifies whether the Do Not Disturb feature has been activated or cleared, or whether an attempt to do so has failed. `mtype` specifies the type of Do Not Disturb feature affected. `rmtDN`, `rmtName`, and `rmtSubaddress` are unused and should be set to zero.

telDNForwardMsg

Your tool should send `telDNForwardMsg` to indicate a change in the status of the Call Forward feature on a particular directory number.

`value` specifies whether call forwarding has been activated or cleared, or whether an attempt to do so has failed. `mtype` specifies the type of call forwarding affected. `rmtDN` specifies the remote directory number to which calls are forwarded. `rmtName` and `rmtSubaddress` are a name and subaddress associated with `rmtDN`.

telDNOtherMsg

Your tool should send `telDNOtherMsg` to convey directory-number messages other than those listed in this section.

`value` and `mtype` can be set to any values; those values should be described in your tool documentation.

telDNSelectedMsg

Your tool should send `telDNSelectedMsg` after selecting or deselecting a directory number.

`value` specifies whether the directory number was selected or deselected. `mtype`, `rmtDN`, `rmtName`, and `rmtSubaddress` are unused and should be set to zero.

telDNVoiceMailMsg

Your tool should send `telDNVoiceMailMsg` to indicate a change in the status of the voice-mail feature associated with a particular directory number.

`value` specifies either that a new voice-mail message is waiting or that all messages have been cleared. `mtype` is unused and should be set to zero. `rmtName`, `rmtDN`, and `rmtSubaddress` specify the name, number, and subaddress of the party who left the voice-mail message. If any of these items are not available, the corresponding fields should be set to zero.

Terminal messages

This section describes terminal messages: messages that your tool can send regarding a particular terminal as a whole. To send a terminal message your tool must call the call-appearance master message handler passed in the Telephone Manager message `telTermMsgHandMsg`. When calling this message handler, your tool must pass a pointer to the following parameter block:

```
TELTermMsgPB      = RECORD
    toolID         : INTEGER;
    tRef           : INTEGER;
    msg            : LONGINT;
    mtype          : INTEGER;
    value          : INTEGER;
END;
```

`toolID` is a unique identifier that the Telephone Manager assigns to your tool when sending the message `telNewMsg`. Your tool should save `toolID` in private storage.

`tRef` is the terminal reference number that your tool assigns to this particular terminal. Do not change the value of `tRef` while the terminal is open.

`msg` identifies the terminal message your tool is sending.

`mtype` and `value` vary in meaning according to the message being sent, and are described in the description of each message.

The rest of this section describes each of the terminal messages that your tool can send.

telTermCloseMsg

Your tool should send `telTermCloseMsg` after closing the terminal driver. `value` and `mtype` are unused and should be set to zero.

telTermDisplayMsg

Your tool should send `telTermDisplayMsg` after changing the display. `value` specifies the type of display information that changed. `mtype` specifies the current display mode.

telTermEnableMsg

Your tool should send `telTermEnableMsg` once it can communicate with the terminal. `mtype` is unused and should be set to zero. `value` indicates whether communication with the terminal is enabled.

telTermErrorMsg

Your tool should send `telTermErrorMsg` after a terminal error has occurred. `value` and `mtype` are unused and should be set to zero.

This message tells the application only that an error has occurred; it does not specify the exact error. Send `telOtherMsg` to specify the exact error.

telTermHookMsg

Your tool should send `telTermHookMsg` each time the physical state of an attached device changes. For instance, if someone lifts the hookswitch of a phone that is attached to the terminal, your tool should send `telTermHookMsg`. `value` specifies the state of the hookswitch, and `mtype` specifies the device type.

telTermKeyMsg

Your tool should send `telTermKeyMsg` each time a key is physically pressed on an attached telephone set.

`value` specifies either the ASCII value of the key or a key feature code. `mtype` specifies whether the key pressed was a keypad key (a digit between 0 and 9, the number sign, or the asterisk) or a feature key (such as Hold or Conference).

telTermOpenMsg

Your tool should send `telTermOpenMsg` after opening the terminal driver. `value` and `mtype` are unused and should be set to zero.

telTermOtherMsg

Your tool should send `telTermOtherMsg` to convey terminal-related messages other than those listed in this section. For example, to alert an application that a terminal driver has encountered an error, you could define and send a `telTermOtherMsg` message.

`value` and `mtype` can be set to any values; those values should be described in your tool documentation.

telTermResetMsg

Your tool should send `telTermResetMsg` after resetting the terminal driver.

`value` and `mtype` are unused and should be set to zero.

telTermVolMsg

Your tool should send `telTermVolMsg` after changing the volume of a device.

`value` specifies the new volume level. `mtype` specifies the device on which the volume has been changed.

Appendix A **Result Codes for Routines**

THIS APPENDIX lists and describes the result codes returned by Telephone Manager routines.

Each result code is of data type `TELErr`, which is the same as the data type `OSerr`. For information about `OSerr`, refer to Volume II of *Inside Macintosh* and the include file `types.p`.

Result Code	Meaning
<code>noErr</code>	The routine finished without error.
<code>telAlreadyOpen</code>	The terminal is already open.
<code>telAPattNotSupp</code>	The tool does not support this alerting pattern.
<code>telBadAPattErr</code>	The alerting pattern is invalid.
<code>telBadBearerType</code>	The bearer type is invalid.
<code>telBadCAErr</code>	The call-appearance handle is invalid or not found.
<code>telBadDNErr</code>	The directory-number handle is invalid or not found.
<code>telBadDNDType</code>	The Do Not Disturb type is invalid.
<code>telBadDNType</code>	The directory-number type is invalid.
<code>telBadFeatureID</code>	The feature ID is invalid.
<code>telBadFunction</code>	The message code is invalid.
<code>telBadFwdType</code>	The forward type is invalid.
<code>telBadHandErr</code>	The handle is invalid.
<code>telBadHTypeErr</code>	The hook type is invalid.
<code>telBadIndex</code>	The index is invalid.
<code>telBadInterComID</code>	The intercom ID is invalid.
<code>telBadIntExt</code>	The internal/external specifier is invalid.
<code>telBadLevelErr</code>	The volume-level setting is invalid.
<code>telBadPageID</code>	The page ID is invalid.
<code>telBadParkID</code>	The park ID is invalid.
<code>telBadPickupGroupID</code>	The pickup-group ID is invalid.
<code>telBadProcErr</code>	<code>msgProc</code> is invalid.
<code>telBadRate</code>	The rate is invalid.
<code>telBadSelect</code>	Cannot select or deselect the directory number.
<code>telBadStateErr</code>	The device state is invalid.
<code>telBadTermErr</code>	The telephone record handle is invalid or not found.
<code>telBadVTypeErr</code>	The volume type is invalid.

<code>telCANotAcceptable</code>	This call appearance is not "acceptable."
<code>telCANotDeflectable</code>	This call appearance is not "deflectable."
<code>telCANotRejectable</code>	This call appearance is not "rejectable."
<code>telCAUnavailable</code>	A new call appearance is not "available."
<code>telCBErr</code>	The specified Call Back feature has not been previously set.
<code>telChooseAborted</code>	The application user tried to change the tool settings while the terminal was running—that is, after the application has called <code>TELOpenTerm</code> and before the application has called <code>TELCloseTerm</code> .
<code>telChooseCancel</code>	The user clicked the Cancel button of the dialog box.
<code>telChooseDisaster</code>	The routine failed because no tools were found in the Extensions folder. The telephone record has been destroyed. <code>NIL</code> is returned in the telephone record handle.
<code>telChooseFailed</code>	Attempt to choose a tool or to change settings of the current tool failed. The telephone record remains unchanged.
<code>telChooseOKMajor</code>	The application user clicked the OK button of the dialog box after choosing a different tool or changing the settings of the current tool.
<code>telChooseOKMinor</code>	The application user clicked the OK button of the dialog box but did not change the settings of the telephone tool.
<code>telChooseOKTermChanged</code>	The application user clicked on the OK button of the dialog box after choosing a different tool or changing the settings of the current tool. The terminal reference number (<code>tRef</code>) has changed.
<code>telConfErr</code>	The call appearance specified in <code>hTELCA1</code> is not the conference initiator.
<code>telConfLimitErr</code>	The limit specified is too high for this configuration.
<code>telConfLimitExceeded</code>	Attempted to conference more call appearances than the network switch allows.
<code>telConfNoLimit</code>	A limit is required, but none was specified.
<code>telConfRej</code>	The conference request was rejected.
<code>telDNDNotSupp</code>	The Do Not Disturb type is not supported.
<code>telDNTTypeNotSupp</code>	The directory-number type is not supported.

<code>telFeatActive</code>	This feature is already active.
<code>telFeatNotAvail</code>	This feature is subscribed but not available.
<code>telFeatNotSubscr</code>	This feature is not subscribed.
<code>telFeatNotSupp</code>	The tool does not support the specified feature program call.
<code>telFwdTypeSupp</code>	The tool does not support this type of forwarding.
<code>telHTypeNotSupp</code>	The tool does not support this hook type.
<code>telIndexNotSupp</code>	The tool does not support this index.
<code>telIntExtNotSupp</code>	The tool does not support this <code>intExt</code> type.
<code>telInitFailed</code>	Cannot initialize the Telephone Manager.
<code>telModeNotSupp</code>	The tool does not support this display mode.
<code>telNoCallbackRef</code>	You must supply a callback reference value.
<code>telNoCommFolder</code>	Cannot find the Extensions folder.
<code>telNoMemErr</code>	No memory to allocate a handle or internal storage.
<code>telNoOpenErr</code>	Cannot open the terminal.
<code>telNoSuchTool</code>	Cannot find a tool with specified name.
<code>telNoTools</code>	Cannot find any telephone tools in the Extensions folder.
<code>telPBErr</code>	The format of the parameter block is invalid.
<code>telStateNotSupp</code>	Device state is not supported by the tool.
<code>telStillNeeded</code>	Another user still needs the terminal driver.
<code>telTermNotOpen</code>	The terminal has not yet been opened by the <code>TELOpenTerm</code> routine.
<code>telTransfErr</code>	The implicit transfer was not prepared.
<code>telTransfRej</code>	The transfer request was rejected.
<code>telUnknownErr</code>	Cannot set the configuration.
<code>telVTypeNotSupp</code>	Volume type is not supported by this tool.

Appendix B **Message Codes for Applications**

THIS APPENDIX lists and describes the message codes that application message handlers receive from the Telephone Manager. The message codes are listed alphabetically and are presented in three groups: call-appearance message codes, directory-number message codes, and terminal message codes.

Call-appearance message codes for applications

The message codes in this section relate to specific call appearances. For information on the format and parameters of call-appearance messages, refer to the template for call-appearance message handlers (`MyCAMsgHandler`) in the section "Routines Your Application Must Provide" in Chapter 2.

`telCAActiveMsg`

Description `telCAActiveMsg` indicates that the call appearance `hTELCA` has been successfully connected to the destination and that conversation or data is free to flow over the connection. Your application receives this message if you connect with `TELConnect` or if the user has placed a call manually. This message is received at the destination when a successful call is made to `TELAnswer` or when the remote party manually answers the telephone. `mtype`, `value`, and `msgInfo` are unused and set to zero.

`telCAAlertingMsg`

Description `telCAAlertingMsg` specifies that an incoming call `hTELCA` is alerting at this terminal. `mtype` is unused and set to zero.

`value` is the alerting pattern, if any. Some switches assign certain ring patterns to certain calls. For instance, a switch might assign one pattern for outside calls, another for data calls, and so on. `value` is assigned one of the following values:

<code>telPattern0</code>	<code>= 0;</code>	<code>{ Normal Alerting Pattern }</code>
<code>telPattern1</code>	<code>= 1;</code>	<code>{ Alerting Pattern - type 1 }</code>
<code>telPattern2</code>	<code>= 2;</code>	<code>{ Alerting Pattern - type 2 }</code>
<code>telPattern3</code>	<code>= 3;</code>	<code>{ Alerting Pattern - type 3 }</code>
<code>telPattern4</code>	<code>= 4;</code>	<code>{ Alerting Pattern - type 4 }</code>
<code>telPattern5</code>	<code>= 5;</code>	<code>{ Alerting Pattern - type 5 }</code>
<code>telPattern6</code>	<code>= 6;</code>	<code>{ Alerting Pattern - type 6 }</code>
<code>telPattern7</code>	<code>= 7;</code>	<code>{ Alerting Pattern - type 7 }</code>
<code>telPatternOff</code>	<code>= 8;</code>	<code>{ Alerting Pattern - turned off }</code>
<code>telPatternUndefined</code>	<code>= 15;</code>	<code>{ Alerting Pattern undefined }</code>

`msgInfo` is unused and set to zero.

telCACallbackMsg

Description `telCACallbackMsg` gives information about activities related to Call Back features. This message is sent when the user presses the Call Back key or when your application calls `TELCallbackSet` or `TELCallbackClear`.

`hTELCA` is set to zero if value equals `telCallbackNowAvail`.

`mType` returns a callback reference value, if one is used when value equals `telCallbackNowAvail`, `telCallbackDesired`, or `telCallbackDesiredCleared`. Otherwise, `mType` is unused and set to zero.

value is as follows:

<code>telCallbackCleared</code>	= 0; { Callback has been cleared, <code>hTELCA</code> is zero. }
<code>telCallbackEst</code>	= 1; { Callback has been setup/established. }
<code>telCallbackNowAvail</code>	= 2; { Call can be called back with <code>TELCallbackNow</code> . CA is NIL. }
<code>telCallbackDesired</code>	= 4; { A user has called this terminal, received no answer, and requests that this terminal call the user back. <code>hTELCA</code> is zero. }
<code>telCallbackDesiredCleared</code>	= 5; { Callback for no answer no longer desired. <code>hTELCA</code> is zero. }
<code>telCalledback</code>	= 6; { Callback has successfully occurred. }

`msgInfo` points to the following structure:

```
TYPE
CAGenericMsgRec   = RECORD
    rmtDN          : StringPtr;
    rmtName        : StringPtr;
    rmtSubaddress  : StringPtr;
    dialType       : INTEGER;
END
```

If value is `telCallbackCleared`, `telCallbackEst`, or `telCallbackFailed`, `rmtDN` specifies the remote directory number for which callback has been set, and `rmtName` specifies the name of the party associated with that directory number. If value is `telCallbackNowAvail`, `rmtDN` and `rmtName` specify the directory number and party name to be called back. If value is `telCallbackDesiredCleared` or `telCallbackDesired`, `rmtDN` specifies the remote directory number for which callback is desired and `rmtName` specifies the name of the party associated with that directory number.

`rmtSubaddress` indicates the subaddress, if any, associated with `rmtDN`. `dialType` indicates the "dialability" of the number specified in `rmtDN`.

telCAConferenceDropMsg

Description `telCAConferenceDropMsg` specifies that `hTELCA` has been dropped from a conference. This message will be sent when the application calls `TELDrop` and the `hTELCA` is part of a conference. `mtype` is unused and set to zero. `value` is set to one of the following values:

```
telConferenceDropFailed    = 0;    { CA could not be dropped }
telConferenceDropped       = 1;    { CA dropped successfully }
```

`msgInfo` points to the following structure:

```
TYPE
CAConfMsgRec    = RECORD
relatedCA       : TELCAHandle;
END
```

`relatedCA` specifies a handle to the conference initiator (`hTELCA^^.relatedCA`)

telCAConferenceMsg

Description `telCAConferenceMsg` specifies that conference activity is occurring for `hTELCA`. This activity could be occurring because of calls to `TELConferencePrep` or `TELConferenceEstablish`, or because the user is manually initiating a conference. `mtype` is unused and set to zero. `value` is set to one of the following values:

```
telConferencePrepFailed    = 0;    { conference could not be prepared }
telConferencePending       = 1;    { conference prepared successfully }
telConferenceEstFailed     = 2;    { conference could not be established }
telConferenceEst           = 3;    { conference established }
```

`msgInfo` points to the following structure:

```
TYPE
CAConfMsgRec    = RECORD
relatedCA       : TELCAHandle;
END
```

`relatedCA` specifies a handle to the conference initiator for values of `telConferenceEstFailed` and `telConferenceEst`. Otherwise, `relatedCA` is zero.

telCAConferenceSplitMsg

Description `telCAConferenceSplitMsg` specifies that conference-splitting activity is occurring for `hTELCA`. This message will be sent when the application calls `TELConferenceSplit`. `mtype` is unused and set to zero. `value` is set to one of the following values:

```
telConferenceSplitFailed    = 0;   { CA could not be split }
telConferenceSplitEst      = 1;   { CA split successfully }
```

msgInfo points to the following structure:

```
TYPE
CAConfMsgRec  = RECORD
relatedCA     : TELCAHandle;
END
```

relatedCA specifies a handle to the conference initiator.

telCADeflectMsg

Description

telCADeflectMsg gives information about activities related to Call Deflect features. This message is sent when the user presses the Call Deflect button or when your application calls TELDeflectCall. This message can be received on either end of a call appearance. mtype is unused and set to zero. value is as follows:

```
telCallDeflectFailed      = 0;   { attempt to deflect call has failed }
telCallDeflectEst         = 1;   { call successfully deflected }
telCallDeflectRecall      = 2;   { deflected call has been recalled }
telCallDeflected         = 3;   { message to originator that call
                                   was deflected to rmtDN }
telAutoDeflectNoAnswer    = 4;   { a call was automatically deflected
                                   from this terminal as a result
                                   of call forwarding on noanswer }
telAutoDeflectBusy        = 5;   { a call was automatically deflected
                                   from this terminal as a result
                                   of call forwarding on busy }
telAutoDeflectImmediate   = 6;   { a call was automatically deflected
                                   from this terminal as a result
                                   of immediate call forwarding }
```

msgInfo points to the following structure:

```
TYPE
CAGenericMsgRec  = RECORD
rmtDN            : StringPtr;
rmtName          : StringPtr;
rmtSubaddress    : StringPtr;
dialType         : INTEGER;
END
```

rmtDN and rmtName specify the directory number and user name to which the call was deflected. dialType indicates the "dialability" of the number specified in rmtDN.

telCADigitsMsg

Description `telCADigitsMsg` indicates that digits are being pressed by the destination user. (The physical terminal message handler would handle keys pressed at the local keypad.) `mtype` specifies whether or not the signaling is audible (in-band DTMF).

```
telDigitAudible      = 0;    { Digits audible }
telDigitNotAudible   = 1;    { Digits not audible }
```

`value` specifies the ASCII keypad digit sent by the remote user (a digit between 0 and 9, the number sign, or the asterisk). `msgInfo` is unused and set to zero.

telCADisconnectMsg

Description `telCADisconnectMsg` specifies that a call has been disconnected.

`mtype` specifies the party responsible for the disconnect:

```
telLocalDisconnect   = 0;    { This user responsible for disconnect}
telRemoteDisconnect  = 1;    { Remote party responsible for disconnect}
```

`Value` specifies the reason for the disconnect and is set to one of the following values:

```
telCADNormal          = 1;    { Normal disconnect }
telCADBusy            = 2;    { Remote user busy }
telCADNoResponse      = 3;    { Remote not responding }
telCADRejected        = 4;    { Call rejected }
telCADNumberChanged   = 5;    { Number changed }
telCADInvalidDest     = 6;    { Invalid destination address }
telCADFacilityRejected = 7;    { Requested facility rejected }
telCADUnobtainableDest = 9;    { Destination not obtainable }
telCADCongested       = 10;   { Network congestion }
telCADIncompatibleDest = 11;   { Incompatible destination }
telCADTimeout         = 12;   { Call timed out }
telCADUnknown         = 15;   { Reason unknown }
```

`msgInfo` is unused and set to zero.

`UserUserInfo` points to any user-to-user information sent with the disconnect from the remote end. Not all switches support the sending of user-to-user information at disconnect time.

telCAFaxToneMsg

Description telCAFaxToneMsg specifies that a fax tone has been detected on hTELCA. mType is unused and set to zero. value is as follows:

```
telFaxToneDetected    = 0;    { Fax Tone was detected }
telFaxToneCleared     = 1;    { Fax Tone went away }
```

msgInfo is unused and set to zero.

telCAForwardMsg

Description telCAForwardMsg is sent to the originator of a call appearance when the call has been forwarded to a new directory number. The receiver of the forwarded call will receive a telCAAlertingMsg OR telCAOfferMsg message specifying that the incoming call has been forwarded by another directory number. The set that forwarded the call will receive a telCADeflectMsg message.

mtype is unused and set to zero. value indicates the type of call forwarding that occurred and is set to one of the following values:

```
telForwardImmediate   = 1;    { Call forwarding immediate }
telForwardBusy        = 2;    { Call forwarding on busy }
telForwardNoAnswer    = 3;    { Call forwarding on no answer }
```

msgInfo points to the following structure:

```
TYPE
CAGenericMsgRec      = RECORD
    rmtDN              : StringPtr;
    rmtName            : StringPtr;
    rmtSubaddress      : StringPtr;
    dialType           : Short;
END
```

rmtDN, rmtName, and rmtSubaddress specify the directory number, user name, and subaddress (if any) to which the call was forwarded. dialType indicates the "dialability" of the number specified in rmtDN.

telCAHoldMsg

Description `telCAHoldMsg` specifies that hold activity is occurring on this call appearance. This message is sent when the user presses the Hold key or when your application calls `TELHold` or `TELRetrieve`. `mtype` is unused and set to zero. `value` is set to one of the following values:

<code>telHoldCleared</code>	<code>= 0;</code>	<code>{ hold was cleared }</code>
<code>telHoldEst</code>	<code>= 1;</code>	<code>{ hold established }</code>
<code>telHoldFailed</code>	<code>= 2;</code>	<code>{ hold could not be established }</code>

`msgInfo` is unused and set to zero.

telCAIdleMsg

Description `telCAIdleMsg` indicates that the state of the call has changed to `telCAIdleState`. `mtype`, `value`, and `msgInfo` are unused and set to zero.

telCAIntercomMsg

Description `telCAIntercomMsg` specifies that activity relating to the Intercom feature has occurred. This message is received after a call to `TELIntercom`. `mtype` specifies the intercom ID, if any. `value` is as follows:

<code>telIntercomEst</code>	<code>= 0;</code>	<code>{ Intercom was successful }</code>
<code>telIntercomComplete</code>	<code>= 1;</code>	<code>{ Intercom activity completed }</code>
<code>telIntercomFailed</code>	<code>= 2;</code>	<code>{ intercom failed }</code>

`msgInfo` is unused and set to zero.

telCAInUseMsg

Description `telCAInUseMsg` indicates that the specified call appearance is part of a multiple-access directory number (MADN) and is in use. `mtype` and `msgInfo` are unused and set to zero. `value` is set to one of the following values:

<code>inUsePrivate</code>	<code>= 0;</code>	<code>{ MADN in use; cannot be accessed }</code>
<code>inUseCanAccess</code>	<code>= 1;</code>	<code>{ MADN in use; others can join in }</code>
<code>inUseCanMakePrivate</code>	<code>= 2;</code>	<code>{ MADN in use; any MADN group member can access it and make it private }</code>
<code>inUseClear</code>	<code>= 3;</code>	<code>{ MADN no longer in use }</code>

telCAModemToneMsg

Description `telCAModemToneMsg` specifies that a modem tone has been detected on `hTELCA`. `mType` is unused and set to zero. `value` is as follows:

```
telModemToneDetected = 0;    { Modem Tone was detected }
telModemToneCleared  = 1;    { Modem Tone went away }
```

`msgInfo` is unused and set to zero.

telCAOfferMsg

Description `telCAOfferMsg` specifies that an incoming call `hTELCA` is being offered to this terminal.

`mtype` and `value` are unused and set to zero. `msgInfo` is unused and set to zero.

telCAOtherMsg

Description `telCAOtherMsg` is available for use by tools to relay tool-specific messages.

The tool defines the value of `mtype`, `value`, and `msgInfo`.

telCAOutgoingMsg

Description `telCAOutgoingMsg` specifies that an outgoing call has been initiated. `mType` is unused and set to zero. `value` is as follows:

```
telPhysical          = 0;    { User lifted handset and initiated call }
telProgrammatic      = 1;    { Outgoing call initiated programmatically }
```

If `value` is `telPhysical`, a new call appearance has been allocated, and its handle is available in `hTELCA`. If instead `value` is `telProgrammatic`, and your application initiated the telephone call, the value of `hTELCA` is equal to the `hTELCA` value returned by `TELSetupCall`. (If your application does not recognize `hTELCA`, some other application initiated the call.)

`rmtdN` is unused and set to zero. If the application wishes to know the `rmtdN` and name, it should inspect the record pointed to by `hTELCA`.

telCAPagingMsg

Description `telCAPagingMsg` specifies that activity relating to paging has occurred. This message is received after a call to `TELPaging`. `mType` specifies the paging ID, if any. `value` is as follows:

<code>telPageEst</code>	<code>= 0;</code>	<code>{ Paging was successful }</code>
<code>telPageComplete</code>	<code>= 1;</code>	<code>{ Paging activity completed }</code>
<code>telPageFailed</code>	<code>= 2;</code>	<code>{ Paging failed }</code>

`msgInfo` is unused and set to zero.

telCACallParkMsg

Description `telCACallParkMsg` gives information about activities related to Call Park features. This message is sent when the user presses a Call Park key or when your application calls `TELCallParkSet` or `TELCallParkClear`. `mType` is unused and set to zero. `value` is as follows:

<code>telCallParkEst</code>	<code>= 1;</code>	<code>{ Call has been successfully parked }</code>
<code>telCallParkRetrieveEst</code>	<code>= 2;</code>	<code>{ Parked call has been successfully retrieved }</code>
<code>telCallParkFailed</code>	<code>= 3;</code>	<code>{ Attempt to setup call park has failed }</code>
<code>telCallParkRetrieveFailed</code>	<code>= 4;</code>	<code>{ Attempt to retrieve parked call failed }</code>
<code>telCallParkRecall</code>	<code>= 5;</code>	<code>{ Call park has been recalled }</code>

`msgInfo` points to the following structure:

```
TYPE
    CAGenericMsgRec    = RECORD
        rmtDN           : StringPtr;
        rmtName         : StringPtr;
        rmtSubaddress   : StringPtr;
        dialType        : INTEGER;
    END
```

`rmtDN`, `rmtName`, and `rmtSubaddress` specify the directory number, user name, and subaddress against which the call was parked, if the network switch parks calls against directory numbers. `dialType` indicates the "dialability" of the number specified in `rmtDN`.

telCACallPickupMsg

Description

`telCACallPickupMsg` specifies the success or failure of a call pickup at the pickup end, or specifies to the call originator that the call was picked up at a different directory number. `mType` is unused and set to zero. `value` is as follows:

```
telCallPickupEst      = 0; { Call pickup was successful }
telCallPickupFailed   = 1; { Call pickup failed }
telCallPickedUp       = 2; { Message to originator that call was picked
                             up at a different dn }
```

`msgInfo` points to the following structure:

```
TYPE
CAGenericMsgRec      = RECORD
    rmtDN             : StringPtr;
    rmtName           : StringPtr;
    rmtSubaddress     : StringPtr;
    dialType          : INTEGER;
END
```

For `telCallPickupEst` and `telCallPickupFailed`, `rmtDN` and `rmtSubaddress` identify the directory number and subaddress whose call was picked up successfully or unsuccessfully. For `telCallPickedUp`, `rmtDN` and `rmtSubaddress` specify the directory number and subaddress at which the call was picked up.

`rmtName` identifies the name of the user associated with `rmtDN`. `dialType` indicates the "dialability" of the number specified in `rmtDN`.

telCAPProgressMsg

Description

telCAPProgressMsg reports on any progress of an outgoing call appearance specified by hTELCA. mtype is unused. value specifies the progress as one of the following value:

```
telCAPDialTone      = 1; { Dial tone }
telCAPRinging       = 2; { Destination CA is alerting }
telCAPDialing       = 3; { One or more digits has been dialed }
telCAPReorder       = 4; { Reorder }
telCAPBusy          = 5; { Busy }
telCAPRouted        = 6; { Call routed; rmtDN will hold the routing }
telCAPRoutedOff     = 7; { Call routed off-network; no further progress
                           will be available }
telCAPTimeout       = 8; { Call timed out }
telCAPUpdate        = 9; { name and rmtDN information has been updated }
telCAPPrompt        = 10; { The network is prompting for more
                           information }
telCAPWaiting       = 11; { Call is proceeding, but there is no response
                           yet from the destination }
telCAPUnknown       = 15; { Call Progress state unknown }
```

msgInfo points to the following structure:

```
TYPE
CAGenericMsgRec      = RECORD
  rmtDN               : StringPtr;
  rmtName             : StringPtr;
  rmtSubaddress       : StringPtr;
  dialType            : INTEGER;
END
```

If value is telCAPUpdate, rmtDN, rmtName, and rmtSubaddress specify the updated directory number, name, and subaddress of the remote end, if this information is available. The Telephone Manager stores these values in the call-appearance record referenced by hTELCA. If value is telCAPRouted, rmtDN, rmtName, and rmtSubaddress specify the updated directory number, name, and subaddress of the routing directory number. If value is neither telCAPUpdate nor telCAPRouted, rmtDN, rmtName, and rmtSubaddress are empty.

dialType indicates the "dialability" of the number specified in rmtDN.

telCAQueuedMsg

Description `telCAQueuedMsg` specifies that a call is being queued for this terminal. `mtype`, `value`, and `msgInfo` is unused and set to zero.

telCARejectMsg

Description `telCARejectMsg` gives information about activities related to Call Reject features. This message is sent when the user presses a Call Reject key or when your application calls `TELRejectCall`. `mtype` is the reason for disconnect, but is unused and set to zero. `value` is as follows:

<code>telCallRejectFailed</code>	<code>= 0;</code>	<code>{ attempt to reject call has failed }</code>
<code>telCallRejectEst</code>	<code>= 1;</code>	<code>{ call successfully rejected }</code>
<code>telCallRejected</code>	<code>= 2;</code>	<code>{ message to originator that call was rejected }</code>

`msgInfo` is unused and set to zero.

telCASuccessiveAlert

Description `telCASuccessiveAlert` is sent by the tool each time the attached phone “rings.” `mtype`, `value`, and `msgInfo` are unused and set to zero.

telCATransferMsg

Description

telCATransferMsg specifies that transfer activity is occurring for hTELCA. This activity could be occurring because of calls to TELTransferPrep, TELTransferEstablish, or TELTransferBlind, or because the user is manually initiating a transfer. mtype is unused and set to zero. value is set to one of the following values:

```
telTransferPrepFailed = 0; { transfer could not be prepared}
telTransferPending    = 1; { transfer prepared successfully}
telTransferEst        = 2; { consult or blind transfer successful}
telTransferFailed     = 3; { consult or blind transfer failed }
telTransferred        = 4; { message to originator of CA specifying
                             that the call was transferred to rmtDN
                             and that CAProgress messages will give
                             new rmtDN }
```

msgInfo points to the following structure:

```
TYPE
CATransfMsgRec      = RECORD
    rmtDN            : StringPtr;
    rmtName          : StringPtr;
    rmtSubaddress    : StringPtr;
    dialType         : INTEGER;
    prepCA           : TELCAHandle;
END
```

If value equals telTransferEst or telTransferFailed, prepCA specifies a handle to the prepared hTELCA; rmtDN, rmtName, and rmtSubaddress specify the directory number, user, and subaddress to which the call was transferred. If value is neither telTransferEst nor telTransferFailed, then prepCA, rmtDN, rmtName, and rmtSubaddress are set to zero.

telCAUserUserInfoMsg

Description

telCAUserUserInfoMsg specifies that transfer activity is occurring for hTELCA. mtype and value are unused and set to zero.

msgInfo points to the following structure:

```
TYPE
CAUserUserInfoMsgRec = RECORD
    userUserInfo     : StringPtr;
END
```

userUserInfo points to any user-to-user information sent with the disconnect from the remote end. Not all switches support the sending of user-to-user information at disconnect time.

Directory-number message codes for applications

The message codes in this section relate to specific directory numbers. For information on the format and parameters of directory-number messages, refer to the template for directory-number message handlers (`MyDNMsgHandler`) in the section "Routines Your Application Must Provide," in Chapter 2.

`telDNDNDMsga`

Description

`telDNDNDMsg` gives information about activities related to Do Not Disturb features. This message is sent after application calls to `TELDNDSet` and `TELDNDClr`. `mType` is the type of Do Not Disturb feature, as follows:

```
telDNDIntExt      = 0;    { Do Not Disturb for all inside and outside calls }
telDNDEsternal    = 1;    { Do Not Disturb for outside calls only }
telDNDInternal    = 2;    { Do Not Disturb for inside calls only }
telDNDNonIntercom = 3;    { DND for all calls except intercom calls }
```

value is as follows:

```
telDNDcleared = 0;    { Do Not Disturb has been cleared }
telDNDEst     = 1;    { Do Not Disturb has been established }
telDNDFailed  = 2;    { Attempt to setup Do Not Disturb has failed }
```

`rmtDN`, `rmtName`, and `rmtSubaddress` are unused.

`telDNForwardMsg`

Description

`telDNForwardMsg` gives information about activities related to call forwarding. This message is received when `TELForwardSet` and `TELForwardClr` are called. `mType` is the type of Call Forward feature, as follows:

```
telForwardImmediate = 1;    { Immediately forward calls }
telForwardBusy      = 2;    { Forward on Busy }
telForwardNoAnswer  = 3;    { Forward on No answer }
telForwardBusyNA    = 4;    { Forwarding for busy and no answer }
```

value is as follows:

```
telForwardCleared = 0;    { Forwarding has been cleared }
telForwardEst     = 1;    { Forwarding has been established }
telForwardFailed  = 2;    { attempt to setup forwarding has failed }
```

`rmtDN` specifies the remote directory number to which the call was forwarded. `rmtName` specifies the name of the party associated with the remote directory number, if this information is available. `rmtSubaddress` specifies the subaddress, if any, of the remote directory number.

telDNOtherMsg

Description The custom message parameters are defined by individual telephone tools for use specific to those tools.

telDNSelectedMsg

Description `telDNSelectedMsg` indicates that the directory number specified in this message has been either selected or deselected. This message is sent in response to a `TELDNSelect` call. `value` is one of the following values:

```
telDNdeselected        = 0;    { dn has been deselected }
telDNSelected          = 1;    { dn has been selected }
```

All other fields are unused.

telDNVoiceMailMsg

Description `telDNVoiceMailMsg` specifies that a voice-mail message has arrived for this directory number. `mtype` is unused. `value` is assigned the following values:

```
telAllVoiceMessagesRead = 0; { all messages have been read,
                               none are waiting to be read}
telNewVoiceMessage      = 1; { a new message has arrived, or
                               messages are waiting for this dn}
```

`rmtDN` specifies the calling directory number of the party leaving the message, if available.

`rmtName` specifies the name of the party leaving the message, if available.

`rmtSubaddress` specifies the subaddress, if any, of `rmtDN`.

Terminal message codes for applications

The message codes in this section relate to a specific terminal. For information on the format and parameters of terminal messages, refer to the template for terminal message handlers (`MyTermMsgHandler`) in the section "Routines Your Application Must Provide," in Chapter 2.

`telTermCloseMsg`

Description `telTermCloseMsg` specifies that the terminal has shut down. This message is sent after a successful `TELCloseTerm` call. `mtype` and `value` are unused and set to zero.

`telTermDisplayMsg`

Description `telTermDisplayMsg` specifies that the display has changed. This message is sent when the terminal display has been changed—for instance, as a result of a new incoming call. `mtype` contains the display mode and is one of the following values:

<code>telNormalDisplayMode</code>	<code>= 1;</code>	<code>{ Normal display mode }</code>
<code>telInspectMode</code>	<code>= 2;</code>	<code>{ Inspect display mode }</code>
<code>telMiscMode</code>	<code>= 3;</code>	<code>{ Miscellaneous display mode }</code>
<code>telRetrieveMode</code>	<code>= 4;</code>	<code>{ Message retrieval mode }</code>
<code>telDirectoryQueryMode</code>	<code>= 5;</code>	<code>{ Electronic directory mode }</code>

`value` contains the type of information which has changed in the display and is one of the following values:

<code>telEntireDisplay</code>	<code>= 0;</code>	<code>{ Entire display }</code>
<code>CallAppID</code>	<code>= 1;</code>	<code>{ Network assigned CA ID }</code>
<code>CalledDN</code>	<code>= 2;</code>	<code>{ Called party dn }</code>
<code>CallingDN</code>	<code>= 3;</code>	<code>{ Calling party dn }</code>
<code>CalledName</code>	<code>= 4;</code>	<code>{ Called party name }</code>
<code>CallingName</code>	<code>= 5;</code>	<code>{ Calling party name }</code>
<code>OrigPermissions</code>	<code>= 6;</code>	<code>{ Originating permissions }</code>
<code>DateandTime</code>	<code>= 7;</code>	<code>{ Date and time of day }</code>
<code>toolspecific</code>		<code>{This value for other types that may be supported by a particular Telephone Tool. Values range from 256 to 32768}</code>

`{values from 8 to 255 are reserved by the Telephone Manager, and
will not be passed in this message }`

telTermEnableMsg

Description `telTermEnableMsg` specifies whether the tool is able to communicate with the terminal. If so, commands can be sent and messages can be received.

`mtype` is unused and set to zero.

`value` has one of the following values:

<code>terminalEnabled</code>	<code>= 0; { Can communicate with the terminal }</code>
<code>terminalDisabled</code>	<code>= 1; { Cannot communicate with the terminal }</code>

telTermErrorMsg

Description `telTermErrorMsg` specifies that the terminal has had some kind of hard equipment failure. `mtype` and `value` are reserved.

telTermHookMsg

Description `telTermHookMsg` specifies that the hookswitch state has changed. This message is sent when the telephone is physically placed on-hook or off-hook, or when `TELSetHookSw` is called.

`value` specifies the hookswitch state and is one of the following values:

<code>deviceOnHook</code>	<code>= 0; { device on hook }</code>
<code>deviceOffHook</code>	<code>= 1; { device off hook }</code>

`mtype` specifies the type of device and is one of the following values:

<code>telHandset</code>	<code>= 1; { handset hookswitch }</code>
<code>telSpeakerphone</code>	<code>= 2; { speakerphone 'on' switch }</code>
<code>toolspecific</code>	<code>{This value for other types that may be supported by a particular Telephone Tool. Values range from 256 to 32768}</code>

`{values from 3 to 255 are reserved by the Telephone Manager and will not be passed in this message }`

telTermKeyMsg

Description `telTermKeyMsg` specifies that keys on the telephone pad or feature keys have been pressed. This message is sent if the keys are pressed on the handset. If keys are "pressed" programmatically, the application will be notified through normal use of directory-number and call-appearance message handlers.

If the user presses a key on the Macintosh keypad, `value` contains the ASCII value of the key. If the user presses a feature key, `value` contains one of the following values:

<code>telHangupKey</code>	<code>= 1;</code>	<code>{ Drop, or release, key pressed }</code>
<code>telHoldKey</code>	<code>= 2;</code>	<code>{ Hold Key pressed }</code>
<code>telConferenceKey</code>	<code>= 3;</code>	<code>{ Conference Key pressed }</code>
<code>telTransferKey</code>	<code>= 4;</code>	<code>{ Transfer Key pressed }</code>
<code>telForwardKey</code>	<code>= 5;</code>	<code>{ Call Forward Key pressed }</code>
<code>telCallbackKey</code>	<code>= 6;</code>	<code>{ Callback Key pressed }</code>
<code>telDNDKey</code>	<code>= 7;</code>	<code>{ Do Not Disturb Key pressed }</code>
<code>telCallPickupKey</code>	<code>= 8;</code>	<code>{ Call Pickup Key pressed }</code>
<code>telCallParkKey</code>	<code>= 9;</code>	<code>{ Call Park Key pressed }</code>
<code>telCallDeflectKey</code>	<code>= 10;</code>	<code>{ Call Deflect Key pressed }</code>
<code>telVoiceMailAccessKey</code>	<code>= 11;</code>	<code>{ Message Waiting Key pressed }</code>
<code>telCallRejectKey</code>	<code>= 12;</code>	<code>{ Call Reject Key pressed }</code>
<code>VoiceMailKeyPressed</code>	<code>= 13;</code>	<code>{ Voice Mail Key pressed }</code>
<code>telOtherKey</code>	<code>= 16;</code>	<code>{ Other Key pressed }</code>

`mtype` contains one of the following values:

<code>telKeyPadPress</code>	<code>= 1;</code>	<code>{ Key pressed on 12-digit keypad }</code>
<code>telFeatureKeyPress</code>	<code>= 2;</code>	<code>{ Feature Key Pressed }</code>

`telTermOtherMsg`

Description `telTermOtherMsg` is a vendor-defined message: Its meaning varies, depending on the tool that sends it. `value` should contain an error code, and `mtype` should be set if appropriate.

`telTermOpenMsg`

Description `telTermOpenMsg` specifies that the terminal has been opened. This message is sent after a successful `TELOpenTerm` call. `mtype` and `value` are unused and set to zero.

`telTermResetMsg`

Description `telTermResetMsg` specifies that the terminal has been reset. This message is sent after a successful `TELResetTerm` call. `mtype` and `value` are unused and set to zero.

telTermVolMsg

Description `telTermVolMsg` specifies that the volume type `mtype` has been changed to the level `value`. This message will be sent if the user changes the volume, either using the handset or by means of `TELSetVolume` or `TELAlert`.

`mtype` contains one of the following values:

<code>telHandsetSpeakerVol</code>	<code>= 1;</code>	<code>{ volume of the handset speaker }</code>
<code>telHandsetMicVol</code>	<code>= 2;</code>	<code>{ sensitivity of the handset mic }</code>
<code>telSpeakerphoneVol</code>	<code>= 3;</code>	<code>{ speakerphone volume }</code>
<code>telSpeakerphoneMicVol</code>	<code>= 4;</code>	<code>{ sensitivity of the spkrphone mic }</code>
<code>telRingerVol</code>	<code>= 5;</code>	<code>{ volume of the ringer }</code>

`value` contains the new volume level, which can range from 1 to `telVolMax`.

Appendix C **Call-Appearance States**

THIS APPENDIX lists and describes the call-appearance states recognized by the Telephone Manager.

State Code	Meaning
<code>telCAActiveState</code>	This call is active; parties are free to exchange information.
<code>telCAAlertingState</code>	A call is alerting at this terminal.
<code>telCABusyState</code>	The destination is busy or cannot be reached.
<code>telCAConferencedHeldState</code>	This call is part of a conference and has been put on hold by this terminal.
<code>telCAConferencedState</code>	This call is now part of a conference.
<code>telCADialingState</code>	This initiated outgoing call is dialing.
<code>telCADialToneState</code>	This initiated outgoing call has a dial tone.
<code>telCAHeldState</code>	This call has been put on hold by this terminal.
<code>telCAIdleState</code>	No call exists at this time.
<code>telCAInUseState</code>	The call is active at another terminal.
<code>telCAOfferState</code>	A call is being offered to this terminal.
<code>telCAQueuedState</code>	A call is being queued at this terminal.
<code>telCAREorderState</code>	This call is in the reorder state.
<code>telCARingingState</code>	This outgoing call is ringing.
<code>telCAUnknownState</code>	The state of this call is unknown.
<code>telCAWaitingState</code>	This initiated outgoing call is waiting for a response from the destination.

Index

A

active state, description of 5
alerting state, description of 5
application, your
 call-appearance record and 22
 definition of 7
 directory-number record and 15
 message handler templates for 96

B

bearerType field 26
bForwardDN field 20
bForwardPartyName field 20
bForwardSubaddress field 20
blind transfer 61, 147
bundle resource 98–99

C

CA. *See* call appearances
call-appearance messages
 for conference calls 159–160
 descriptions of 170–182
 general 151–155
 incoming/outgoing 155–158
 for transferring calls 158–159
call-appearance record 22–29
call appearances. *See also* outgoing call appearances
 accepting 56, 117
 answering 56, 118
 connecting 126
 controlling 90–93
 counting 90, 126
 deflecting 58
 description of 5
 disposing of handle for 93, 118

 dropping 59, 132, 153, 160
 finding 90–91
 finding state of features of 92
 finding state of 92
 finding supported messages 53
 forwarding 62
 getting information about 91
 maximum number of 15, 17
 message handler for
 clearing 52
 registering 50–51
 template for 96
 parking 68–69, 142
 picking up 68
 putting on hold 60, 138
 queuing 155
 reference number for 23
 rejecting 57, 142–143
 retrieving those held 60, 143–144
 retrieving those parked 69, 144
 sending message on activity of
 121–122
 setting up 145
 states of 190
Call Back feature 152
 clearing 119
 messages for 171
 requesting 65
 setting 120
 showing state of 28
Call Back Now feature 66, 119–120
 showing state of 28
Call Deflect feature 127, 152
 messages for 173
Call Forward feature 62, 153
 change in status of 161
 clearing 133

 messages for 175, 183
 setting 133
Call Park feature 68–69, 142, 154
 messages for 178
 showing state of 28
Call Park Retrieve feature 28
Call Pickup feature 68, 120–121, 154
 messages for 179
 showing state of 18–19
Call Reject feature 155
 messages for 181
callType field 25
caRef field 23
caState field 24
 updating 135
Communications Resource Manager 99
Conference Drop feature 27
Conference feature
 call-appearance messages for
 159–160
 establishing 64, 124–125
 maximum number of parties for 26
 messages for 172–173
 preparing for 63, 125
 showing state of 27
Conference Split feature 64, 125
 showing state of 27
config field 11
configuration fields, validating 33
configuration record 111
configuration strings 40–41, 109–110
 converting 75–76
confLimit field 26
connectTime field 24
Consult transfer 60–61
controls, setting up 37

curAllocCAS field
in directory-number data structure 17
in **TELTermRecord** data structure 15

D

data structures 8-29
call appearance 23-29
for directory-number record 15-21
for telephone record 10-15
defproc field 11
device states
monitoring 78-79
setting 80-81
dialType field 25
directory-number messages 160-162
descriptions of 183-184
directory numbers
call appearances and 5
controlling 85-89
counting 14, 85, 126-127
description of 4
disposing of handle for 89, 128-129
features supported by 129
finding by index 86
finding by name 87
finding state of features of 88
finding supported messages 52
getting information about 87
locating 129-130
message handler for
clearing 51
registering 49
template for 96
reference numbers for 16
selecting 88, 131, 162
Directory Number Select feature 18
displayRows field 14
displayText 14
changing 163
getting current 135
messages for 185
monitoring 83
setting 84, 144
DN. *See* directory numbers
dn field 17
dnPartyName field 17
dnRef field 16
dnSubaddress field 17
dnType field 17

Do Not Disturb feature 18, 66-67
change in status of 161
clearing 128
messages for 183
setting 129
Drop feature 59

E

English, American 75-76, 109-110
events
activate 45
for custom tool-settings dialog box 37-38
menu 45
message for 132
resume 45

F, G

Fax tone 153, 175
featureFlags field 13
in call-appearance data structure 27
in directory-number data structure 17-19
updating
in call-appearance record 133-134
in directory-number record 135-136
flags field 10-11
Forward feature. *See* Call Forward feature
forwardFlags field 19
Forward On Busy And No Answer feature 19
Forward On Busy feature 19-20
Forward On No Answer feature 19-20

H

handles 17
call appearances and 23
disposing of 128
call-appearance 93
directory-number 89
handsetMicVol field 14
handsets 14
handsetSpeakerVol field 14
hardware 3-4
failure messages for 186
hasDisplay field 14
held state, description of 5

Hold feature 60
messages for 138, 153, 176
showing state of 27
hookswitch
finding current state of 137
messages for 186
monitoring 77
setting 78, 144-145
hTELDN field 23
hTEL field
in call-appearance data structure 24
in directory-number data structure 17

I, J, K

idle state 138, 153
description of 5
messages for 176
idle-loop tasks 43
iForwardDN field 20
iForwardPartyName field 20
iForwardSubaddress field 20
Immediate Call Forwarding feature 19-20
incoming call appearances
messages 155-158
state progression of 5
information, getting from a terminal 44
InitTEL routine 31
Intercom feature 18-19, 153
activating 138-139
messages for 176
using 71
interface, physical 4
intExt field 25
ISDN 2

L

localization code resource 98, 109-110
logical directory numbers 4

M

Macintosh Communications Toolbox 2
Macintosh computers 6
Telephone Manager concepts and 4
Macintosh Device Manager 4
main code resource 98, 114
messages accepted by 115-116
messages sent by 149-150
master message handlers 149

maxAllocCAs field
 in directory-number data structure 17
 in TELTermRecord data
 structure 15
 menu events 45, 139
 message handler templates 96-97
 messages
 handling 47
 sending 3
 microphones 14
 modem tone 154, 177
 multiple-access directory numbers
 153, 176
 MyCAMsgHandler routine 96
 MyDNMsgHandler routine 96
 MyTermMsgHandler routine 95

N

naForwardDN field 20
 naForwardPartyName field 20
 naForwardRings field 20
 naForwardSubaddress field 20
 networks
 telephone 2
 Telephone Manager routines and 8
 network switches 4, 47
 implementing features for 73
 listing features for 72-73
 numDNs field 14
 numIntercomIDs field 19
 numPageIDs field 19
 numPickupIDs field 19

O

oldConfig field 11
 otherFeatures field 27-28
 updating
 in call-appearance record 133-134
 in directory-number record
 135-136
 otherVol field 14
 outgoing call appearances
 change of state in 154
 connecting 55
 dialing 55
 messages 155-158
 setting up 54-55
 state progression of 5

P, Q

Paging feature 18-19, 154
 accessing 70-71
 activating 141-142
 messages for 178
 parameters, passing of 3
 physical directory numbers 4
 priority field 26
 procID field 10
 getting 31
 pTELTerm field 12
 pTELTermSize field 12

R

rate field 26
 records 6
 refCon field
 in call-appearance data structure 28
 in directory-number data structure 21
 in TELRecord data structure 11
 relatedCA field 24
 reserved1 field 12
 reserved2 field 12
 reserved field
 in call-appearance data field 29
 in directory-number data structure 21
 in TELRecord data structure 11
 in TELTermRecord data
 structure 15
 resource IDs 99
 result codes 166-168
 ringerTypes field 14
 ringerVol field 14
 rmtDN field 26
 rmtPartyName field 26
 rmtSubaddress field 26
 routedDN field 26
 routePartyName field 26
 routeSubaddress field 26

S

scripting language
 code resource 98, 107-109
 interfacing with 40-41
 setup-definition code resource 98,
 102-106
 speakerphones 14
 speakerphoneVol field 14

strings

configuration 40-41, 109-110
 converting 75-76
 Pascal-style 8
 system requirements 6

T

telAcceptCallMsg message 117
 TELAcceptCall routine 56
 telActivateMsg message 117
 TELActivate routine 45
 telAlertMsg message 118
 TELAlert routine 82
 telAnswerCallMsg message 118
 TELAnswerCall routine 56
 telCAActiveMsg message 152
 description of 170
 telCAAlertingMsg message 157
 description of 170
 telCACallbackMsg message 152
 description of 171
 telCACallParkMsg message 154
 description of 178
 telCACallPickupMsg message
 154
 description of 179
 telCAConferenceDropMsg
 message 160
 description of 172
 telCAConferenceMsg message
 160
 description of 172
 telCAConferenceSplitMsg
 message 160
 description of 172-173
 telCADeflectMsg message 152
 description of 173
 telCADigitsMsg message 152
 description of 174
 telCADisconnectMsg message
 153
 description of 174
 telCADisposeMsg message 118
 TELCADispose routine 93
 telCAEventsSuppMsg message
 119
 TELCAEventsSupp routine 53
 telCAFaxToneMsg message 153
 description of 175
 telCAForwardMsg message 153
 description of 175

APPLE COMPUTER, INC. SOFTWARE LICENSE

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, PROMPTLY RETURN THE UNUSED SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT AND YOUR MONEY WILL BE REFUNDED.

1. License. The application, demonstration, system and other software accompanying this License, whether on disk, in read only memory, or on any other media (the "Apple Software") and related documentation are licensed to you by Apple. You own the disk on which the Apple Software is recorded but Apple and/or Apple's Licensor(s) retain title to the Apple Software and related documentation. This License allows you to use the Apple Software on a single Apple computer and make one copy of the Apple Software in machine-readable form for backup purposes only. You must reproduce on such copy the Apple copyright notice and any other proprietary legends that were on the original copy of the Apple Software. You may also transfer all your license rights in the Apple Software, the backup copy of the Apple Software, the related documentation and a copy of this License to another party, provided the other party reads and agrees to accept the terms and conditions of this License.

2. Restrictions. The Apple Software contains copyrighted material, trade secrets and other proprietary material and in order to protect them you may not decompile, reverse engineer, disassemble or otherwise reduce the Apple Software to a human-perceivable form. You may not modify, network, rent, lease, loan, distribute or create derivative works based upon the Apple Software in whole or in part. You may not electronically transmit the Apple Software from one computer to another or over a network.

3. Support. You acknowledge and agree that Apple may not offer any technical support in the use of the Software.

4. Termination. This License is effective until terminated. You may terminate this License at any time by destroying the Apple Software and related documentation and all copies thereof. This License will terminate immediately without notice from Apple if you fail to comply with any provision of this License. Upon termination you must destroy the Apple Software and related documentation and all copies thereof.

5. Export Law Assurances. You agree and certify that neither the Apple Software nor any other technical data received from Apple, nor the direct product thereof, will be exported outside the United States except as authorized and as permitted by the laws and regulations of the United States.

6. Government End Users. If you are acquiring the Apple Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees:

(i) if the Apple Software is supplied to the Department of Defense (DoD), the Apple Software is classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Apple Software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and

(ii) if the Apple Software is supplied to any unit or agency of the United States Government other than DoD, the Government's rights in the Apple Software and its documentation will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.

7. Limited Warranty on Media. Apple warrants the disks on which the Apple Software is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of purchase as evidenced by a copy of the receipt. Apple's entire liability and your exclusive remedy will be replacement of the disk not

meeting Apple's limited warranty and which is returned to Apple or an Apple authorized representative with a copy of the receipt. Apple will have no responsibility to replace a disk damaged by accident, abuse or misapplication. ANY IMPLIED WARRANTIES ON THE DISKS, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

8. Disclaimer of Warranty on Apple Software. You expressly acknowledge and agree that use of the Apple Software is at your sole risk. The Apple Software and related documentation are provided "AS IS" and without warranty of any kind and Apple and Apple's Licensor(s) (for the purposes of provisions 8 and 9, Apple and Apple's Licensor(s) shall be collectively referred to as "Apple") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. APPLE DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE APPLE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE APPLE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE APPLE SOFTWARE WILL BE CORRECTED. FURTHERMORE, APPLE DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE APPLE SOFTWARE OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE APPLE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

9. Limitation of Liability. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL APPLE BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE APPLE SOFTWARE OR RELATED DOCUMENTATION, EVEN IF APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

In no event shall Apple's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Apple Software.

10. Controlling Law and Severability. This License shall be governed by and construed in accordance with the laws of the United States and the State of California, as applied to agreements entered into and to be performed entirely within California between California residents. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this License shall continue in full force and effect.

11. Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the Apple Software and related documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Apple.