PALM COMPUTING® PLATFORM™

Palm OS® SDK Reference

Document Number 3003-002
Print Date 3/00

CONTRIBUTORS

Written by Christopher Bey, Elly Freeman, Dwayne Mulder, and Jean Ostrem
Production by <dot>**PS** document production services
Engineering contributions by David Fedor, Roger Flores, Steve Lemke, Bob Ebert, Ken Krugler, Bruce Thompson, Jesse Donaldson, Tim Wiegman, Gavin Peacock, Ryan Robertson, and Waddah Kudaimi

Palm OS SDK Reference
Document Number 3003-002
March 16, 2000

Palm, Inc.
5400 Bayfront Plaza
Santa Clara, CA 95052
USA

www.palm.com/devzone

Table of Contents

2 Palm OS Resources 75

3 Palm OS Events 105

Part II: System Management

Part III: Communications

56 New Serial Manager 1015

Part IV: Libraries

About This Document

Palm OS SDK Reference is part of the Palm OS® Software Development Kit. This introduction provides an overview of SDK documentation, discusses what materials are included in this document, and what conventions are used.

Palm OS SDK Documentation

The following documents are part of the SDK:

Document	Description
Palm OS SDK Reference	An API reference document that contains descriptions of all Palm OS function calls and important data structures.
Palm OS Programmer's Companion	A guide to application programming for the Palm OS. This volume contains conceptual and "how-to" information that complements the Reference.
CodeWarrior Constructor for the Palm OS Platform	A guide to using CodeWarrior Constructor to create Palm OS resource files.
Palm OS Programming Development Tools Guide	A guide to writing and debugging Palm OS applications with the various tools available.

What This Volume Contains

This section provides an overview of this volume.

- Part I, "User Interface," documents the API contained in the header files in the \Incs\Core\UI\ folder. This part contains chapters covering subjects such as application launch codes,

user interface resources, events, and all window, form, and field object managers.

- Part II, "System Management," documents the API contained in the header files in the \Incs\Core\System\ folder. This part contains chapters covering subjects such as the alarm manager, data and resource manager, feature manager, float manager, graffiti manager, key manager, memory manager, preferences manager, sound manager, string manager, and system manager.
- Part III, "Communications," documents the API related to communications, such as the exchange manager, IR library, net library, serial manager, and serial drivers.
- Part IV, "Libraries," documents the API contained in the header files in the \Incs\Libraries\ folder. This part contains chapters covering the Internet Library and the Palm OS Glue library.

Conventions Used in This Guide

This guide uses the following typographical conventions:

This style...	Is used for...
`fixed width font`	Code elements such as function, structure, field, bitfield.
`fixed width underline`	Emphasis (for code elements).
bold	Emphasis (for other elements).
blue and underlined	Hot links.
black and underlined	New function in one of the current releases (headings only)
red and underlined	New function in one of the current releases (Table of Contents only)

Part I: User Interface

1

Application Launch Codes

This chapter provides detailed information about the predefined application launch codes. Launch codes are declared in the header file `SystemMgr.h`. The associated parameter blocks are declared in `AppLaunchCmd.h`, `AlarmMgr.h`, `ExgMgr.h`, and `Find.h`.

Table 1.1 lists all Palm OS® standard launch codes. More detailed information is provided immediately after the table:

- Launch Codes
- Launch Flags

To learn what a launch code is and how to use it, see the chapter titled "Application Startup and Stop" in the *Palm OS Programmer's Companion*.

Table 1.1 Palm OS Launch Codes

Code	Request
`scptLaunchCmdExecuteCmd`	Execute the specified Network login script plugin command.
`scptLaunchCmdListCmds`	Provide information about the commands that your Network script plugin executes.
`sysAppLaunchCmdAddRecord`	Add a record to a database.
`sysAppLaunchCmdAlarmTriggered`	Schedule next alarm or perform quick actions such as sounding alarm tones.
`sysAppLaunchCmdCountryChange`	Respond to country change.
`sysAppLaunchCmdDisplayAlarm`	Display specified alarm dialog or perform time-consuming alarm-related actions.

Table 1.1 Palm OS Launch Codes *(continued)*

Code	Request
sysAppLaunchCmdExgAskUser	Let application override display of dialog asking user if they want to receive incoming data via the exchange manager.
sysAppLaunchCmdExgReceiveData	Notify application that it should receive incoming data via the exchange manager.
sysAppLaunchCmdFind	Find a text string.
sysAppLaunchCmdGoto	Go to a particular record, display it, and optionally select the specified text.
sysAppLaunchCmdGoToURL	Launch Clipper application and open a URL.
sysAppLaunchCmdInitDatabase	Initialize database.
sysAppLaunchCmdLookup	Look up data. In contrast to sysAppLaunchCmdFind, a level of indirection is implied. For example, look up a phone number associated with a name.
sysAppLaunchCmdNormalLaunch	Launch normally.
sysAppLaunchCmdNotify	Notify about an event.
sysAppLaunchCmdOpenDB	Launch application and open a database.
sysAppLaunchCmdPanelCalledFromApp	Tell preferences panel that it was invoked from an application, not the Preferences application.
sysAppLaunchCmdReturnFromPanel	Tell an application that it's restarting after preferences panel had been called.
sysAppLaunchCmdSaveData	Save data. Often sent before find operations.

Table 1.1 Palm OS Launch Codes *(continued)*

Code	Request
`sysAppLaunchCmdSyncNotify`	Notify applications that a HotSync has been completed.
`sysAppLaunchCmdSystemLock`	Sent to the Security application to request that the system be locked down.
`sysAppLaunchCmdSystemReset`	Respond to system reset. No UI is allowed during this launch code.
`sysAppLaunchCmdTimeChange`	Respond to system time change.
`sysAppLaunchCmdURLParams`	Launch an application with parameters from Clipper.

Launch Codes

This section provides supplemental information about launch codes. For some launch codes, it lists the parameter block, which in some cases provides additional information about the launch code.

sysAppLaunchCmdAddRecord

Add a record to an application's database.

This launch code is used to add a message to the Mail or iMessenger™ (on the Palm VII™ organizer) application's outbox. You pass information about the message such as address, body text, etc. in the parameter block. For iMessenger, you can set the `edit` field of the parameter block to control whether or not the iMessenger editor is displayed. Set it to `true` to display the editor or `false` not to display it.

For more information on sending messages via iMessenger, see "Sending Messages" on page 312 in the *Palm OS Programmer's Companion*.

IMPORTANT: Implemented for iMessenger only if Wireless Internet Feature Set is present. Implemented for Mail only on OS version 3.0 or later.

sysAppLaunchCmdAddRecord Parameter Block for Mail Application

Prototype

```
typedef enum {
  mailPriorityHigh,
  mailPriorityNormal,
  mailPriorityLow
} MailMsgPriorityType;

typedef struct {
  Boolean secret;
  Boolean signature;
  Boolean confirmRead;
  Boolean confirmDelivery;
  MailMsgPriorityType priority;
  UInt8 padding
  Char* subject;
  Char* from;
  Char* to;
  Char* cc;
  Char* bcc;
  Char* replyTo;
  Char* body;
} MailAddRecordParamsType;
```

Fields

`secret`	`True` means that the message should be marked secret.
`signature`	`True` means that the signature from the Mail application's preferences should be attached to the message.
`confirmRead`	`True` means that a confirmation should be sent when the message is read.

`confirmDelivery`	`True` means that a confirmation should be sent when the message is delivered.
`priority`	Message priority. Specify one of the `MailMsgPriorityType` enumerated types.
`padding`	Reserved for future use.
`subject`	Message's subject, a null-terminated string (optional).
`from`	Message's sender, a null-terminated string (not used on outgoing mail).
`to`	Address of the recipient, a null-terminated string (required).
`cc`	Addresses of recipients to be copied, a null-terminated string (optional).
`bcc`	Addresses of recipients to be blind copied, a null-terminated string (optional).
`replyTo`	Reply to address, a null-terminated string (optional).
`body`	The text of the message, a null-terminated string (required).

sysAppLaunchCmdAddRecord Parameter Block for iMessenger Application

Prototype

```
typedef struct {
  UInt16 category;
  Boolean edit;
  Boolean signature;
  Char *subject;
  Char *from;
  Char *to;
  Char *replyTo;
  Char *body;
} MsgAddRecordParamsType;
```

Fields

`category`	Category in which to place the message. Specify one of the following categories:
	`MsgInboxCategory`
	`MsgOutboxCategory`
	`MsgDeletedCategory`
	`MsgFiledCategory`
	`MsgDraftCategory`

`edit`	`True` means that the message should be opened in the editor. `False` means that the message should simply be placed into the outbox and the editor not opened. You can specify `true` only if the category is set to `MsgOutboxCategory`.
`signature`	`True` means that the signature from the iMessenger application preferences should be attached to the message.
`subject`	Message's subject, a null-terminated string (optional).
`from`	Message's sender, a null-terminated string (not used on outgoing mail).
`to`	Address of the recipient, a null-terminated string (required).
`replyTo`	Reply to address, a null-terminated string (optional).
`body`	The text of the message, a null-terminated string (required).

sysAppLaunchCmdAlarmTriggered

Performs quick action such as scheduling next alarm or sounding alarm.

This launch code is sent as close to the actual alarm time as possible. An application may perform any quick, non-blocking action at this time. Multiple alarms may be pending at the same time for multiple applications, and one alarm display shouldn't block the system and

prevent other applications from receiving their alarms in a timely fashion. An opportunity to perform more time-consuming actions will come when sysAppLaunchCmdDisplayAlarm is sent.

sysAppLaunchCmdAlarmTriggered Parameter Block

Prototype

```
typedef struct SysAlarmTriggeredParamType {
  UInt32  ref;
  UInt32  alarmSeconds;
  Boolean purgeAlarm;
  UInt8   padding;
} SysAlarmTriggeredParamType;
```

Fields

`-> ref`	The caller-defined value specified when the alarm was set with AlmSetAlarm.
`-> alarmSeconds`	The date/time specified when the alarm was set with `AlmSetAlarm`. The value is given as the number of seconds since 1/1/1904.
`<- purgeAlarm`	Upon return, set to `true` if the alarm should be removed from the alarm table. Use this as an optimization to prevent the application from receiving sysAppLaunchCmdDisplayAlarm if you don't wish to perform any other processing for this alarm. If you do want to receive the launch code, set this field to `false`.
`padding`	Not used.

sysAppLaunchCmdCountryChange

Responds to country change.

Applications should change the display of numbers to use the proper number separators. To do this, call LocGetNumberSeparators, `StrLocalizeNumber`, and `StrDelocalizeNumber`.

sysAppLaunchCmdDisplayAlarm

Performs full, possibly blocking, handling of alarm.

This is the application's opportunity to handle an alarm in a lengthy or blocking fashion. Notification dialogs are usually displayed when this launch code is received. This work should be done here, not when sysAppLaunchCmdAlarmTriggered is received. Multiple alarms may be pending at the same time for multiple applications, and one alarm display shouldn't block the system and prevent other applications from receiving their alarms in a timely fashion.

sysAppLaunchCmdDisplayAlarm Parameter Block

Prototype

```
typedef struct SysDisplayAlarmParamType {
  UInt32  ref;
  UInt32  alarmSeconds;
  Boolean soundAlarm;
  UInt8   padding;
  } SysDisplayAlarmParamType;
```

Fields

-> ref	The caller-defined value specified when the alarm was set with AlmSetAlarm.
-> alarmSeconds	The date/time specified when the alarm was set with AlmSetAlarm. The value is given as the number of seconds since 1/1/1904.
-> soundAlarm	true if the alarm should be sounded, false otherwise. This value is currently not used.
padding	Not used.

sysAppLaunchCmdExgAskUser

Exchange manager sends this launch code to the application when data has arrived for that application. This launch code lets the application tell the exchange manager whether or not to display a dialog asking the user if they want to accept the data. If the application chooses not to handle this launch code, the default course of action is that the exchange manager displays a dialog asking the user if they want to accept the incoming data.

Prior to Palm OS release 3.5, most applications didn't need to handle this launch code, since the default action was the preferred

alternative. On Palm OS 3.5, you can have the dialog display a category pop-up list from which the user can choose a category in which to file the incoming data. To do so, you must handle `sysAppLaunchCmdExgAskUser` to call the `ExgDoDialog` function. See the description of that function for more information. If you don't handle the launch code, the exchange manager displays the dialog without the category pop-up list.

If an application responds to this launch code, it must set the `result` field in the parameter to the appropriate value. Possible values are:

`exgAskDialog`	Display the dialog without the category pop-up list (the default).
`exgAskOk`	Accept the incoming data.
`exgAskCancel`	Reject the incoming data.

For example, if your entire response to this launch code is to set the `result` field to `exgAskCancel`, your application always rejects all incoming data without displaying the dialog. If it is to set the result field to `exgAskOk`, it always accepts all incoming data without displaying the dialog.

On Palm OS 3.5 or higher if you are calling `ExgDoDialog` in your handler, return `exgAskOk` if `ExgDoDialog` was successful, or `exgAskCancel` if it failed. If you don't set the `result` field on Palm OS 3.5, the dialog is displayed twice.

If the application sets the `result` field to `exgAskOk`, or the dialog is displayed and the user presses the OK button, then the exchange manager sends the application the next launch code, `sysAppLaunchCmdExgReceiveData`, so that it can actually receive the data.

IMPORTANT: Implemented only if 3.0 New Feature Set is present.

---

sysAppLaunchCmdExgAskUser Parameter Block

Prototype

```
typedef struct {
  ExgSocketPtr    socketP;
```

```
ExgAskResultTyperesult;
UInt8           reserved;
} ExgAskParamType;
```

Fields

`<-> socketP`	Socket pointer
`<- result`	Show dialog, auto-confirm, or auto-cancel
`-> reserved`	Reserved for future use

sysAppLaunchCmdExgReceiveData

Following the launch code `sysAppLaunchCmdExgAskUser`, the exchange manager sends this launch code to the application to notify it that it should receive the data (assuming that the application and/or the user has indicated the data should be received).

The application should use exchange manager functions to receive the data and store it or do whatever it needs to with the data.

Note that the application may not be the active application, and thus may not have globals available when it is launched with this launch code. You can check if you have globals by using this code in the `PilotMain` routine:

```
Boolean appIsActive = launchFlags & sysAppLaunchFlagSubCall;
```

The `appIsActive` value will be true if your application is active and globals are available; otherwise, you won't be able to access any of your global variables during the receive operation.

The parameter block sent with this launch code is of the `ExgSocketPtr` data type. It is a pointer to the `ExgSocketType` structure corresponding to the exchange manager connection via which the data is arriving. You will need to pass this pointer to the `ExgAccept` function to begin receiving the data. For more details, refer to the "Exchange Manager" chapter.

IMPORTANT: Implemented only if 3.0 New Feature Set is present.

sysAppLaunchCmdFind

This launch command is used to implement the global find. It is sent by the system whenever the user enters a text string in a Find dialog. At that time, the system queries each application whether it handles this launch code and returns any records matching the find request.

The system sends this launch code with the `FindParamsType` parameter block to each application. The system displays the results of the query in the Find dialog.

Most applications that use text records should support this launch code. When they receive it, they should search all records for matches to the find string and return all matches.

An application can also integrate the find operation in its own user interface and send the launch code to a particular application.

Applications that support this launch code should support `sysAppLaunchCmdSaveData` and `sysAppLaunchCmdGoto` as well.

sysAppLaunchCmdFind Parameter Block

Prototype

```
typedef struct {

// These fields are used by the applications.
  UInt16            dbAccesMode;
  UInt16            recordNum;
  Boolean           more;
  Char              strAsTyped
[maxFindStrLen+1];
  Char              strToFind
[maxFindStrLen+1];
// These fields are private to the Find routine
//and should NOT be accessed by applications.
  UInt8             reserved1;
  UInt16            numMatches;
  UInt16              lineNumber;
  Boolean             continuation;
  Boolean             searchedCaller;
  LocalID             callerAppDbID;
```

```
    UInt16                  callerAppCardNo;
    LocalID                 appDbID;
    UInt16                  appCardNo;
    Boolean                 newSearch;
    UInt8                   reserved2;
    DmSearchStateType       searchState;
    FindMatchType           match [maxFinds];
  } FindParamsType;
```

Fields

`dbAccesMode`	Read mode. May be "show secret."
`recordNum`	Index of last record that contained a match.
`more`	`true` if more matches to display.
`strAsTyped [maxFindStrLen+1]`	Search string as entered.
`strToFind [maxFindStrLen+1]`	Search string in lower case.
`reserved1`	Reserved for future use.
`numMatches`	System use only.
`lineNumber`	System use only.
`continuation`	System use only.
`searchedCaller`	System use only.
`callerAppDbID`	System use only.
`callerAppCardNo`	System use only.
`appDbID`	System use only.
`appCardNo`	System use only.
`newSearch`	System use only.
`reserved2`	Reserved for future use.
`searchState`	System use only.
`match [maxFinds]`	System use only.

sysAppLaunchCmdGoto

Sent in conjunction with `sysAppLaunchCmdFind` or `sysAppLaunchCmdExgReceiveData` to allow users to actually inspect the record that the global find returned or that was received by the exchange manager.

Applications should do most of the normal launch actions, then display the requested item. The application should continue running unless explicitly closed.

An application launched with this code does have access to global variables, static local variables, and code segments other than segment 0 (in multi-segment applications).

sysAppLaunchCmdGoto Parameter Block

Prototype

```
typedef struct {
  Int16   searchStrLen;
  UInt16  dbCardNo;
  LocalID dbID;
  UInt16  recordNum;
  UInt16  matchPos;
  UInt16  matchFieldNum;
  UInt32  matchCustom;
  } GoToParamsType;
```

Fields

`searchStrLen`	Length of search string.
`dbCardNo`	Card number of the database.
`dbID`	Local ID of the database.
`recordNum`	Index of record containing a match.
`matchPos`	Position of the match.
`matchFieldNum`	Field number string was found in.
`matchCustom`	Application-specific information.

sysAppLaunchCmdGoToURL

You can send this launch code to the Clipper application to launch the application and cause it to retrieve and display the specified URL.

The parameter block for this launch command is simply a pointer to a string containing the URL.

For more information and an example of how to use this launch code, see "Using Clipper to Display Information" on page 310 in the *Palm OS Programmer's Companion*.

IMPORTANT: Implemented only if Wireless Internet Feature Set is present.

sysAppLaunchCmdInitDatabase

This launch code is sent by the Desktop Link server in response to a request to create a database. It is sent to the application whose creator ID matches that of the requested database.

The most frequent occurrence of this is when a `'data'` database is being installed or restored from the desktop. In this case, HotSync® creates a new database on the device and passes it to the application via a `sysAppLaunchCmdInitDatabase` command, so that the application can perform any required initialization. HotSync will then transfer the records from the desktop database to the device database.

When a Palm OS application crashes while a database is installed using HotSync, the reason may be that the application is not handling the `sysAppLaunchCmdInitDatabase` command properly. Be especially careful not to access global variables.

The system will create a database and pass it to the application for initialization. The application must perform any initialization required, then pass the database back to the system, unclosed.

sysAppLaunchCmdInitDatabase Parameter Block

Prototype

```
typedef struct {
  DmOpenRef dbP;
```

```
    UInt32    creator;
    UInt32    type;
    UInt16    version;
  } SysAppLaunchCmdInitDatabaseType;
```

Fields

`dbP`	Database reference.
`creator`	Database creator.
`type`	Database type.
`version`	Database version.

sysAppLaunchCmdLookup

The system or an application sends this launch command to retrieve information from another application. In contrast to Find, there is a level of indirection; for example, this launch code could be used to retrieve the phone number based on input of a name.

This functionality is currently supported by the standard Palm OS Address Book.

Applications that decide to handle this launch code must search their database for the string the user entered and perform the match operation specified in the launch code's parameter block.

If an application wants to allow its users to perform lookup in other applications, it has to send it properly, including all information necessary to perform the match. An example for this is in `Address.c` and `AppLaunchCmd.h`, which are included in your SDK.

sysAppLaunchCmdLookup Parameter Block

The parameter block is defined by the application that supports this launch code. See `AppLaunchCmd.h` for an example.

IMPORTANT: Implemented only if 2.0 New Feature Set is present.

sysAppLaunchCmdNotify

The system or an application sends this launch code to notify applications that an event has occurred. The parameter block specifies the type of event that occurred, as well as other pertinent information. To learn which notifications are broadcast by the system, see "Notification Manager Event Constants" in the "Notification Manager" chapter.

IMPORTANT: Implemented only if Notification Feature Set is present.

sysAppLaunchCmdNotify Parameter Block

The `SysNotifyParamType` structure declared in `NotifyMgr.h` defines the format of this launch code's parameter block. See its description in the "Notification Manager" chapter.

sysAppLaunchCmdOpenDB

You can send this launch code to the Clipper application to launch the application and cause it to open and display a Palm query application stored on the device. This is the same mechanism that the Launcher uses to launch query applications.

IMPORTANT: Implemented only if Wireless Internet Feature Set is present.

sysAppLaunchCmdOpenDB Parameter Block

Prototype

```
typedef struct {
  UInt16  cardNo;
  LocalID dbID;
  } SysAppLaunchCmdOpenDBType;
```

Fields

`cardNo`	Card number of database to open.
`dbID`	Database id of database to open.

sysAppLaunchCmdPanelCalledFromApp

`sysAppLaunchCmdPanelCalledFromApp` and `sysAppLaunchCmdReturnFromPanel` allow an application to let users change preferences without switching to the Preferences application. For example, for the calculator, you may launch the Formats preferences panel, set up a number format preference, then directly return to the calculator that then uses the new format.

`sysAppLaunchCmdPanelCalledFromApp` lets a preferences panel know whether it was switched to from the Preferences application or whether an application invoked it to make a change. The panel may be a preference panel owned by the application or a system preferences panel.

Examples of these system panels that may handle this launch code are:

- Network panel (called from network applications)
- Modem panel (called if modem selection is necessary)

All preferences panels must handle this launch code. If a panel is launched with this command, it should:

- Display a Done button.
- **Not** display the panel-switching pop-up trigger used for navigation within the preferences application.

IMPORTANT: Implemented only if 2.0 New Feature Set is present.

sysAppLaunchCmdReturnFromPanel

This launch code is used in conjunction with `sysAppLaunchCmdPanelCalledFromApp`. It informs an application that the user is done with a called preferences panel. The system passes this launch code to the application when a previously-called preferences panel exists.

IMPORTANT: Implemented only if 2.0 New Feature Set is present.

sysAppLaunchCmdSaveData

Instructs the application to save all current data. For example, before the system performs a Find operation, an application should save all data.

Any application that supports the Find command and that can have buffered data should support this launch code. Generally, an application only has to respond if it's the currently running application. In that case, all buffered data should be saved when the launch code is received.

sysAppLaunchCmdSaveData Parameter Block

Prototype

```
typedef struct {
  Boolean uiComing;
  UInt8   reserved1;
} SysAppLaunchCmdSaveDataType;
```

Fields

`uiComing`	`true` if system dialog is displayed before launch code arrives.
`reserved1`	Reserved for future use.

sysAppLaunchCmdSyncNotify

This launch code is sent to applications to inform them that a HotSync operation has occurred.

This launch code is sent only to applications whose databases were changed during the HotSync operation. (Installing the application database itself is considered a change.) The record database(s) must have the same creator ID as the application in order for the system to know which application to send the launch code to.

This launch code provides a good opportunity to update, initialize, or validate the application's new data, such as resorting records, setting alarms, and so on.

Because applications only receive `sysAppLaunchCmdSyncNotify` when their databases are updated, this launch code is not a good place to perform any operation that must occur after every HotSync operation. Instead, you may register to receive the `sysNotifySyncFinishEvent` on systems that have the

Notification Feature Set. This notification is sent at the end of a HotSync operation, and it is sent to all applications registered to receive it, whether the application's data changed or not. Note that there is also a `sysNotifySyncStartEvent` notification.

sysAppLaunchCmdSystemLock

Launch code sent to the system-internal security application to lock the device.

As a rule, applications don't need to do respond to this launch code. If an application replaces the system-internal security application, it must handle this launch code.

IMPORTANT: Implemented only if 2.0 New Feature Set is present.

sysAppLaunchCmdSystemReset

Launch code to respond to system soft or hard reset.

Applications can respond to this launch code by performing initialization, indexing, or other setup that they need to do when the system is reset. For more information about resetting the device, see "System Boot and Reset" in the *Palm OS Programmer's Companion*.

sysAppLaunchCmdSystemReset Parameter Block

Prototype

```
typedef struct {
  Boolean hardReset;
  Boolean createDefaultDB;
} SysAppLaunchCmdSystemResetType;
```

Fields

`hardReset` `true` if system was `hardReset`. `false` if system was `softReset`.

`createDefaultDB` If `true`, application has to create default database.

sysAppLaunchCmdTimeChange

Launch code to respond to a time change initiated by the user.

Applications that are dependent on the current time or date need to respond to this launch code. For example, an application that sets alarms may want to cancel an alarm or set a different one if the system time changes.

On systems that have the Notification Feature Set, applications should register to receive the `sysNotifyTimeChangeEvent` notification instead of responding to this launch code. The `sysAppLaunchCmdTimeChange` launch code is sent to all applications. The `sysNotifyTimeChangeEvent` notification is sent only to applications that have specifically registered to receive it, making it more efficient than `sysAppLaunchCmdTimeChange`.

sysAppLaunchCmdURLParams

This launch code is sent from the Clipper application to launch another application.

The parameter block consists of a pointer to a special URL string, which the application must know how to parse. The string is the URL used to launch the application and may contain encoded parameters. For more information, see "Launching Other Applications from Clipper" on page 311 in the *Palm OS Programmer's Companion*.

An application launched with this code may or may not have access to global variables, static local variables, and code segments other than segment 0 (in multi-segment applications). It depends on the URL that caused Clipper to send this launch code. If this launch code results from a `palm` URL, then globals are available. If the launch code results from a `palmcall` URL, then globals are not available.

The best way to test if you have global variable access is to test the `sysAppLaunchFlagNewGlobals` launch flag sent with this launch code. If this is flag is set, then you have global variable access.

IMPORTANT: Implemented only if Wireless Internet Feature Set is present.

Launch Flags

When an application is launched with any launch command, it also is passed a set of launch flags.

An application may decide not to handle the flags even if it handles the launch code itself. For applications that decide to include this launch code, the following table provides additional information:

Table 1.2 Launch Flags

Flag	Functionality
`sysAppLaunchFlagNewThread`	Creates a new thread for the application. Implies `sysAppLaunchFlagNewStack`.
`sysAppLaunchFlagNewStack`	Creates a separate stack for the application.
`sysAppLaunchFlagNewGlobals`	Creates and initializes a new globals world for the application. Implies new owner ID for memory chunks.
`sysAppLaunchFlagUIApp`	Notifies launch routine that this is a UI application being launched.
`sysAppLaunchFlagSubCall`	Notifies launch routine that the application is calling its entry point as a subroutine call. This tells the launch code that it's OK to keep the A5 (globals) pointer valid through the call. If this flag is set, it indicates that the application is already running as the current application.

Generally, the system sends launch flags along with all launch codes. Applications should just pass 0 (zero) when sending a launch code to another application.

2

Palm OS Resources

Palm OS® User Interface resources are the elements of an application's GUI (graphical user interface). This chapter provides reference material you can use when creating user interface resources in Metrowerks Constructor. It provides detailed guidelines for using each resource, and it provides descriptions of the attributes you set in Metrowerks Constructor.

NOTE: For more information see the following manuals:

The *Palm OS Tutorial* provides more detailed instruction on how to create a GUI using the Constructor tool.

The *Constructor for Palm OS* manual in the CodeWarrior Documentation folder provides detailed reference-style documentation as well as information on how to use each individual resource.

System Resources

Every application running under Palm OS must have certain minimum system (not UI) resources defined to be recognized by the Palm OS system software. These required resources are created for your application by the development environment. You may find that you need additional, application-specific resources. The required resources are 'code' #1, 'code' #0, and 'data' #0. All other system resources are optional. This section describes both the required and optional resources.

The 'code' #1 Resource

The system creates a `'code'` #1 resource for every application. This resource is the entry point for the application and is where

application initialization is performed. When the Palm OS device launches an application, it starts executing at the first byte of the 'code' #1 resource. All of the application code that you provide is included in this resource as well.

Typically, some startup code provided with the Palm OS development environment is linked in with your application code. This startup code works as follows:

- The startup code performs application setup and initialization.
- The startup code calls your `main` routine.
- When your `main` routine exits, control is returned to the startup code, which performs any necessary cleanup of your application and returns control to the Palm OS system software.

The 'code' #0 and 'data' #0 Resources

The `'code'` #0 and `'data'` #0 resources contain the required size of your global data and an image of the initialized area of that global data. When your application is launched, the system allocates a memory chunk in the dynamic heap that's big enough to hold all your globals. The `'data'` #0 resource is then used to initialize those globals.

The 'pref' #0 Resource

The system creates a`'pref'` #0 resource for every application. This resource contains startup information for launching your application. The resource includes

- Required stack size
- Dynamic heap space required (not currently used)
- Task priority (not currently used)

This resource applies only to Palm OS 3.0 and higher. It is ignored on older versions of Palm OS.

Resource Types

Metrowerks Constructor divides resources into two types: catalog resources and project resources.

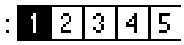
Catalog Resources

Catalog resources are available in Constructor's Catalog window and can be dragged directly on a form. Table 2.1 lists the available catalog resources. The Macintosh ResEdit resource name is included for reference only; it's not needed by developers who use Constructor exclusively, and not relevant for Windows developers.

Table 2.1 Catalog Resources

Name	Resource	Resource
tBTN	Button Resource	
tCBX	Check Box Resource	
tFLD	Field Resource	
tFBM	Form Bitmap Resource	(container for Bitmap resource)
tGDT	Gadget Resource	(application defined)
tGSI	Graffiti Shift Indicator Resource	
tLBL	Label Resource	(container for a String)

Table 2.1 Catalog Resources *(continued)*

Name	Resource	Resource
tLST	List Resource	
tPUT	Popup Trigger Resource	
tPBN	Push Button Resource	
tREP	Repeating Button Resource	
	Scroll Bar Resource	(see below)
tSLT	Selector Trigger Resource	
tTBL	Table Resource	

Project Resources

Project resources are instantiated from the projects window.

Table 2.2 lists the project resources. The Macintosh ResEdit resource name is included for reference only; it's not needed by developers who use Constructor exclusively, and not relevant for Windows developers.

Table 2.2 Project Resources

Name	Resource	UI Name
Talt	Alert Resource	Alert
tFRM	Form Resource	Form
	Menu Resource	Menu
	Menu bar Resource	Menu bar
tSTR	String Resource	String
	Icons	
	Bitmaps	

Alert Resource

Example

Overview

The alert resource defines a modal dialog that displays a message, an icon, and one or more buttons.

A small icon indicates the category of the dialog box; for example, an exclamation mark for an error message. The icon appears on the left side of the dialog. The text is justified left but placed to the right of the dialog icon.

Type	Icon	Definition	Button	Example
Information	i	Lowest-level warning. Action shouldn't or can't be completed but doesn't generate an error or risk data loss.	OK	An alarm setting must be between 1 and 99.
Confirmation	?	Confirm an action or suggest options.	OK, Cancel	Change settings before switching applications? (For example, when pressing an application key with an open dialog box.)
Warning	!	Ask if user wishes to continue a potentially dangerous action.	OK, Cancel	Are you sure you want to delete this entry?
Error	(stop sign)	Attempted action generated error and/or cannot be completed.	OK	Disk full.

The Alert resource has the following attributes.

Attributes

Alert Type — Determines the sound played and the icon displayed when the alert is drawn. There are four possible icons:

- InformationAlert (Alert Number 0)
- ConfirmationAlert (Alert Number 1)
- WarningAlert (Alert Number 2)
- ErrorAlert (Alert Number 3)

Help ID — The ID of a String resource that's the help text for the alert dialog box. If you provide a value, the system displays an "i" in the top right corner of the alert box.

Default Button ID	The number of a button that the system assumes is selected if the user switches to another application, forcing the form to go away without making a selection.
Title	Title of the alert form.
Message	Message displayed by the alert dialog. May contain ^1, ^2, ^3 as substitution variables for use in conjunction with `FrmCustomAlert`.
Button Text	Text of the button (e.g. OK or Cancel), determined by an entry in the resource of each button. To add a button, select Item Text 0, and type Cmd-K.

Button Resource

UI Structure `ControlType`

Overview A button is a clickable UI object, often used to trigger events in an application. A button displays as a text label surrounded by a rectangular frame. The frame has rounded corners. The label may be regular text or a glyph from one of the symbol fonts provided with your development environment, for example, an arrow.

Examples

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Button ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of button. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159

Width	Width of button in pixels. Size the buttons to allow 3-6 pixels of white space at each end of the label. Valid values: 0 – 160
Height	Height of the button in pixels. Should be 3 pixels larger than the font size, for example, height =12 for 9-point labels. Valid values: 1 – 160
Usable	A nonusable object is not considered part of the application's interface and doesn't draw. Nonusable object can programmatically be set to usable. If checked, the object is usable.
Anchor Left	Controls how the object resizes itself when its text label is changed. If checked, the left bound of the object is fixed; if unchecked, the right bound is fixed.
Frame	If checked, a rectangular frame with rounded corners is drawn around the button. Most buttons have frames. Buttons whose labels are single symbol characters such as scroll buttons don't have frames.
Non-bold Frame	If checked, a one-pixel-wide rectangular frame with rounded corners is drawn around the button. If unchecked, a bold frame (two pixels wide) is drawn around the button. Non-bold frames are the default.
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed inside the button: one line of text or a single character from a symbol font to create an increment arrow.

Comments

The label is centered in the button. If the label text is wider than the button, the whole label is centered and both the right and left sides are clipped.

Place command buttons at the bottom of table views and dialog boxes. Leave three pixels between the dialog bottom and buttons.

Increment arrows are a special case; they are buttons that let users increment the value displayed in a data field.

To create an increment arrow, use an arrow character from the Symbol font as a label. Several arrow styles and sizes are available.

Tip **Making a Button with a Bitmap Label**

It's not possible to make a bitmap the label of a button; the label always has to be a text string. However, the same effect can be achieved by

- Creating a bitmap the same size of a button
- Placing them at the same location.

Make sure the bitmap is a Form Bitmap, selected from the catalog.

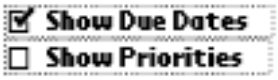
When the user selects the button, the system inverts the bitmap graphic as well.

Check Box Resource

UI Structure `ControlType`

Overview A check box is a small, square UI object with an optional text label to the right.

Example The figure below shows a checked and an unchecked check box with a label to the right (the default).



Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Check Box ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159

Width	Width of the picking area around the check box. Valid values: 0 – 160
Height	Height of the picking area around the check box. Valid values: 1– 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Selected	Initial selection state of the checkbox. If the box is checked (the default), the checkbox is initially checked.
Group ID	Group ID of a check box that is part of an exclusive group. Ungrouped (nonexclusive) check boxes have 0 as a group ID. Valid values: 0 – 65535
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed to the right of the check box. This text is part of the activation area. To create a (nonactive) label to the left of the check box, leave this attribute blank and create a separate Label resource.

Comments Make sure that only one check box in a group is initially checked.

All check boxes are the same size. The Height and Width determine the toggle area, which is the screen area the user needs to press to check or uncheck the box.

If a label attribute is defined, it's part of the activation area.

Field Resource

UI Structure `FieldType`

Overview

The field UI object is for user data entry in an application. It displays one or more lines of text. A field can be underlined, justified left or right, and editable or uneditable.

Text fields can be located anywhere except in menus and in the command button area.

The following is an underlined, left-justified field containing data:

Look Up: Text

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Field ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159
Width	Width of the object in pixels. Valid values: 0 – 160
Height	Height of the object in pixels. Valid values: 1– 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Editable	If this box is checked, the field is editable. Noneditable fields don't accept user input but can be changed programmatically. Noneditable text fields are useful when you want to display text on a form but don't want users to edit it.
Underline	If set, each line of text is underlined with a gray line.

Single Line	If checked, the field doesn't scroll horizontally and doesn't accept Return or Tab characters. Only a single line of text is displayed. If the user attempts to enter text beyond this, the system beeps. Multiline text fields expand. An empty field may display one or more blank lines; for example, records in a To Do list or a text page.
Dynamic Size	If checked, the height of the field is expanded or compressed as characters are added or removed. Set this attribute to false if the Single Line attribute is set.
Left Justified	Text justification. Supported only for fields that have the Single Line attribute checked. Valid values: checked (left-justified)—recommended unchecked (right-justified)
Max characters	Maximum number of characters the field accepts. This is a limit on the number of characters a user can enter, but not on what can be displayed. All fields can display up to 32,767 characters regardless of this setting. Valid values: 0 – 32767
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Auto-Shift	If checked, 2.0 (and later) auto-shift rules are applied.
Has Scrollbar	If checked, the field has a scroll bar. The system sends more frequent `fldHeightChangedEvent`s so the application can adjust the height appropriately.
Numeric	If checked, only the characters 0 through 9 are allowed to be entered in the field.

Form Resource

Overview A form is a container for one or more of the Catalog Resources.

Applications usually contain several different forms that the user triggers by tapping buttons or other control UI objects. Most UI objects are displayed only if they are contained within a form.

Example

The example below shows a modal form. A form can also be as large as the screen.

Attributes

Left Origin	Window-relative position of left side of form. Valid values: 0 – 159
Top Origin	Window-relative position of top of form. Valid values: 0 – 159
Width	Width of the form in pixels. Valid values: 0 – 160
Height	Height of the form in pixels. Valid values: 1– 160
Usable	Not currently supported for forms.
Modal	If checked, form is modal. Modal forms ignore pen events outside their boundaries. Used for dialogs.
Save Behind	If checked, the region obscured by the form is saved when it's drawn and restored when it's erased. Used for dialogs.
Form ID	Form ID assigned by Constructor.
Help ID	ID number of a string that's displayed when the user taps the "i" icon. The system adds the icon to the form when you provide a value for this property. Currently, only modal dialogs have help resources.

Menu Bar ID	Contains the ID of a menu bar resource to be associated with this form.
Default Button ID	ID number of a button that the system assumes is selected if the user switches to another application, forcing the form to go away without making a selection.
Form Title	Title of that form. Use titles for dialogs, menu bars for views. By convention, the name of the application and the name of the screen, if possible, for example Address List or Address Edit. The title must be one line; it uses about 13 pixels of the top of the form.
Palm OS Version	Version of the device for which this form is created.

Comments

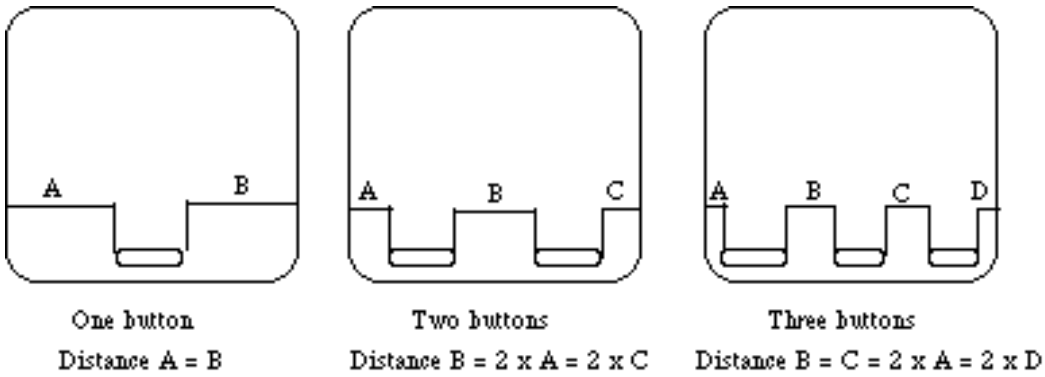
The total display on the Palm device is 160 pixels by 160 pixels. If you want your whole form to be seen, make sure it fits within this display area. For pop-up dialogs, you can make the form smaller. Align a popup dialog with the bottom of the screen.

Here are some general design guidelines:

- Each form should have a title that displays the name or view of the application, or both.
- Scroll bars in fields and tables appear and disappear dynamically if you've selected that option for that UI element. Place them to the right of command buttons.
- Modal dialogs always occupy the full width of the screen and are justified to the bottom of the screen. They hide the command buttons of the base application but don't obscure the title bar of the base application if possible. There should be a minimum of three pixels between the top of the modal dialog title bar and the bottom of the application title bar. If the dialog is too large to accommodate this, the entire application title bar should be obscured.
- Screen command buttons should always be at the bottom of the screen.

- Dialog command buttons appear four pixels above the bottom of the dialog box frame. Two-pixel default ring is three pixels above the bottom, and the baseline of the text within the buttons should be aligned.
- Command buttons should be centered so that the spaces between the buttons are twice the width of the spaces between the edges and the border (see diagram below).

  If possible, all buttons should be the same width. At a minimum, they should be spaced equidistant, as illustrated below.



Form Bitmap Resource

Overview Places predefined bitmaps on a given form. Used for icons in Alert dialogs to indicate a warning, error, information, and so on. You have to associate a Bitmap with the Form Bitmap to actually make a picture appear.

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Left Origin.	Left bounds of bitmap.
Top Origin	Top bounds of bitmap.

Bitmap Resource ID	ID of a PICT resource containing the graphic. You can also assign an ID number, then click on Create and draw the picture in the bitmap editor that appears.
Usable	Checked if the bitmap should be drawn.

Gadget Resource

Name tGDT

UI Name Gadget

Overview A gadget object lets developers implement a custom UI gadget. The gadget resource contains basic information about the custom gadget, which is useful to the gadget writer for drawing and processing user input.

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Gadget ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159
Width	Width of the gadget in pixels. Valid values: 0 – 160

Height	Height of the gadget in pixels. Valid values: 1– 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.

Graffiti Shift Indicator Resource

Overview

The Graffiti® Shift Indicator resource specifies the window-relative or form-relative position of the Graffiti shift state indicator. The different shift states are punctuation, symbol, uppercase shift, and uppercase lock. These indicators will appear at the position of the Graffiti Shift resource.

Note: By convention, Graffiti Shift indicators are placed at the bottom-right of every form that has an editable text field.

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159
Object ID	ID of the object (assigned by Constructor).

Label Resource

Overview

The label resource displays noneditable text or labels on a form (dialog box or full-screen). It's used, for example, to have text appear to the left of a checkbox instead of the right.

Comments Pressing Return in a label wraps the text to the next line.

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Label ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Text	Text of the label.

List Resource

UI Structure `ListType`

Example

Overview A list provides a box with a list of choices to the user. The list is scrollable if the choices don't all fit in the box.

The list box appears as a vertical list of choices surrounded by a rectangular frame. The current selection of the list is inverted. Arrows for scrolling the list appear in the right margin if necessary.

Lists can appear as popup lists when used with popup triggers. See Popup Trigger Resource.

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
List ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159
Width	Width of the list. Valid values: 0 – 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Visible items	Height of the list box, in items (choices). For example, if the list has six items but only four fit, specify four.
Items	Items in the list.

Comments

Errors may occur if the number of visible items is greater than the actual number of items. An item's text is not clipped against the list box's borders. Set a list to not usable if it's linked to a popup trigger.

Use a list to let users choose between items of data; use a menu to activate a command.

If a list becomes too tall to fit below the trigger, it's justified up. If it becomes to large for the screen, it scrolls.

Menus and Menu Bars

Overview

A menu assembly consists of a menu bar, menu names indicating the available menus, and the menus themselves with their commands:

- **Menu bar**. The menu bar at the top of the screen contains the names of the available menus. Each application has different sets of menu names; within an application, different views may have different menus.
- **Menu name**. Each menu is displayed below the menu name. The following menu names are commonly found:
  - Record—Place Record to the left of Edit (if applicable).
  - Edit—Screens that allow editing need an Edit menu. Note, however, that most editing is edit-in-place.
  - Options—Typically, the last menu. The About command, which provides version and creator information, should always be an Options command under Palm OS.
- **Menu**. The menus themselves consist of menu items and optional shortcuts. Under Palm OS, menu items should **not** duplicate functionality available via command buttons. Menus justify left with the active heading of the menu name when invoked. If the menu doesn't fit, it's justified to the right border of the screen.

NOTE: For each menu, provide shortcuts for all commands or for none at all. Don't assign the same shortcut twice within one application.

Menu Bar and Menu Resources

The only information provided for the menu and menu bar resource is the resource name and resource ID.

Menu User Interaction

- **Default Menu and Menu Item.** A pen-up on the menu icon displays the menu bar. The first time a menu is invoked after an application is launched, no menus are displayed unless there is only one menu available. Afterwards the menu and menu item of the last command executed from the menu are displayed. Graffiti command equivalents are ignored.

  For example, if the user selects Edit > Copy, the Edit menu is popped down and the Copy command is highlighted the next time the menu bar is displayed. This expedites execution of commonly used commands or of grouped commands (e.g., Copy/Paste). The last menu heading is not saved if the user switches to a different view or a different application.

- **View-specific Menus.** Each view within an application can have a unique menu, that is, different menu headings and items.

- **Menu Display.** As a rule, a Palm OS application should try to have the menu visible on screen as rarely as possible:
  - After a menu command is executed, the menu bar is dismissed.
  - The menu bar is active when the menu headings in it are active. When not active, the menu bar is not visible.
  - There are no grayed-out menu headings or grayed-out menu items. A command not accessible in a certain mode doesn't appear at all.

- **Size.** The vertical active area of menu headings is 2 pixels beyond the ascender and 1 pixel below a potential descender of the menu heading text. The horizontal active area covers half the distance to the next menu heading, leaving no gaps between the headings. If the menu headings aren't as wide as the menu bar, part of the menu bar may be inactive.

- **Active Area.** The entire area of the menu, excluding the border, is active. Divider lines and status items on the

launcher menu are inactive; that is, they do not highlight when tapped.

Popup Trigger Resource

UI structure `ControlType`

Overview The popup trigger shows the selection of a list. The user can press the popup trigger to pop up the list and change the selection.

A popup trigger displays a text label and a triangle to the left of the label that indicates the object is a popup trigger.

When the user selects a popup trigger, a list of items pops up.

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Popup ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of button. Valid values: 0 – 159
Top Origin	Form-relative position of top of button. Valid values: 0 – 159
Width	Width of the button's picking area in pixels. Valid values: 1 – 160
Height	Height of the button's picking area in pixels. Valid values: 1 – 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.

Left anchor	Controls how the object resizes itself when its text label is changed. Valid values: • checked (left bound fixed) • unchecked (right bound fixed)
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed in the popup trigger (right of the arrow).
List ID	ID of the List object that pops up when the user taps the pop-up trigger.

Push Button Resource

UI Structure `ControlType`

Overview Push buttons allow users to select an option from a group of items. The choices should have few characters; if the choices are long; check boxes are preferable.

Push buttons display a text label surrounded by a 1-pixel-wide rectangular frame. They appear in a horizontal or vertical row with no pixels between the buttons. The buttons share a common border so there appears to be a one pixel line between two controls. The current selection is highlighted.

The List By dialog of the Address Book and the Details dialog of the ToDo List contain examples of rows of push buttons.

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Button ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of button. Valid values: 0 – 159
Top Origin	Form-relative position of top of button. Valid values: 0 – 159
Width	Width of the button in pixels. Should be size of label plus two pixels at each end. Valid values: 1 – 160
Height	Height of the button in pixels. Should be font size plus two pixels. Valid values: 1 – 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Group ID	Group ID of a push button that is part of an exclusive group. Only one push button in an exclusive group may be depressed at a time. Ungrouped (nonexclusive) push buttons have zero as a group ID. This feature must be enforced by the application. Valid values: 0 – 65535
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed inside the push button.

Comment To create a row of push buttons, create a number of individual push button resources with the same height and align them by specifying the same top position for each button.

Repeating Button Resource

Overview

The repeating button object is identical to the button object in its appearance. The repeating button is used for buttons that need to be triggered continuously by holding the pen down on them.

A good example for a repeating button is the scroll arrow, which moves text as long as it's held down.

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Button ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of button. Valid values: 0 – 159
Top Origin	Form-relative position of top of button. Valid values: 0 – 159
Width	Width of the button in pixels. Valid values: 1 – 160
Height	Height of the button in pixels. Valid values: 1 – 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Anchor Left	Controls how the object resizes itself when its text label is changed. If checked, the left bound of the object is fixed; if unchecked, the right bound is fixed.
Frame	If checked, a rectangular frame with rounded corners is drawn around the button.

Non-bold Frame	Determines the width of the rectangular frame drawn around the object. Valid values: • checked (1-pixel-wide frame) • unchecked (2-pixel-wide frame)
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed inside the button.

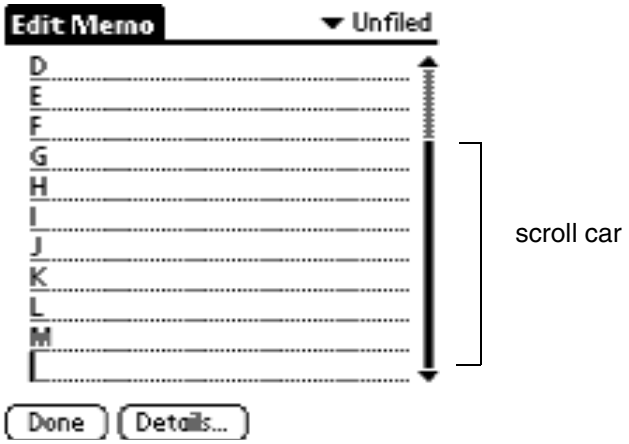
Comments The attributes match those of the Button Resource (tBTN); the behavior differs.

You can also use repeating buttons to create increment arrows. See Button Resource for more information.

Scroll Bar Resource

Overview The scroll bar resource helps developers to provide scrolling behavior for fields and tables.

Example



Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Scrollbar ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of left side of the scroll bar. Valid values: 0 – 159
Top Origin	Form-relative position of top of the scroll bar. Valid values: 0 – 159
Width	Width of the scroll bar in pixels. 7 (the default) is strongly recommended.
Height	Height of the scroll bar in pixels. Valid values:1 – 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Value	Current top value of the scroll bar's car (movable piece).
Min Value	Position of the scroll car when the scroll bar is at the top. Default should usually be 0.
Max Value	Position of the scroll car when the scroll bar is at the bottom. To compute this value, use the formula: Number of lines – Page size + Overlap.
Page Size	Number of lines to scroll at one time.

Selector Trigger Resource

UI Structure `ControlType`

Overview Users can tap a selector trigger to pop up a dialog that lets them select an item. The selected item becomes the label of the selector

trigger. For example, a selector trigger for time pops up a time selector. The selected time is entered into the selector trigger.

A selector trigger displays a text label surrounded by a gray rectangular frame, as shown below:

Attributes

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Selector Trigger ID	ID of the object (assigned by Constructor).
Left Origin	Form-relative position of the left side of the object. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159
Width	Width of the object in pixels. Valid values: 1– 160
Height	Height of the object in pixels. Height extends two pixels above and one pixel below the 9-point plain font. Height is one pixel above command buttons to accommodate the gray frame. Valid values: 1– 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Anchor Left	Controls how the object resizes itself when its text label is changed. If checked, the left bound of the object is fixed, if unchecked, the right bound is fixed. Valid values: • checked (left bound fixed) • unchecked (right bound fixed)

Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text in the selector trigger.

String Resource

Name Strings

Overview Stores data strings used by the program. String resources may be entered as text strings or as a series of hexadecimal characters.

Attributes

String	The text string to be stored, in decimal ASCII.

Comments The string resource uses either the string or data. If both are entered, they are concatenated.

Table Resource

Overview The table object allows the developer to organize a collection of objects on the display. For example, a table might contain a column of labels that correspond to a column of fields. Under some circumstances, a one-column table may be preferable to a list.

Comments Since tables are scrollable, they may be larger than the display.

Example

Attributes

Object Identifier	Name of the object. Assigned by developer, used by Constructor during header file generation/update.
Table ID	ID of the object (assigned by Constructor).

Left Origin	Form-relative position of left side of the object. Valid values: 0 – 159
Top Origin	Form-relative position of top of object. Valid values: 0 – 159
Width	Width of the object in pixels. Valid values: 1– 160
Height	Height of the object in pixels. Valid values: 1–160
Editable	If the user can modify the table.
Rows	Number of rows in the table.
Columns	Number of columns in the table.
Column width	Width of the nth column.

3

Palm OS Events

Palm OS® events are structures (defined in the header files `Event.h`, `SysEvent.h`, and `INetMgr.h`) that the system passes to the application when the user interacts with the graphical user interface. Chapter 4, "Event Loop" on page 65 in the *Palm OS Programmer's Companion* discusses in detail how this works. This chapter provides reference-style information about each event. First it shows the types used by Palm OS events. Then it discusses the following events in alphabetical order:

Event	UI Object
`appStopEvent`	N.A.
`ctlEnterEvent`, `ctlExitEvent`, `ctlRepeatEvent`, `ctlSelectEvent`	Control
`daySelectEvent`	N.A.
`fldChangedEvent`, `fldEnterEvent`, `fldHeightChangedEvent`	Field
`frmCloseEvent`, `frmGotoEvent`, `frmLoadEvent`, `frmOpenEvent`, `frmSaveEvent`, `frmUpdateEvent`, `frmTitleEnterEvent`, `frmTitleSelectEvent`	Form
`frmGadgetEnterEvent`, `frmGadgetMiscEvent`	Extended gadget
`inetSockReadyEvent`, `inetSockStatusChangeEvent`	N.A. (INetLib)
`keyDownEvent`	N.A.
`lstEnterEvent`, `lstExitEvent`, `lstSelectEvent`	List
`menuEvent`, `menuOpenEvent`, `menuCloseEvent`, `menuCmdBarOpenEvent`	Menu
`nilEvent`	N.A.
`penDownEvent`, `penMoveEvent`, `penUpEvent`	N.A. (pen)

Event	UI Object
popSelectEvent	Popup (Control)
sclEnterEvent, sclRepeatEvent, sclExitEvent	Scroll bar
tblEnterEvent, tblExitEvent, tblSelectEvent	Table
winEnterEvent, winExitEvent	Window

Event Data Structures

eventsEnum

The `eventsEnum` enum specifies the possible event types.

```
enum events {
  nilEvent = 0,
  penDownEvent,
  penUpEvent,
  penMoveEvent,
  keyDownEvent,
  winEnterEvent,
  winExitEvent,
  ctlEnterEvent,
  ctlExitEvent,
  ctlSelectEvent,
  ctlRepeatEvent,
  lstEnterEvent,
  lstSelectEvent,
  lstExitEvent,
  popSelectEvent,
  fldEnterEvent,
  fldHeightChangedEvent,
  fldChangedEvent,
  tblEnterEvent,
  tblSelectEvent,
  daySelectEvent,
  menuEvent,
  appStopEvent = 22,
  frmLoadEvent,
```

```
    frmOpenEvent,
    frmGotoEvent,
    frmUpdateEvent,
    frmSaveEvent,
    frmCloseEvent,
    frmTitleEnterEvent,
    frmTitleSelectEvent,
    tblExitEvent,
    sclEnterEvent,
    sclExitEvent,
    sclRepeatEvent,
    tsmFepModeEvent,

    menuCmdBarOpenEvent = 0x0800,
    menuOpenEvent,
    menuCloseEvent,
    frmGadgetEnterEvent,
    frmGadgetMiscEvent,

    firstINetLibEvent = 0x1000,
    firstWebLibEvent = 0x1100,

    firstUserEvent = 0x6000
  } eventsEnum;
```

Each of these event types is discussed in alphabetical order below.

EventType

The `EventType` structure contains all the data associated with a system event. All event types have some common data. Most events also have data specific to those events. The specific data uses a union that is part of the `EventType` data structure. The union can have up to 8 words of specific data.

The common data is documented below the structure. The Event Reference section gives details on the important data associated with each type of event.

```
  typedef struct {
    eventsEnum  eType;
    Boolean     penDown;
```

```
    UInt8       tapCount;
    Int16       screenX;
    Int16       screenY;
    union{
      ...
    } data;
  } EventType;
```

Common Field Descriptions

`eType` One of the `eventsEnum` constants. Specifies the type of the event.

`penDown` `true` if the pen was down at the time of the event, otherwise `false`.

`tapCount` The number of taps received at this location. This value is used mainly by fields. When the user taps in a text field, two taps selects a word, and three taps selects the entire line.

`screenX` Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

`screenY` Window-relative position of the pen in pixels (number of pixels from the top left of the window).

`data` The specific data for an event, if any. The data is a union, and its exact contents depend on the `eType` field. The Event Reference section in this chapter shows what the `data` field contains for each event.

NOTE: Remember that the `data` field is part of the access path to an identifier in the `EventType` structure. As an example, the code to access the `controlID` field of a `ctlEnterEvent` would be:

```
EventType *event;
//...
if (event->data.ctlEnter.controlID ==
            MyAppLockButton)
```

Compatibility

The `tapCount` field is only defined if 3.5 New Feature Set is present. Because of the `tapCount` field, it's particularly important that you clear the event structure before you use it to add a new event to the queue in Palm OS 3.5 and higher. Otherwise, the `tapCount` value may be incorrect for the new event.

EventPtr

The `EventPtr` defines a pointer to an `EventType`.

```
typedef EventType *EventPtr;
```

Event Reference

appStopEvent

When the system wants to launch a different application than the one currently running, the event manager sends this event to request the current application to terminate. In response, an application has to exit its event loop, close any open files and forms, and exit.

If an application doesn't respond to this event by exiting, the system can't start the other application.

ctlEnterEvent

The control routine `CtlHandleEvent` sends this event when it receives a `penDownEvent` within the bounds of a control.

For this event, the `data` field contains the following structure:

```
struct ctlEnter {
  UInt16 controlID;
  struct ControlType *pControl;
} ctlEnter;
```

Field Descriptions

`controlID` Developer-defined ID of the control.

`pControl` Pointer to a control structure (`ControlType`).

ctlExitEvent

The control routine `CtlHandleEvent` sends this event. When `CtlHandleEvent` receives a `ctlEnterEvent`, it tracks the pen until the pen is lifted from the display. If the pen is lifted within the bounds of a control, a `ctlSelectEvent` is added to the event queue; if not, a `ctlExitEvent` is added to the event queue.

The following data is passed with the event:

Field Descriptions

`penDown` `true` if the pen was down at the time of the event, otherwise `false`.

`screenX` Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

`screenY` Window-relative position of the pen in pixels (number of pixels from the top left of the window).

ctlRepeatEvent

The control routine `CtlHandleEvent` sends this event. When `CtlHandleEvent` receives a `ctlEnterEvent` in a repeating button (tREP) or a feedback slider control (tslf), it sends a `ctlRepeatEvent`. When `CtlHandleEvent` receives a `ctlRepeatEvent` in a repeating button, it sends another `ctlRepeatEvent` if the pen remains down within the bounds of the control for 1/2 second beyond the last `ctlRepeatEvent`.

When `CtlHandleEvent` receives a `ctlRepeatEvent` in a feedback slider control, it sends a `ctlRepeatEvent` each time the slider's thumb moves by at least one pixel. Feedback sliders do not send `ctlRepeatEvents` at regular intervals like repeating buttons do.

If you return `true` in response to a `ctlRepeatEvent`, it stops the `ctlRepeatEvent` loop. No further `ctlRepeatEvents` are sent.

For this event, the `data` field contains the following structure:

```
struct ctlRepeat {
  UInt16 controlID;
  struct ControlType *pControl;
  UInt32 time;
  UInt16 value;
} ctlRepeat;
```

Field Descriptions

`controlID`	Developer-defined ID of the control.
`pControl`	Pointer to a control structure (`ControlType`).
`time`	System-ticks count when the event is added to the queue.
`value`	Current value if the control is a feedback slider.

Compatibility

The `value` field is only present if 3.5 New Feature Set is present.

ctlSelectEvent

The control routine `CtlHandleEvent` sends this event. When `CtlHandleEvent` receives a `ctlEnterEvent`, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of the same control it went down in, a `cltSelectEvent` is added to the event queue; if not, a `ctlExitEvent` is added to the event queue.

For this event, the `data` field contains the following structure:

```
struct ctlSelect {
  UInt16  controlID;
  struct ControlType *pControl;
  Boolean on;
  UInt8   reserved1;
  UInt16  value;
} ctlSelect;
```

Field Descriptions

`controlID`	Developer-defined ID of the control.
`pControl`	Pointer to a control structure (`ControlType`).
`on`	`true` when the control is depressed; otherwise, `false`.
`reserved1`	Unused.
`value`	Current value if the control is a slider.

Compatibility

The `value` field is only present if 3.5 New Feature Set is present.

daySelectEvent

The system-internal `DayHandleEvent` routine, which handles events in the day selector object, handles this event. When the day selector object displays a calendar month, the user can select a day by tapping on it.

This event is sent when the pen touches and is lifted from a day number.

For this event, the `data` field contains the following structure:

```
struct daySelect {
  struct DaySelectorType *pSelector;
  Int16 selection;
  Boolean useThisDate;
  UInt8 reserved1;
} daySelect;
```

Field Descriptions

`pSelector`	Pointer to a day selector structure (`DaySelectorType`).
`selection`	Not used.
`useThisDate`	Set to `true` to automatically use the selected date.
`reserved1`	Unused.

fldChangedEvent

The field routine FldHandleEvent sends this event when the text of a field has been scrolled as a result of drag-selecting. When FldHandleEvent receives a fldEnterEvent, it positions the insertion point and tracks the pen until it's lifted. Text is selected (highlighted) appropriately as the pen is dragged.

For this event, the data field contains the following structure:

```
struct fldChanged {
  UInt16 fieldID;
  struct FieldType *pField;
} fldChanged;
```

Field Descriptions

fieldID Developer-defined ID of the field.

pField Pointer to a field structure (FieldType).

fldEnterEvent

The field routine FldHandleEvent sends this event when the field receives a penDownEvent within the bounds of a field. For this event, the data field contains the following structure:

```
struct fldEnter {
  UInt16 fieldID;
  struct FieldType *pField;
} fldEnter;
```

Field Descriptions

fieldID Developer-defined ID of the field.

pField Pointer to a field structure (FieldType).

fldHeightChangedEvent

The field routine FldHandleEvent sends this event. The field API supports a feature that allows a field to dynamically resize its visible height as text is added or removed from it. Functions in the field API send a fldHeightChangedEvent to change the height of a field.

If the field is contained in a table, the table's code handles the `fldHeightChangedEvent`. If the field is directly on a form, your application code should handle the `fldHeightChangedEvent` itself. The form code does not handle the event for you.

For this event, the `data` field contains the following structure:

```
struct fldHeightChanged {
  UInt16  fieldID;
  struct FieldType *pField;
  Int16   newHeight;
  UInt16  currentPos;
} fldHeightChanged;
```

Field Descriptions

`fieldID`	Developer-defined ID of the field.
`pField`	Pointer to a field structure (`FieldType`).
`newHeight`	New visible height of the field, in number of lines.
`currentPos`	Current position of the insertion point.

frmCloseEvent

The form routines `FrmGotoForm` and `FrmCloseAllForms` send this event. `FrmGotoForm` sends a `frmCloseEvent` to the currently active form; `FrmCloseAllForms` sends a `frmCloseEvent` to all forms an application has loaded into memory. If an application doesn't intercept this event, the routine `FrmHandleEvent` erases the specified form and releases any memory allocated for it.

For this event, the `data` field contains the following structure:

```
struct frmClose {
  UInt16 formID;
} frmClose;
```

Field Descriptions

`formID`	Developer-defined ID of the form.

frmGadgetEnterEvent

The function FrmHandleEvent sends this event when there is a penDownEvent within the bounds of an extended gadget. The gadget handler function (see FormGadgetHandler) should handle this event.

For this event, the data field contains the following structure:

```
struct gadgetEnter {
  UInt16 gadgetID;
  struct FormGadgetType *gadgetP;
} gadgetEnter;
```

Field Descriptions

gadgetID	Developer-defined ID of the gadget.
gadgetP	Pointer to the FormGadgetType object representing this gadget.

Compatibility

Implemented only if 3.5 New Feature Set is present.

frmGadgetMiscEvent

An application may choose to send this event when it needs to pass information to an extended gadget. The FrmHandleEvent function passes frmGadgetMiscEvents on to the extended gadget's handler function (see FormGadgetHandler).

For this event, the data field contains the following structure:

```
struct gadgetMisc {
  UInt16 gadgetID;
  struct FormGadgetType *gadgetP;
  UInt16 selector;
  void *dataP;
} gadgetMisc;
```

Field Descriptions

`gadgetID`	Developer-defined ID of the gadget.
`gadgetP`	Pointer to the `FormGadgetType` object representing this gadget.
`selector`	Any necessary integer value to pass to the gadget handler function.
`dataP`	A pointer to any necessary data to pass to the gadget handler function.

Compatibility

Implemented only if 3.5 New Feature Set is present.

frmGotoEvent

An application may choose to send itself this event when it receives a `sysAppLaunchCmdGoto` launch code. `sysAppLaunchCmdGoto` is generated when the user selects a record in the global find facility. Like `frmOpenEvent`, `frmGotoEvent` is a request that the application initialize and draw a form, but this event provides extra information so that the application may display and highlight the matching string in the form.

The application is responsible for handling this event.

For this event, the `data` field contains the following structure:

```
struct frmGoto {
  UInt16 formID;
  UInt16 recordNum;
  UInt16 matchPos;
  UInt16 matchLen;
  UInt16 matchFieldNum;
  UInt32 matchCustom;
} frmGoto;
```

Field Descriptions

`formID`	Developer-defined ID of the form.
`recordNum`	Index of record containing the match string.

`matchPos`	Position of the match.
`matchLen`	Length of the matched string.
`matchFieldNum`	Number of the field the matched string was found in.
`matchCustom`	Application-specific information. You might use this if you need to provide extra information to locate the matching string within the record.

frmLoadEvent

The form routines `FrmGotoForm` and `FrmPopupForm` send this event. It's a request that the application load a form into memory.

The application is responsible for handling this event.

For this event, the `data` field contains the following structure:

```
struct frmLoad {
  UInt16 formID;
} frmLoad;
```

Field Descriptions

`formID`	Developer-defined ID of the form.

frmOpenEvent

The form routines `FrmGotoForm` and `FrmPopupForm` send this event. It is a request that the application initialize and draw a form.

The application is responsible for handling this event.

For this event, the `data` field contains the following structure:

```
struct frmOpen {
  UInt16 formID;
} frmOpen;
```

Field Descriptions

`formID`	Developer-defined ID of the form.

frmSaveEvent

The form routine FrmSaveAllForms sends this event. It is a request that the application save any data stored in a form.

The application is responsible for handling this event.

No data is passed with this event.

frmTitleEnterEvent

The control routine FrmHandleEvent sends this event when it receives a penDownEvent within the bounds of the title of the form. Note that only the written title, not the whole title bar is active.

For this event, the `data` field contains the following structure:

```
struct frmTitleEnter {
  UInt16 formID;
  } frmTitleEnter;
```

Field Descriptions

`formID`	Developer-defined ID of the form.

frmTitleSelectEvent

The control routine FrmHandleEvent sends this event. FrmHandleEvent receives a frmTitleEnterEvent, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of the active same title bar region, a `frmTitleSelectEvent` is added to the event queue.

For this event, the `data` field contains the following structure:

```
struct frmTitleSelect {
  UInt16 formID;
} frmTitleSelect;
```

Field Descriptions

`formID`	Developer-defined ID of the form.

Compatibility

In Palm OS version 3.5 and higher, FrmHandleEvent responds to frmTitleSelectEvent. Its response is to enqueue a keyDownEvent with a vchrMenu character to display the form's menu.

frmUpdateEvent

The form routine FrmUpdateForm, or in some cases the routine FrmEraseForm, sends this event when it needs to redraw the region obscured by the form being erased.

Generally, the region obscured by a form is saved and restored by the form routines without application intervention. However, in cases where the system is running low on memory, the form's routine may not save obscured regions itself. In that case, the application adds a frmUpdateEvent to the event queue. The form receives the event and redraws the region using the updateCode value.

An application can define its own updateCode and then use this event to also trigger behavior in another form, usually when changes made to one form need to be reflected in another form.

For this event, the data field contains the following structure:

```
struct frmUpdate {
  UInt16 formID;
  UInt16 updateCode;
} frmUpdate;
```

Field Descriptions

formID	Developer-defined ID of the form.
updateCode	The reason for the update request. FrmEraseForm sets this code to frmRedrawUpdateCode, which indicates that the entire form needs to be redrawn. Application developers can define their own updateCode. The updateCode is passed as a parameter to FrmUpdateForm.

inetSockReadyEvent

This event is returned only by `INetLibGetEvent` (not `EvtGetEvent`) when the Internet library determines that a socket has data ready for an `INetLibSockRead`.

For this event, the `data` field contains the following structure:

```
struct  {
  MemHandle sockH;
  UInt32    context;
  Boolean   inputReady;
  Boolean   outputReady;
} inetSockReady;
```

Field Descriptions

`sockH`	Socket handle of the socket that this event refers to.
`context`	Not used.
`inputReady`	`true` when the socket has data ready for the `INetLibSockRead` call.
`outputReady`	Not used.

The `penDown`, `tapCount`, `screenX` and `screenY` fields are **not** valid for Internet library events and should be ignored.

Compatibility

Implemented only if Wireless Internet Feature Set is present.

inetSockStatusChangeEvent

This event is returned only by `INetLibGetEvent` (not `EvtGetEvent`) when the Internet library determines that a socket has data ready for an `INetLibSockRead`.

For this event, the `data` field contains the following structure:

```
struct  {
  MemHandle sockH;
  UInt32    context;
  UInt16    status;
```

```
    Err        sockErr;
  }inetSockStatusChange;
```

Field Descriptions

`sockH`	Socket handle of the socket that this event refers to.
`context`	Not used.
`status`	Current status of the socket. This is one of the `INetStatusEnum` constants.
`sockErr`	Reason for failure of the last operation, if any. The current socket error can be cleared by calling `INetLibSockStatus`.

The `penDown`, `tapCount`, `screenX` and `screenY` fields are **not** valid for Internet library events and should be ignored.

Compatibility

Implemented only if Wireless Internet Feature Set is present.

keyDownEvent

This event is sent by the system when the user enters a Graffiti® character, presses one of the buttons below the display, or taps one of the icons in the icon area; for example, the Find icon.

For this event, the `data` field contains the following structure:

```
  struct _KeyDownEventType {
    WChar    chr;
    UInt16   keyCode;
    UInt16   modifiers;
  };
```

Field Descriptions

`chr`	The character code.
`keyCode`	Unused.
`modifiers`	0, or one or more of the following values:

`shiftKeyMask`	Graffiti is in case-shift mode.
`capsLockMask`	Graffiti is in cap-shift mode.
`numLockMask`	Graffiti is in numeric-shift mode.
`commandKeyMask`	The Graffiti glyph was the menu command glyph or a virtual key code.
`optionKeyMask`	Not implemented. Reserved.
`controlKeyMask`	Not implemented. Reserved.
`autoRepeatKeyMask`	Event was generated due to auto-repeat.
`doubleTapKeyMask`	Not implemented. Reserved.
`poweredOnKeyMask`	The key press caused the system to be powered on.
`appEvtHookKeyMask`	System use only.
`libEvtHookKeyMask`	System use only.

lstEnterEvent

The list routine `LstHandleEvent` sends this event when it receives a `penDownEvent` within the bounds of a list object.

For this event, the `data` field contains the following structure:

```
struct lstEnter {
  UInt16 listID;
  struct ListType *pList;
  Int16 selection;
} lstEnter;
```

Field Descriptions

`listID`	Developer-defined ID of the list.
`pList`	Pointer to a list structure (`ListType`).
`selection`	Unused.

lstExitEvent

The list routine LstHandleEvent sends this event. When LstHandleEvent receives a lstEnterEvent, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of a list, a lstSelectEvent is added to the event queue; if not, a lstExitEvent is added to the event queue.

For this event, the data field contains the following structure:

```
struct lstExit {
  UInt16 listID;
  struct ListType *pList;
} lstExit;
```

Field Descriptions

listID Developer-defined ID of the list.

pList Pointer to a list structure (ListType).

lstSelectEvent

The list routine LstHandleEvent sends this event. When LstHandleEvent receives a lstEnterEvent, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of a list, a lstSelectEvent is added to the event queue; if not, a lstExitEvent is added to the event queue.

For this event, the data field contains the following structure:

```
struct lstSelect {
  UInt16 listID;
  struct ListType *pList;
  Int16 selection;
} lstSelect;
```

Field Descriptions

listID Developer-defined ID of the list.

pList Pointer to a list structure (ListType).

selection Item number (zero-based) of the new selection.

menuCloseEvent

This event is not currently used.

menuCmdBarOpenEvent

The menu routine `MenuHandleEvent` sends this event when the user enters the menu shortcut keystroke, causing the command toolbar to be displayed at the bottom of the screen. Applications might respond to this event by calling `MenuCmdBarAddButton` to add custom buttons to the command toolbar. Shared libraries or other non-application code resources can add buttons to the toolbar by registering to receive the `sysNotifyMenuCmdBarOpenEvent` notification. (See Chapter 36, "Notification Manager.")

For this event, the `data` field contains the following structure:

```
struct menuCmdBarOpen {
  Boolean preventFieldButtons;
  UInt8   reserved;
} menuCmdBarOpen;
```

Field Descriptions

`preventFieldButtons`	If `true`, the field manager does not add the standard cut, copy, paste, and undo buttons when the focus is on a field. If `false`, the field adds the buttons.
`reserved`	Unused.

To prevent the command toolbar from being displayed, respond to this event and return `true`. Returning `true` prevents the form manager from displaying the toolbar.

Compatibility

Implemented only if 3.5 New Feature Set is present.

menuEvent

The menu routine `MenuHandleEvent` sends this event:

- When the user selects an item from a pull-down menu

- When the user selects a menu command using the Graffiti command keystroke followed by an available command; for example, Command-C for copy
- When the user taps one of the buttons on the command toolbar and the button is set up to generate a `menuEvent`.

For this event, the `data` field contains the following structure:

```
struct menu {
  UInt16 itemID;
} menu;
```

Field Descriptions

`itemID` Item ID of the selected menu command.

menuOpenEvent

The menu routine `MenuHandleEvent` sends this event when a new active menu has been initialized. A menu becomes active the first time the user taps the Menu silk-screen button or taps the form's titlebar, and it might need to be re-initialized and reactivated several times during the life of an application.

A menu remains active until one of the following happens:

- A `FrmSetMenu` call changes the active menu on the form.
- A new form, even a modal form or alert panel, becomes active.

Suppose a user selects your application's About item from the Options menu then clicks the OK button to return to the main form. When the About dialog is displayed, it becomes the active form, which causes the main form's menu state to be erased. This menu state is not restored when the main form becomes active again. The next time the user requests the menu, it must be initialized again, so `menuOpenEvent` is sent again.

Applications might respond to this event by adding, hiding, or unhiding menu items using the functions `MenuAddItem`, `MenuHideItem`, or `MenuShowItem`.

A `menuCloseEvent` is defined by the system, but it is not currently sent. If you need to perform some cleanup (such as closing a

resource) after the menu item you added is no longer needed, do so in response to frmCloseEvent.

For this event, the data field contains the following structure:

```
struct menuOpen {
  UInt16  menuRscID;
  Int16   cause;
} menuOpen;
```

Field Descriptions

menuRscID	Resource ID of the menu.
cause	Reason for opening the menu. If menuButtonCause, the user tapped the Menu silkscreen button or tapped the form's titlebar, and the menu is going to be displayed. If menuCommandCause, the user entered the command keystroke, so the menu is becoming active without being displayed.

Compatibility

Implemented only if 3.5 New Feature Set is present.

nilEvent

A nilEvent is useful for animation, polling, and similar situations.

The event manager sends this event when there are no events in the event queue. This can happen if the routine EvtGetEvent is passed a time-out value (a value other than evtWaitForever, -1). If EvtGetEvent is unable to return an event in the specified time, it returns a nilEvent. Different Palm OS versions and different devices can send nilEvents under different circumstances, so you might receive a nilEvent even before the timeout has expired.

penDownEvent

The event manager sends this event when the pen first touches the digitizer.

The following data is passed with the event:

Field Descriptions

`penDown` Always `true`.

`tapCount` The number of taps received at this location.

`screenX` Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

`screenY` Window-relative position of the pen in pixels (number of pixels from the top left of the window).

penMoveEvent

The event manager sends this event when the pen is moved on the digitizer. Note that several kinds of UI objects, such as controls and lists, track the movement directly, and no `penMoveEvent` is generated.

The following data is passed with the event:

Field Descriptions

`penDown` Always `true`.

`tapCount` The number of taps received at this location.

`screenX` Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

`screenY` Window-relative position of the pen in pixels (number of pixels from the top left of the window).

penUpEvent

The event manager sends this event when the pen is lifted from the digitizer. Note that several kinds of UI objects, such as controls and lists, track the movement directly, and no `penUpEvent` is generated.

For this event, the `data` field contains the following structure:

```
struct _PenUpEventType {
  PointType start;
  PointType end;
```

```
};
```

Field Descriptions

`start`	Display-relative start point of the stroke.
`end`	Display-relative end point of the stroke.

In addition, the following data is passed with this event:

`penDown`	Always `false`.
`tapCount`	The number of taps received at this location.
`screenX`	Window-relative position of the pen in pixels (number of pixels from the left bound of the window).
`screenY`	Window-relative position of the pen in pixels (number of pixels from the top left of the window).

popSelectEvent

The form routine `FrmHandleEvent` sends this event when the user selects an item in a popup list.

For this event, the `data` field contains the following structure:

```
struct popSelect {
  UInt16 controlID;
  struct ControlType *controlP;
  UInt16 listID;
  struct ListType *listP;
  Int16 selection;
  Int16 priorSelection;
} popSelect;
```

Field Descriptions

`controlID`	Developer-defined ID of the resource.
`controlP`	Pointer to the control structure (`ControlType`) of the popup trigger object.
`listID`	Developer-defined ID of the popup list object.

`listP`	Pointer to the list structure (`ListType`) of the popup list object.
`selection`	Item number (zero-based) of the new list selection.
`priorSelection`	Item number (zero-based) of the prior list selection.

sclEnterEvent

The routine `SclHandleEvent` sends this event when it receives a `penDownEvent` within the bounds of a scroll bar.

Applications usually don't have to handle this event.

For this event, the `data` field contains the following structure:

```
struct sclEnter {
  UInt16 scrollBarID;
  struct ScrollBarType *pScrollBar;
} sclEnter;
```

Field Descriptions

`scrollBarID`	Developer-defined ID of the scroll bar resource.
`pScrollBar`	Pointer to the scroll bar structure.

sclExitEvent

The routine `SclHandleEvent` sends this event when the user lifts the pen from the scroll bar.

Applications that want to implement non-dynamic scrolling should wait for this event, then scroll the text using the values provided in `value` and `newvalue`.

Note that this event is sent regardless of previous `sclRepeatEvents`. If, however, the application has implemented dynamic scrolling, it doesn't have to catch this event.

For this event, the `data` field contains the following structure:

```
struct sclExit {
  UInt16 scrollBarID;
```

```
    struct ScrollBarType *pScrollBar;
    Int16 value;
    Int16 newValue;
  } sclExit;
```

Field Descriptions

`scrollBarID`	Developer-defined ID of the scroll bar resource.
`pScrollBar`	Pointer to the scroll bar structure.
`value`	Initial position of the scroll bar
`newvalue`	New position of the scroll bar. Given `value` and `newValue`, you can actually tell how much you have scrolled.

sclRepeatEvent

The routine `SclHandleEvent` sends this event when the pen is continually held within the bounds of a scroll bar.

Applications that implement dynamic scrolling should watch for this event. In dynamic scrolling, the display is updated as the user drags the scroll bar (not after the user releases the scroll bar).

For this event, the `data` field contains the following structure:

```
  struct sclRepeat {
    UInt16 scrollBarID;
    struct ScrollBarType *pScrollBar;
    Int16 value;
    Int16 newValue;
    Int32 time;
  } sclRepeat;
```

Field Descriptions

`scrollBarID`	Developer-defined ID of the scroll bar resource.
`pScrollBar`	Pointer to the scroll bar structure.
`value`	Initial position of the scroll bar.

`newValue`	New position of the scroll bar. Given `value` and `newValue`, you can actually tell how much you have scrolled.
`time`	System-ticks count when the event is added to the queue to determine when the next event should occur.

tblEnterEvent

The table routine `TblHandleEvent` sends this event when it receives a `penDownEvent` within the bounds of an active item in a table object.

For this event, the `data` field contains the following structure:

```
struct tblEnter {
  UInt16 tableID;
  struct TableType *pTable;
  Int16 row;
  Int16 column;
} tblEnter;
```

Field Descriptions

`tableID`	Developer-defined ID of the table.
`pTable`	Pointer to a table structure (`TableType`).
`row`	Row of the item.
`column`	Column of the item.

tblExitEvent

The table routine `TblHandleEvent` sends this event. When `TblHandleEvent` receives a `tblEnterEvent`, it tracks the pen until it's lifted from the display. If the pen is lifted within the bounds of the same item it went down in, a `tblSelectEvent` is added to the event queue; if not, a `tblExitEvent` is added to the event queue.

For this event, the `data` field contains the following structure:

```
struct tblExit {
  UInt16 tableID;
  struct TableType *pTable;
  Int16 row;
  Int16 column;
} tblExit;
```

Field Descriptions

`tableID`	Developer-defined ID of the table.
`pTable`	Pointer to a table structure (`TableType`).
`row`	Row of the item.
`column`	Column of the item.

tblSelectEvent

The table routine `TblHandleEvent` sends this event. When `TblHandleEvent` receives a `tblEnterEvent`, it tracks the pen until the pen is lifted from the display. If the pen is lifted within the bounds of the same item it went down in, a `tblSelectEvent` is added to the event queue; if not, a `tblExitEvent` is added to the event queue.

For this event, the `data` field contains the following structure:

```
struct tblSelect {
  UInt16 tableID;
  struct TableType *pTable;
  Int16 row;
  Int16 column;
} tblSelect;
```

Field Descriptions

`tableID`	Developer-defined ID of the table.
`pTable`	Pointer to a table structure (`TableType`).
`row`	Row of the item.
`column`	Column of the item.

winEnterEvent

The event manager sends this event when a window becomes the active window. This can happen in two ways: a call to WinSetActiveWindow is issued (FrmSetActiveForm calls this routine), or the user taps within the bounds of a window that is visible but not active. All forms are windows, but not all windows are forms; for example, the menu bar is a window but not a form.

For this event, the `data` field contains the following structure:

```
struct _WinEnterEventType {
  WinHandle enterWindow;
  WinHandle exitWindow;
};
```

Field Descriptions

`enterWindow`	Handle to the window we are entering. If the window is a form, then this is a pointer to a FormType structure; if not, it's a pointer to a WindowType structure.
`exitWindow`	Handle to the window we are exiting, if there is currently an active window, or zero if there is no active window. If the window is a form, then this is a pointer to a FormType structure; if not, it's a pointer to a WindowType structure.

winExitEvent

This event is sent by the event manager when a window is deactivated. A window is deactivated when another window becomes the active window (see winEnterEvent).

For this event, the `data` field contains the following structure:

```
struct _WinExitEventType {
  WinHandle enterWindow;
  WinHandle exitWindow;
};
```

Field Descriptions

`enterWindow` Handle to the window we are entering. If the window is a form, then this is a pointer to a `FormType` structure; if not, it's a pointer to a `WindowType` structure.

`exitWindow` Handle to the window we are exiting. If the window is a form, then this is a pointer to a `FormType` structure; if not, it's a pointer to a `WindowType` structure.

4

Categories

This chapter describes the categories API as declared in the header file `Category.h`. It discusses the following topics:

- Category Data Structures
- Category Constants
- Category Functions

For more information on categories see the section "Categories" on page 107 in the *Palm OS Programmer's Companion*.

Category Data Structures

An `AppInfo` block can hold any data at all. The category APIs provide a way to implement categories and use the `AppInfo` block as the storage area. An application could implement the category popup on its own without this API and use the Data Manager category routines, and /or the `AppInfo` block, as it chooses.

This API requires that the `AppInfo` block be used like this:

AppInfoType

```
typedef struct {
  UInt16   renamedCategories;
  Char     categoryLabels
             [dmRecNumCategories]
             [dmCategoryLength];
  UInt8    categoryUniqIDs
             [dmRecNumCategories];
  UInt8    lastUniqID;
  UInt8    padding;
} AppInfoType;

typedef AppInfoType *AppInfoPtr;
```

Field Descriptions

`renamedCategories`	Used by `CategorySetName` as a bit field indicating which categories have been renamed. Usually cleared by a conduit.
`categoryLabels`	An array of strings containing the category names.
`dmRecNumCategories`	Number of categories in the list.
`dmCategoryLength`	Length of the category names.
`categoryUniqIDs`	Category IDs used for synchronization. Unique IDs generated by the device are between 0 - 127. Those from the PC are 128 - 255.
`lastUniqID`	Used for sorting and assigning unique IDs.

Category Constants

The following category constants are defined:

Constant	Value	Description
`categoryHideEditCategory`	10000	Used as an argument to `CategoryCreateList` to suppress adding the "Edit Categories" item to the list.
`categoryDefaultEditCategoryString`	10001	Used as an argument to `CategoryCreateList` to show the default "Edit Categories" item in the list.

Compatibility

The functionality of the constants `categoryHideEditCategory` and `categoryDefaultEditCategoryString` is present only if the 3.5 New Feature Set is present.

Category Functions

CategoryCreateList

Purpose Read a database's categories and store them in a list.

Prototype

```
void CategoryCreateList (DmOpenRef db,
ListType *listP, Uint16 currentCategory,
Boolean showAll, Boolean showUneditables,
UInt8 numUneditableCategories,
UInt32 editingStrID, Boolean resizeList)
```

Parameters

`->db` Opened database containing category info.

`<-listP` A pointer to the list of category names. See `ListType`.

`->currentCategory` Category to select.

`->showAll` `true` to have an "All" category.

`->showUneditables` `true` to show uneditable categories.

`->numUneditableCategories` This is the number of categories, starting with the first one at zero, that may not have their names edited by the user. For example, it's common to have an "Unfiled" category at position zero which is not editable.

`->editingStrID` The resource ID of a string to use with the "Edit Categories" list item.

If you don't want users to edit categories, pass the `categoryHideEditCategory` constant.

	If you want to allow users to edit categories, pass the `categoryDefaultEditCategoryString` constant. To display an alternate string, pass a tSTR resource ID of your own string.
`->resizeList`	`true` to resize the list to the number of categories. Set to `true` for pop-ups, `false` otherwise.

Result A memory block is allocated containing the list of categories. The `ListType` in `listP` must be allocated outside this function. However, this function allocates some structs that are stored INSIDE the `ListType`, so `CategoryFreeList` must be called when you are done with the list to free the memory block.

Comments You use this function to create a list of categories to display in your application's user interface, usually by calling `LstDrawList` or `LstPopupList`. The category list is obtained from the `AppInfoType` structure of the database specified by the `db` parameter.

If the `showAll` parameter is `true`, the "All" item is first in the list, followed by the editable categories in the database and then the categories that cannot be edited. The option to edit categories is last in the list and can be suppressed if desired. If the current selection is not in any category, it is marked "Unfiled".

Compatibility Implemented only if 2.0 New Feature Set is present.

The functionality of the constants `categoryDefaultEditCategoryString` and `categoryHideEditString` is available only if 3.5 New Feature Set is present. In earlier versions, you can suppress the Edit Categories string by passing 0 for the `editingStrID` parameter, or include the item by passing `categoryEditStrID`.

See Also `CategoryCreateListV10`

CategoryCreateListV10

Purpose Read a database's categories and set the category list.

This function is obsolete and should not be used.

Prototype

```
void CategoryCreateListV10 (DmOpenRef db,
ListType *lst, UInt16 currentCategory,
Boolean showAll)
```

Parameters

`->db`	Database containing categories to extract.
`<-lst`	List object to load categories into.
`->currentCategory`	Set as the current selection in the resulting list.
`->showAll`	`true` if an "All" category should be included in the list.

Result Returns nothing.

Compatibility This function corresponds to the Palm OS® 1.0 version of `CategoryCreateList`.

NOTE: Obsolete functions are provided ONLY for backward compatibility; for example, so a 1.0 application will work on 3.x OS releases. New code should not call these routines!

See Also `CategoryCreateList`

CategoryEdit

Purpose Event handler for the "Edit Categories" dialog. Called by `CategorySelect` if the user chooses the `Edit Category` line. (If the Edit Category line is present)

`->db`	Database containing the categories to be edited.

`<-categoryP` Set to the category selected when the dialog is done.

`->titleStrID` Title of the dialog bar.

`->numUneditableCategories`
This is the number of categories (starting with the first one at zero) that may not have their names edited by the user. For example, it's common to have an "Unfiled" category at position zero which is not editable.

Result Returns `true` if any of the following conditions are `true`:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Compatibility This function was revised for Palm OS 2.0, and Palm OS 3.0. In Palm OS 3.0, the `numUneditableCategories` parameter was added.

NOTE: This enhancement is implemented only if 3.0 New Feature Set is present.

See Also `CategoryEditV20`, `CategoryEditV10`

CategoryEditV20

Purpose Event handler for the Edit Categories dialog.Called by CategorySelect if the user chooses the Edit Category line. (If the Edit Category line is present.)

This function is obsolete and should not be used.

Prototype

```
Boolean CategoryEdit (DmOpenRef db,
UInt16 *categoryP, UInt32 titleStrID)
```

Parameters `->db` Database containing the categories to be edited.

`<-categoryP`	Set to the category selected when the dialog is done.
`->titleStrID`	Title of the dialog bar.

Result Returns `true` if any of the following conditions are `true`:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Compatibility This function corresponds to the Palm OS 2.0 version of `CategoryEdit`. Implemented only if 3.0 New Feature Set is present.

NOTE: Obsolete functions are provided ONLY for backward compatibility. For example, so a 1.0 application will work on 3.x OS releases. New code should not call these routines!

See Also `CategoryEdit`, `CategoryEditV10`

CategoryEditV10

Purpose Event handler for the Edit Categories dialog. Called by `CategorySelect` if the user chooses the "Edit Category" line. (If the Edit Category line is present.)

This function is obsolete and should not be used.

Prototype

```
Boolean CategoryEditV10 (DmOpenRef db,
UInt16 *categoryP)
```

Parameters

`->db`	Database containing the categories to be edited.
`<-categoryP`	Current category (index into the database).

Result Returns `true` if any of the following conditions are true:

- The current category is renamed.

- The current category is deleted.
- The current category is merged with another category.

Compatibility This function corresponds to the Palm OS 1.0 version of `CategoryEdit`.

NOTE: Obsolete functions are provided ONLY for backward compatibility; for example, so a 1.0 application will work on 3.x OS releases. New code should not call these routines!

See Also `CategoryEdit`, `CategoryEditV20`

CategoryFind

Purpose Return the unique ID of the category that matches the name passed.

Prototype

```
UInt16 CategoryFind (DmOpenRef db, const
Char *name)
```

Parameters

`->db`	Database to search for the passed category.
`->name`	Category name.

Result Returns the category index.

CategoryFreeList

Purpose This routine unlocks or frees memory locked or allocated by `CategoryCreateList`.

Prototype

```
void CategoryFreeList (DmOpenRef db, const
ListType *listP, Boolean showAll,
UInt32 editingStrID)
```

Parameters

`->db`	Categories database.
`->listP`	Pointer to the category list.

`->showAll`	`true` if the list was created with an "All" category.
`->editingStrID`	The `editingStrID` should be the same as that passed to `CategoryCreateList`. The function will unlock the resource.

Result Returns nothing.

Comments Calling this function doesn't remove the categories from the passed database. It frees the items in the list. The developer must manage the `ListType` structure.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also CategoryFreeListV10

CategoryFreeListV10

Purpose Unlock or free memory locked or allocated by `CategoryCreateListV10` which was attached to the passed list object.

This function is obsolete and should not be used.

Prototype

```
void CategoryFreeListV10 (DmOpenRef db, const
ListType *lst)
```

Parameters

`->db`	Database containing the categories.
`->lst`	Pointer to the category list containing the memory to be freed.

Result Returns nothing.

Compatibility This function corresponds to the Palm OS 1.0 version of `CategoryFreeList`.

NOTE: Obsolete functions are provided ONLY for backward compatibility; for example, so a 1.0 application will work on 3.x OS releases. New code should not call these routines!

See Also CategoryFreeList

CategoryGetName

Purpose Return the name of the specified category.

Prototype

```
void CategoryGetName (DmOpenRef db, UInt16 index,
Char *name)
```

Parameters

`->db`	Database that contains the categories.
`->index`	Category index.
`<-name`	Buffer to hold category name. Buffer should be `dmCategoryLength` in size.

Result Stores the category name in the `name` buffer passed.

CategoryGetNext

Purpose Given a category index, this function returns the index of the next category. Note that categories are not stored sequentially.

Prototype

```
UInt16 CategoryGetNext (DmOpenRef db,
UInt16 index)
```

Parameters

`->db`	Database that contains the categories.
`->index`	Category index.

Result Category index of next category.

Comments Don't use this function to search for a category. Instead, use it to allow your users to cycle through categories, for example, using the hard-button scroll bars on the device.

Compatibility In Palm OS 1.0, the system chose `Unfiled` as one category.

In Palm OS 2.0 and later, the system skips both `Unfiled` and empty records. That is, if a category contains no records, then its index will not be returned by this function.

CategoryInitialize

Purpose Initialize the category names, IDs, and flags.

Prototype

```
void CategoryInitialize (AppInfoPtr appInfoP,
UInt16 localizedAppInfoStrID)
```

Parameters

`->appInfoP` Application info pointer. See `AppInfoType`.

`->localizedAppInfoStrID` Resource ID of the localized category names. This must be a resource of the type `appInfoStringsRsc`.

Result Returns nothing.

Comments Used to make sure the first field in your application info block is of type `AppInfoType`, and to initialize category names.

Compatibility Implemented only if 2.0 New Feature Set is present.

CategorySelect

Purpose Process the selection and editing of categories.Usually you call this when the user taps the category pop-up trigger.

Prototype

```
Boolean CategorySelect (DmOpenRef db, const
FormType *frm, UInt16 ctlID, UInt16 lstID,
Boolean title, UInt16 *categoryP,
char *categoryName, UInt8 numUneditableCategories,
UInt32 editingStrID)
```

Parameters

`->db` Database that contains the categories.

`->frm` Form that contains the category popup list.

`->ctlID` ID of the popup trigger.

`->lstID` ID of the popup list.

`->title` `true` if the popup trigger is on the title line, which means that an "All" choice should be added to the list. Pass `false` if the popup trigger appears in a form where a specific record is being modified or any where else the "All" choice should not appear.

`<->categoryP` Current category (pointer into `db` structure).

`<->categoryName`
Name of the current category.

`->numUneditableCategories`
This is the number of categories, starting with the first one at zero, that may not have their names edited by the user. For example, it's common to have an "Unfiled" category at position zero which is not editable.

`->editingStrID`
The resource ID of a string to use with the "Edit Categories" list item.

If you don't want users to edit categories, pass the `categoryHideEditCategory` constant.

If you want to allow users to edit categories, pass the `categoryDefaultEditCategoryString` constant.

To display an alternate string, pass a tSTR resource ID of your own string.

Result Returns `true` if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Comments This function calls `CategoryEdit` if the user selects the Edit Categories option in the list.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `CategorySelectV10`

CategorySelectV10

Purpose Process the selection and editing of categories.

This function is obsolete and should not be used.

Prototype

```
Boolean CategorySelectV10 (DmOpenRef db, const
FormType *frm, UInt16 ctlID, UInt16 lstID,
Boolean title, UInt16 *categoryP,
Char *categoryName)
```

Parameters

`->db`	Database that contains the categories.
`->frm`	Form that contains the category popup list.
`->ctlID`	ID of the popup trigger.
`->lstID`	ID of the popup list.
`->title`	`true` if the popup trigger is on the title line.

`<->categoryP` Current category (index into db structure).

`<->categoryName` Name of the current category.

Result Returns `true` if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Compatibility This function corresponds to the Palm OS 1.0 version of `CategorySelect`.

NOTE: Obsolete functions are provided ONLY for backward compatibility; for example, so a 1.0 application will work on 3.x OS releases. New code should not call these routines!

CategorySetName

Purpose Set the category name and rename bits. A `NULL` pointer removes the category name.

Prototype

```
void CategorySetName (DmOpenRef db, UInt16 index,
const Char nameP)
```

Parameters

`->db` Database containing the categories to change.

`->index` Index of category to set.

`->nameP` A category name (null-terminated) or `NULL` pointer to remove the category.

Result Returns nothing.

Compatibility Implemented only if 2.0 New Feature Set is present.

CategorySetTriggerLabel

Purpose Set the label displayed by the category trigger. The category name is truncated if it is larger than the system default maximum width. `CategorySetTrigger` calls `CategoryTruncateName`, with a default (system-provided) width.

Prototype

```
void CategorySetTriggerLabel (ControlType *ctl,
Char *name)
```

Parameters

`->ctl`	Pointer to control object to relabel.
`->name`	Pointer to the name of the new category.

Result Does not copy the string. `Ctl` points to the passed string when done. See `CtlSetLabel`.

CategoryTruncateName

Purpose Truncate a category name so that it's short enough to display. The category name is truncated if it's longer than `maxWidth`.

Prototype

```
void CategoryTruncateName (Char *name,
UInt16 maxWidth)
```

Parameters

`->name`	Category name to truncate.
`->maxWidth`	Maximum size, in pixels, of truncated category (including ellipsis).

Result Returns nothing. Stores the changed category in `name`

5

Clipboard

This chapter provides reference material for the clipboard API defined in `Clipboard.h`. It covers:

- Clipboard Data Structures
- Clipboard Functions

Clipboard Data Structures

ClipboardFormatType

The `ClipboardFormatType` enum specifies the type of data to add to the clipboard or retrieve from the clipboard.

```
enum clipboardFormats {
  clipboardText,
  clipboardInk,
  clipboardBitmap };
typedef enum clipboardFormats
  ClipboardFormatType;
```

Value Descriptions

`clipboardText`	Textual data. This is the most commonly used clipboard.
`clipboardInk`	Reserved.
`clipboardBitmap`	Bitmap data.

Clipboards for each type of data are separately maintained. That is, if you add a string of text to the clipboard, then add a bitmap, then ask to retrieve a `clipboardText` item from the clipboard, you will receive the string you added before the bitmap; the bitmap does not overwrite textual data and vice versa.

Clipboard Functions

ClipboardAddItem

Purpose Add the item passed to the specified clipboard. Replaces the current item (if any) of that type.

Prototype

```
void ClipboardAddItem
(const ClipboardFormatType format,
const void *ptr, UInt16 length)
```

Parameters

-> `format`	Text, ink, bitmap, etc. See `ClipboardFormatType`.
-> `ptr`	Pointer to the item to place on the clipboard.
-> `length`	Size in bytes of the item to place on the clipboard.

Result Returns nothing.

Comments The clipboard makes a copy of the data that you pass to this function. Thus, you may free any data that you've passed to the clipboard without destroying the contents of the clipboard. You may also add constant data or stack-based data to the clipboard.

See Also `FldCut`, `FldCopy`

ClipboardAppendItem

Purpose Append data to the item on the clipboard.

Prototype

```
Err ClipboardAppendItem
(const ClipboardFormatType format,
const void *ptr, UInt16 length)
```

Parameters

-> `format`	Text, ink, bitmap, etc. See `ClipboardFormatType`. This function is intended to be used only for the `clipboardText` format.
-> `ptr`	Pointer to the data to append to the item on the clipboard.
-> `length`	Size in bytes of the data to append to the clipboard.

Result 0 upon success or `memErrNotEnoughSpace` if there is not enough space to append the data to the clipboard.

Comments This function differs from `ClipboardAddItem` in that it does not overwrite data already on the clipboard. It allows you to create a large text item on the clipboard from several small disjointed pieces. When other applications retrieve the text from the clipboard, it's retrieved as a single unit.

This function simply appends the specified item to the item already on the clipboard without attempting to parse the format. It's assumed that you'll call it several times over a relatively short interval and that no other application will attempt to retrieve text from the clipboard before your application is finished appending.

Compatibility Implemented only if 3.2 New Feature Set is present.

ClipboardGetItem

Purpose Return the handle of the contents of the clipboard of a specified type and the length of a clipboard item.

Prototype

```
MemHandle ClipboardGetItem
(const ClipboardFormatType format, UInt16 *length)
```

Parameters

-> `format`	Text, ink, bitmap, etc. See `ClipboardFormatType`.
<- `length`	The length in bytes of the clipboard item is returned here.

Result Handle of the clipboard item.

Comments The handle returned is a handle to the actual clipboard chunk. It is not suitable for passing to any API that modifies memory (such as `FldSetTextHandle`). Consider this to be read-only access to the chunk. Copy the contents of the clipboard to your application's own storage as soon as possible and use that reference instead of the handle returned by this function.

Don't free the handle returned by this function; it is freed when a new item is added to the clipboard.

Text retrieved from the clipboard does not have a `NULL` terminator. You must use the `length` parameter to determine the length in bytes of the string you've retrieved.

6

Controls

This chapter describes the control object API as declared in the header file `Control.h`. It discusses the following topics:

- Control Data Structures
- Control Resources
- Control Functions

For more information on controls, see the section "Controls" in the *Palm OS Programmer's Companion*.

Control Data Structures

ButtonFrameType

The `ButtonFrameType` enum specifies the border style for the button. It defines values for the `frame` field of `ControlAttrType`.

```
enum buttonFrames {noButtonFrame,
      standardButtonFrame, boldButtonFrame,
      rectangleButtonFrame};
typedef enum buttonFrames ButtonFrameType;
```

Value Descriptions

`noButtonFrame`	The button has no border.
`standardButtonFrame`	Standard button rectangular border with rounded corners.
`boldButtonFrame`	Bolded rectangular border with rounded corners.
`rectangleButtonFrame`	Rectangular border with square corners.

ControlAttrType

The `ControlAttrType` bit field specifies the control's visible characteristics. It is defined as follows:

```
typedef struct {
  UInt8 usable    :1;
  UInt8 enabled   :1;
  UInt8 visible   :1;
  UInt8 on        :1;
  UInt8 leftAnchor:1;
  UInt8 frame     :3;
  UInt8 drawnAsSelected : 1;
  UInt8 graphical :1;
  UInt8 vertical  :1;
} ControlAttrType;
```

Your code should treat the `ControlAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Field Descriptions

`usable`	If 0, the control is not considered to be part of the interface of the current application, and it doesn't appear on screen. You can use `CtlSetUsable`, `CtlShowControl`, or `CtlHideControl` to set or clear this value.
`enabled`	If 0, the control is visible but doesn't respond to the pen. This value is set by `CtlSetEnabled` and returned by `CtlEnabled`.
`visible`	Set and cleared internally when the control is drawn (`CtlDrawControl`) and erased (`CtlEraseControl`).
`on`	If set, the control has the value "on." For example, a check box that has the on value has a check mark displayed in it. Use `CtlGetValue` and `CtlSetValue` to retrieve and set this value.

`leftAnchor`	Used by controls that expand and shrink their width when the label is changed. If this attribute is set, the left bound of the control is fixed.
`frame`	The type of frame drawn around the button controls. See `ButtonFrameType` for possible values. Only button controls use this attribute; for all other controls, the `ControlStyleType` determines the frame.
`drawnAsSelected`	Used on Palm OS® release 3.5 for button controls that contain no text (indicating that the button is displayed on top of a bitmap). If set, the button is drawn as inverted. If clear, the button is drawn normally.
`graphical`	If set, the control is a graphical control, slider, or feedback slider.
`vertical`	Not currently used.

Compatibility

The `drawnAsSelected`, `graphical`, and `vertical` attributes are only present if 3.5 New Feature Set is present.

ControlPtr

The `ControlPtr` is a pointer to a `ControlType` structure.

```
typedef ControlType* ControlPtr;
```

ControlStyleType

The `ControlStyleType` enum specifies values for the `ControlType` `style` field, which specifies the type of the control (button, push button, and so on).

```
enum controlStyles {buttonCtl, pushButtonCtl,
    checkboxCtl, popupTriggerCtl,
    selectorTriggerCtl, repeatingButtonCtl,
    sliderCtl, feedbackSliderCtl};
```

```
typedef enum controlStyles ControlStyleType;
```

Value Descriptions

`buttonCtl`	Button. Buttons display a text label in a box. The `ButtonFrameType` specifies the type of box.
`pushButtonCtl`	Push button. Selecting a push button inverts its display so that it appears highlighted.
`checkboxCtl`	Check box. Check boxes display a setting of either on (checked) or off (unchecked)
`popupTriggerCtl`	Popup trigger. Popup triggers display a graphic element followed by a text label. They are used to display popup lists.
`selectorTriggerCtl`	Selector trigger. Selector triggers display a text label surrounded by a gray rectangular frame. The control expands or contracts to the width of the new label.
`repeatingButtonCtl`	Repeating button. Repeating buttons look like buttons; however, a repeating button is repeatedly selected if the user holds the pen on it.

`sliderCtl`	Slider. Sliders display two bitmaps: one representing the current value (the thumb), and another representing the scale of available values. The user can slide the thumb to the left or the right to change the value.
`feedbackSliderCtl`	Feedback slider. A feedback slider looks like a slider; however, a feedback slider sends events each time the thumb moves while the pen is still down. A regular slider sends an event only when the user releases the pen.

Compatibility

The `sliderCtl` and `feedbackSliderCtl` values are only defined if 3.5 New Feature Set is present.

ControlType

The `ControlType` structure defines the type and characteristics of a control. It is defined as follows:

```
typedef struct {
  UInt16            id;
  RectangleType     bounds;
  Char *            text;
  ControlAttrType   attr;
  ControlStyleType  style;
  FontID            font;
  UInt8             group;
  UInt8             reserved;
} ControlType;
```

Your code should treat the `ControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. Use the functions specified in the descriptions below to retrieve and set the values. Do not attempt to change structure member values directly.

Field Descriptions

`id` ID value you specified when you created the control resource.

`bounds` Bounds of the control, in window-relative coordinates. The control's text label is clipped to the control's bounds. The control's frame is drawn around (outside) the bounds of the control. `FrmGetObjectBounds` and `FrmSetObjectBounds` retrieve and set this value.

`text` Pointer to the control's label. If `text` is `NULL`, the control has no label. Use `CtlGetLabel` and `CtlSetLabel` to retrieve and set this value.

`attr` Control attributes. See `ControlAttrType`.

`style` Style of the control. See `ControlStyleType`.

`font` Font to use to draw the control's label.

`group` Group ID of a push button or a check box that is part of an exclusive group. The control routines don't automatically turn one control off when another is selected. It's up to the application or a higher-level object, like a dialog box, to manage this.

`reserved` Reserved for future use.

GraphicControlType

The `GraphicControlType` struct defines a graphical control. A graphical control is like any other control except that it displays a bitmap in place of the text label.

```
typedef struct GraphicControlType {
  UInt16            id;
  RectangleType     bounds;
  DmResID           bitmapID;
  DmResID           selectedBitmapID;
  ControlAttrType   attr;
  ControlStyleType  style;
  FontID            unused;
  UInt8             group;
```

```
    UInt8               reserved;
  } GraphicControlType;
```

Your code should treat the `GraphicControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. Use the functions specified in the descriptions below to retrieve and set the values. Do not attempt to change structure member values directly.

Field Descriptions

`id`	ID value you specified when you created the control resource.
`bounds`	Bounds of the control, in window-relative coordinates. The control's frame is drawn around (outside) the bounds of the control. `FrmGetObjectBounds` and `FrmSetObjectBounds` retrieve and set this value.
`bitmapID`	Resource ID of the bitmap to display in the button. You can use `CtlSetGraphics` to change this value.
`selectedBitmapID`	If the button should show a different bitmap when selected, this field contains the resource ID of that bitmap. You typically use this field for push buttons or repeating buttons. `CtlSetGraphics` can change this value.
`attr`	Control attributes. See `ControlAttrType`. For a graphical control, the `graphical` attribute must be set.
`style`	Style of the control. See `ControlStyleType`. A graphical control can be any type of control other than `checkboxCtl`.
`unused`	Unused.

`group`	Group ID of a push button that is part of an exclusive group. The control routines don't automatically turn one control off when another is selected. It's up to the application or a higher-level object, like a dialog box, to manage this.
`reserved`	Reserved for future use.

Compatibility

This struct is defined only if 3.5 New Feature Set is present.

SliderControlType

The `SliderControlType` struct defines a slider control or a feedback slider control.

```
typedef struct SliderControlType {
  UInt16              id;
  RectangleType       bounds;
  DmResID             thumbID;
  DmResID             backgroundID;
  ControlAttrType     attr;
  ControlStyleType    style;
  UInt8               reserved;
  Int16               minValue;
  Int16               maxValue;
  Int16               pageSize;
  Int16               value;
  MemPtr              activeSliderP;
} SliderControlType;
```

Your code should treat the `SliderControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. You can use `CtlSetSliderValues` to set new minimum, maximum, page size, and current values, and `CtlGetSliderValues` to retrieve these values. Do not attempt to change structure member values directly.

Field Descriptions

`id`	ID value you specified when you created the control resource.
`bounds`	Bounds of the control, in window-relative coordinates. `FrmGetObjectBounds` and `FrmSetObjectBounds` retrieve and set this value.
`thumbID`	Resource ID of the bitmap to use for the slider knob (called the "thumb"). If `NULL`, the default bitmap is used.
`backgroundID`	Resource ID of the bitmap to use for the slider background. If `NULL`, the default bitmap is used.
`attr`	Control attributes. See `ControlAttrType`. For a slider, the `graphical` attribute is set.
`style`	Style of the control. See `ControlStyleType`. Must be `sliderCtl` or `feedbackSliderCtl`.
`reserved`	Reserved for future use.
`minValue`	Value of the slider when the thumb is all the way to the left.
`maxValue`	Value of the slider when the thumb is all the way to the right.
`pageSize`	Amount by which to increase or decrease the slider value when the user taps to the right or left of the thumb.

`value`	Current value represented by the slider. Use `CtlGetValue` and `CtlSetValue` to retrieve and set this value.
`activeSliderP`	Pointer to a memory location used when the slider is active. A slider is active if it is currently being drawn or if it is tracking the pen. If the slider is inactive, this pointer is `NULL`.

Compatibility

This struct is defined only if 3.5 New Feature Set is present.

Control Resources

Different resources are associated with different controls, as follows:

- Button—Button Resource (tBTN)
- Popup trigger— Popup Trigger Resource (tPUT)
- Selector trigger—Selector Trigger Resource (tSLT)
- Repeat control—Repeating Button Resource (tREP)
- Push button—Push Button Resource (tPBN)
- Check box—Check Box Resource (tCBX)
- Slider— Slider Resource (tsld)
- Feedback slider— Feedback Slider Resource (tslf)

Control Functions

CtlDrawControl

Purpose Draw a control object (and the text or graphic in it) on screen.

Prototype
```
void CtlDrawControl (ControlType *controlP)
```

Parameters

`-> controlP`	Pointer to the control object to draw. (See `ControlType`.)

Result Returns nothing.

Comments The control is drawn only if its `usable` attribute is `true`. This function sets the `visible` attribute to `true`.

Compatibility In releases prior to Palm OS® 3.5, it is common to create graphical buttons by drawing a button with no text label on top of a bitmap. If 3.5 New Feature Set is present, you should use graphical controls instead. (See `GraphicControlType`.) `CtlDrawControl` attempts to provide backward compatibility for the old-style graphical buttons.

See Also `CtlSetUsable`, `CtlShowControl`

CtlEnabled

Purpose Return `true` if the control responds to the pen.

Prototype
```
Boolean CtlEnabled (const ControlType *controlP)
```

Parameters

`-> controlP`	Pointer to control object. (See `ControlType`.)

Result Returns `true` if the controls object responds to the pen; `false` if not.

Comments This function provides no indication of whether the control is visible on the screen. A control that doesn't respond to the pen may be visible, and if so, its appearance is no different from controls that do respond to the pen. You might use such a control to display some state of your application that cannot be modified.

See Also `CtlSetEnabled`

CtlEraseControl

Purpose Erase a usable and visible control object and its frame from the screen.

Prototype

```
void CtlEraseControl (ControlType *controlP)
```

Parameters

`-> controlP`	Pointer to control object to erase. (See `ControlType`.)

Comments This function sets the `visible` attribute to `false`. If 3.5 New Feature Set is present, it also sets the `drawnAsSelected` attribute to `false`.

Don't call this function directly; instead, use `FrmHideObject`, which calls this function.

CtlGetLabel

Purpose Return a character pointer to a control's text label.

Prototype

```
const Char *CtlGetLabel
(const ControlType *controlP)
```

Parameters

`-> controlP`	Pointer to control object. (See `ControlType`.)

Result Returns a pointer to a null-terminated string.

Comments Make sure that `controlP` is not a graphical control or a slider control. The graphical control and slider control structures do not contain a text label field.

See Also `CtlSetLabel`

CtlGetSliderValues

Purpose Return current values used by a slider control.

Prototype

```
void CtlGetSliderValues (const ControlType *ctlP,
UInt16 *minValueP, UInt16 *maxValueP,
UInt16 *pageSizeP, UInt16 *valueP)
```

Parameters

-> `ctlP`	Pointer to a control object. (See `ControlType`.)
<- `minValueP`	The slider's minimum value. Pass `NULL` if you don't want to retrieve this value.
<- `maxValueP`	The slider's maximum value. Pass `NULL` if you don't want to retrieve this value.
<- `pageSizeP`	The slider's page size value. Pass `NULL` if you don't want to retrieve this value.
<- `valueP`	The slider's current value. Pass `NULL` if you don't want to retrieve this value.

Result Returns nothing. The slider's values are returned in the parameters to this function.

Comments If `ctlP` is not a slider or a feedback slider, this function immediately returns.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlSetSliderValues`, `SliderControlType`

CtlGetValue

Purpose Return the current value of the specified control.

Prototype
```
Int16 CtlGetValue (const ControlType *controlP)
```

Parameters

`-> controlP` Pointer to a control object. (See `ControlType`.)

Result Returns the current value of the control. For most controls the return value is either 0 (off) or 1 (on). For sliders, this function returns the value of the `value` field.

See Also `CtlSetValue`, `FrmGetControlGroupSelection`, `FrmSetControlGroupSelection`, `FrmGetControlValue`, `FrmSetControlValue`

CtlHandleEvent

Purpose Handle event in the specified control object.

Prototype
```
Boolean CtlHandleEvent (ControlType *controlP,
EventType pEvent)
```

Parameters

`-> controlP` Pointer to control object. (See `ControlType`.)

`-> pEvent` Pointer to an `EventType` structure.

Result Returns `true` if an event is handled by this function. Events that are handled are:

- `penDownEvent` — If the pen is within the bounds of the control
- `ctlEnterEvent`, `ctlRepeatEvent`, and `ctlExitEvent`— If the control ID in the event data matches the control's ID.

Comments The control object must be usable, visible, and respond to the pen for this function to handle the event.

When this routine receives a `penDownEvent`, it checks if the pen position is within the bounds of the control object. If it is, a `ctlEnterEvent` is added to the event queue and the routine exits.

When this routine receives a `ctlEnterEvent`, the control object is redrawn as necessary as either selected or deselected, depending on its previous state.

When this routine receives a `ctlEnterEvent` or `ctlRepeatEvent`, it checks that the control ID in the passed event record matches the ID of the specified control. If they match, this routine tracks the pen until it comes up or until it leaves the object's bounds. When that happens, `ctlSelectEvent` is sent to the event queue if the pen came up in the bounds of the control. If the pen exits the bounds, a `ctlExitEvent` is sent to the event queue.

CtlHideControl

Purpose Set a control's `usable` attribute to `false` and erase the control from the screen.

Prototype

```
void CtlHideControl (ControlType *controlP)
```

Parameters

`-> controlP`	Pointer to the control object to hide. (See `ControlType`.)

Result Returns nothing.

Comments A control that is not usable doesn't draw and doesn't respond to the pen.

This function is the same as `CtlEraseControl` except that it also sets `usable` to `false` (in addition to setting `visible` to `false`).

Don't call this function directly; instead, use `FrmHideObject`, which performs the same function and works for all user interface objects.

See Also `CtlShowControl`

CtlHitControl

Purpose Simulate tapping a control. This function adds a `ctlSelectEvent` to the event queue.

Prototype

```
void CtlHitControl (const ControlType *controlP)
```

Parameters

`-> controlP`	Pointer to a control object. (See `ControlType`.)

Result Returns nothing.

Comments Useful for testing.

CtlNewControl

Purpose Create a new control object dynamically and install it in the specified form.

Prototype

```
ControlType *CtlNewControl (void **formPP,
UInt16 ID, ControlStyleType style,
const Char *textP, Coord x, Coord y, Coord width,
Coord height, FontID font, UInt8 group,
Boolean leftAnchor)
```

Parameters

`<-> formPP`	Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the `formPP` value may change if the object moves in memory. In subsequent calls, always use the new `formPP` value returned by this function.
`-> ID`	Symbolic ID of the control.
`-> style`	A `ControlStyleType` value specifying the kind of control to create: button, push button, repeating button, check box, popup trigger, or popup selector. To create a graphical control or slider control dynamically, use `CtlNewGraphicControl` or `CtlNewSliderControl`, respectively.

`-> textP`	Pointer to the control's label text. If `textP` is `NULL`, the control has no label. Only buttons, push buttons, and text boxes have text labels. Because the contents of this pointer are copied into their own buffer, you can free the `textP` pointer any time after the `CtlNewControl` function returns. The buffer into which this string is copied is freed automatically when you remove the control from the form or delete the form.
`-> x`	Horizontal coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
`-> y`	Vertical coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
`-> width`	Width of the control, expressed in pixels. Valid values are 1–160. If the value of either of the `width` or `height` parameters is 0, the control is sized automatically as necessary to display the text passed as the value of the text parameter.
`-> height`	Height of the control, expressed in pixels. Valid values are 1–160. If the value of either of the `width` or `height` parameters is 0, the control is sized automatically as necessary to display the text passed as the value of the text parameter.
`-> font`	Font used to draw the control's label.
`-> group`	Group ID of a push button or a check box that is part of an exclusive group. The control routines don't turn one control off automatically when another is selected. It's up to the application or a higher-level object, such as a dialog box, to manage this.

`-> leftAnchor`	`true` specifies that the left bound of this control is fixed. This attribute is used by controls that resize dynamically in response to label text changes.

Result Returns a pointer to the new control.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `CtlValidatePointer`, `FrmRemoveObject`

CtlNewGraphicControl

Purpose Create a new graphical control dynamically and install it in the specified form.

Prototype

```
GraphicControlType *CtlNewGraphicControl
(void **formPP, UInt16 ID, ControlStyleType style,
DmResID bitmapID, DmResID selectedBitmapID,
Coord x, Coord y, Coord width, Coord height,
UInt8 group, Boolean leftAnchor)
```

Parameters

`<-> formPP`	Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the `formPP` value may change if the object moves in memory. In subsequent calls, always use the new `formPP` value returned by this function.
`-> ID`	Symbolic ID of the control.
`-> style`	A `ControlStyleType` value specifying the kind of control to create: button, push button, popup trigger, repeating button, or popup selector. Graphic controls cannot be check boxes.
`-> bitmapID`	Resource ID of the bitmap to display on the control.

`-> selectedBitmapID`	Resource ID of the bitmap to display when the control is selected, if different from `bitmapID`.
`-> x`	Horizontal coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
`-> y`	Vertical coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
`-> width`	Width of the control, expressed in pixels. Valid values are 1–160.
`-> height`	Height of the control, expressed in pixels. Valid values are 1–160.
`-> group`	Group ID of a push button that is part of an exclusive group. The control routines don't turn one control off automatically when another is selected. It's up to the application or a higher-level object, such as a dialog box, to manage this.
`-> leftAnchor`	`true` specifies that the left bound of this control is fixed.

Result Returns a pointer to the new graphical control. See `GraphicControlType`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlNewSliderControl`, `CtlNewControl`, `CtlValidatePointer`, `FrmRemoveObject`

CtlNewSliderControl

Purpose Create a new slider or feedback slider dynamically and install it in the specified form.

Prototype

```
SliderControlType *CtlNewSliderControl
(void **formPP, UInt16 ID, ControlStyleType style,
DmResID thumbID, DmResID backgroundID, Coord x,
Coord y, Coord width, Coord height,
UInt16 minValue, UInt16 maxValue, UInt16 pageSize,
UInt16 value)
```

Parameters

<-> `formPP`	Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the `formPP` value may change if the object moves in memory. In subsequent calls, always use the new `formPP` value returned by this function.
-> `ID`	Symbolic ID of the slider.
-> `style`	Either `sliderCtl` or `feedbackSliderCtl`. See `ControlStyleType`.
-> `thumbID`	Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider's value. To use the default thumb bitmap, pass `NULL` for this parameter.
-> `backgroundID`	Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass `NULL` for this parameter.
-> `x`	Horizontal coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.
-> `y`	Vertical coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.

-> `width`	Width of the slider, expressed in pixels. Valid values are 1–160.
-> `height`	Height of the slider, expressed in pixels. Valid values are 1–160.
-> `minValue`	Value of the slider when its thumb is all the way to the left.
-> `maxValue`	Value of the slider when its thumb is all the way to the right.
-> `pageSize`	Amount by which to increase or decrease the slider's value when the user clicks to the right or left of the thumb.
-> `value`	The initial value to display in the slider.

Result Returns a pointer to the new slider control. See `SliderControlType`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlNewGraphicControl`, `CtlNewControl`, `CtlValidatePointer`, `FrmRemoveObject`

CtlSetEnabled

Purpose Set a control as enabled or disabled. Disabled controls do not respond to the pen.

Prototype

```
void CtlSetEnabled (ControlType *controlP,
Boolean enable)
```

Parameters

-> `controlP`	Pointer to a control object. (See `ControlType`.)
-> `enable`	`true` to enable the control; `false` to disable the control.

Result Returns nothing.

Comments If you disable a visible control, the control is still displayed, and its appearance is no different from controls that do respond to the pen. You might use such a control to inform your users of some state of your application that cannot be modified.

See Also `CtlEnabled`

CtlSetGraphics

Purpose Set the bitmaps for a graphical control and redraw the control if it is visible.

Prototype

```
void CtlSetGraphics (ControlType *ctlP,
DmResID newBitmapID, DmResID newSelectedBitmapID)
```

Parameters

-> `ctlP` Pointer to a graphical control object. (See `GraphicControlType`.)

-> `newBitmapID` Resource ID of a new bitmap to display on the control, or `NULL` to use the current bitmap.

-> `newSelectedBitmapID` Resource ID of a new bitmap to display when the control is selected, or `NULL` to use the current selected bitmap.

Result Returns nothing.

Comments If `ctlP` is not a graphical control, this function immediately returns.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `GraphicControlType`

CtlSetLabel

Purpose Set the current label for the specified control object and redraw the control if it is visible.

Prototype

```
void CtlSetLabel (ControlType *controlP,
const Char *newLabel)
```

Parameters

`-> controlP`	Pointer to a control object. (See `ControlType`.)
`-> newLabel`	Pointer to the new text label. Must be a `NULL`-terminated string.

Result Returns nothing.

Comments This function resizes the width of the control to the size of the new label.

This function stores the `newLabel` pointer in the control's data structure. It doesn't make a copy of the string that is passed in. Therefore, if you use `CtlSetLabel`, you must manage the string yourself. You must ensure that it persists for as long as it is being displayed (that is, for as long as the control is displayed or until you call `CtlSetLabel` with a new string), and you must free the string after it is no longer in use (typically after the form containing the control is freed).

If you never use `CtlSetLabel`, you do not need to worry about freeing a control's label.

Make sure that `controlP` is not a graphical control or a slider control. The graphical controls and slider control structures do not contain a text label field, so attempting to set one will crash your application.

See Also `CtlGetLabel`

CtlSetSliderValues

Purpose Change a slider control's values and redraw the slider if it is visible.

Prototype

```
void CtlSetSliderValues (ControlType *ctlP,
const UInt16 *minValueP, const UInt16 *maxValueP,
const UInt16 *pageSizeP, const UInt16 *valueP)
```

Parameters

-> `ctlP`	Pointer to an inactive slider or feedback slider control. (See `SliderControlType`.)
-> `minValueP`	Pointer to a new value to use for the slider's minimum or `NULL` if you don't want to change this value.
-> `maxValueP`	Pointer to a new value to use for the slider's maximum, or `NULL` if you don't want to change this value.
-> `pageSizeP`	Pointer to a new value to use for the slider's page size, or `NULL` if you don't want to change this value.
-> `valueP`	Pointer to a new value to use for the current value, or `NULL` if you don't want to change this value.

Result Returns nothing.

Comments The control's style must be `sliderCtl` or `feedbackSliderCtl`, and it not be currently tracking the pen. If the slider is currently tracking the pen, use `CtlSetValue` to set the `value` field.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlGetSliderValues`, `SliderControlType`

CtlSetUsable

Purpose Set a control to usable or not usable.

Prototype

```
void CtlSetUsable (ControlType *controlP,
Boolean usable)
```

Parameters

`-> controlP`	Pointer to a control object. (See `ControlType`.)
`-> usable`	`true` to have the control be usable; `false` to have the control be not usable.

Result Returns nothing.

Comments A control that is not usable doesn't draw and doesn't respond to the pen.

This function doesn't usually update the control.

See Also `CtlEraseControl`, `CtlHideControl`, `CtlShowControl`

CtlSetValue

Purpose Set the current value of the specified control. If the control is visible, it's redrawn.

Prototype

```
void CtlSetValue (ControlType *controlP,
Int16 newValue)
```

Parameters

`-> controlP`	Pointer to a control object. (See `ControlType`.)
`-> newValue`	New value to set for the control. For sliders, specify a value between the slider's minimum and maximum. For graphical controls, push buttons, or check boxes, specify 0 for off, nonzero for on.

Result Returns nothing.

Comments This function works only with graphical controls, sliders, push buttons, and check boxes. If you set the value of any other type of control, the behavior is undefined.

Compatibility Sliders and graphical controls are only supported if 3.5 New Feature Set is present.

See Also `CtlGetValue`, `FrmGetControlGroupSelection`, `FrmSetControlGroupSelection`, `FrmGetControlValue`, `FrmSetControlValue`

CtlShowControl

Purpose Set a control's `usable` attribute to `true` and draw the control on the screen. This function calls `CtlDrawControl`.

Prototype

```
void CtlShowControl (ControlType *controlP)
```

Parameters `-> controlP` Pointer to a control object. (See `ControlType`.)

Result Returns nothing.

Comments If the control is already usable, this function is the functional equivalent of `CtlDrawControl`.

Sets the `visible` and the `usable` attributes to `true`. (See `ControlAttrType`.)

Don't use this function directly; instead use `FrmShowObject`, which does the same thing.

See Also `CtlHideControl`

CtlValidatePointer

Purpose Returns `true` if the specified pointer references a valid control object.

Prototype

```
Boolean CtlValidatePointer
(const ControlType *controlP)
```

Parameters -> `controlP` Pointer to a control. (See `ControlType`.)

Result Returns `true` when passed a valid pointer to a control; otherwise, returns `false`.

Comments For debugging purposes; do not include this function in commercial products. In debug builds, this function displays a dialog and waits for the debugger when an error occurs.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmValidatePtr`, `WinValidateHandle`

7

Date and Time Selector

The Palm OS® UI provides two system resources for accepting date and time input values. These resources are dialog boxes that contain UI gadgetry for entering dates and times. The Palm OS UI also provides routines to manage the interaction with these resources. This chapter describes those functions.

The API described in this chapter is declared in the header files `Day.h`, `SelDay.h`, and `SelTime.h`.

Date and Time Selections Data Structures

SelectDayType

```
typedef enum
  {
  selectDayByDay,// return d/m/y
  selectDayByWeek,// return d/m/y with d as
same day of the week
  selectDayByMonth// return d/m/y with d as
same day of the month
  } SelectDayType;
```

DaySelectorType

```
typedef struct DaySelectorType
  {
  RectangleTypebounds;
  Booleanvisible;
  UInt8 reserved1;
  Int16visibleMonth;// month actually displayed
  Int16visibleYear;// year actually displayed
```

```
DateTimeTypeselected;
SelectDayTypeselectDayBy;
UInt8 reserved2;
} DaySelectorType;
```

HMSTime

```
typedef struct {
   UInt8 hours;
   UInt8 minutes;
   UInt8 seconds;
   UInt8 reserved;
} HMSTime;
```

Date and Time Selection Functions

DayHandleEvent

Purpose Handle event in the specified control. This routine handles two types of events, `penDownEvent` and `ctlEnterEvent`.

Prototype

```
Boolean DayHandleEvent
(const DaySelectorPtr pSelector,
const EventType *pEvent)
```

Parameters

`-> pSelector`	Pointer to control object.
`-> pEvent`	Pointer to an `EventType` structure.

Result `true` if the event was handled or `false` if it was not.

Posts a `daySelectEvent` with information on whether to use the date.

Comments A date is used if the user selects a day in the visible month.

When this routine receives a penDownEvent, it checks if the pen position is within the bounds of the control object. If it is, a dayEnterEvent is added to the event queue and the routine exits.

When this routine receives a `dayEnterEvent`, it checks that the control id in the event record matches the id of the control specified. If they match, this routine will track the pen until it comes up in the bounds in which case daySelectEvent is sent.

If the pen exits the bounds a `dayExitEvent` is sent.

SelectDay

Purpose Display a form showing a date; allow user to select a different date.

Prototype

```
Boolean SelectDay
(const SelectDayType selectDayBy, Int16 *month,
Int16 *day, Int16 *year, const Char *title)
```

Parameters

`selectDayBy` The method by which the user should choose the day. Possible values are `selectDayByDay`, `selectDayByWeek`, and `selectDayByMonth`. See SelectDayType

`<-> month, day, year` Month, day, and year selected.

`-> title` String title for the dialog.

Result `true` if the OK button was pressed. If `true`, `month`, `day`, and `year` contain the new date.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `SelectDayV10`

SelectDayV10

Purpose Display a form showing a date, allow user to select a different date.

Prototype

```
Boolean SelectDay (Int16 *month, Int16 *day,
Int16 *year, const Char title)
```

Parameters

`<-> month, day, year` Month, day, and year selected. The initial values passed in these parameters must be valid.

`-> title` String title for the dialog.

Result Returns `true` if the OK button was pressed. In that case, the parameters passed are changed.

Compatibility This function corresponds to the 1.0 version of `SelectDay`.

See Also `SelectDay`

SelectOneTime

Purpose Display a form showing the time and allow the user to select a different time.

Prototype

```
Boolean SelectOneTime (Int16 *hour, Int16 *minute,
const Char *titleP)
```

Parameters

`<-> hour` The hour selected in the form.

`<-> minute` The minute selected in the form.

`-> titleP` A pointer to a string to display as the title. Doesn't change as the function executes.

Result Returns `true` if the user selects OK and `false` otherwise. If `true` is returned, the values in `hour` and `minute` have probably been changed.

Comments Use this function instead of `SelectTime` if you want to display a dialog that specifies a single point in time, not a range of time from start to end.

Compatibility Implemented only if 3.1 New Feature Set is present.

See Also `SelectTimeV33`

SelectTime

Purpose Display a form showing a start and end time. Allow the user to select a different time.

Prototype

```
Boolean SelectTime (TimeType * startTimeP,
TimeType * endTimeP, Boolean untimed, const Char *
titleP, Int16 startOfDay, Int16 endOfDay,
Int16 startOfDisplay)
```

Parameters

`<-> startTimeP, endTimeP`
Pointers to values of type `TimeType`. Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.

`-> untimed`
Pass in `true` to indicate that no time is selected. If the user sets the time to no time then `startTimeP` and `EndTimeP` are both set to the constant `noTime (-1)`.

`-> titleP`
A pointer to a string to display as the title. Doesn't change as the function executes.

`-> startOfDay`
The hour that the hour list displays at its top. To see earlier hours, the user can scroll the list up. The value must be between 0 to 12, inclusive.

`-> endOfDay`
The hour used when the "All Day" button is selected.

`-> startOfDisplay`
First hour initially visible.

Result Returns `true` if the user selects OK and `false` otherwise. If `true` is returned, the values in `hour` and `minute` have probably been changed.

Comments This version of `SelectTime` adds the `endOfDay` and `startOfDisplay` functionality.

Compatibility Implemented if 3.5 New Feature Set is present.

See Also `SelectDay`, `SelectOneTime`

SelectTimeV33

Purpose Display a form showing the time and allow the user to select a different time.

This function is obsolete and should not be used.

Prototype

```
Boolean SelectTimeV33 (TimeType *startTimeP,
TimeType *EndTimeP, Boolean untimed, Char *title,
Int16 startOfDay)
```

Parameters

`<-> startTimeP, EndTimeP`
Pointers to values of type `TimeType`. Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.

`-> untimed`
Pass in `true` to indicate that no time is selected. If the user sets the time to no time then `startTimeP` and `EndTimeP` are both set to the constant `noTime (-1)`.

`-> title`
A pointer to a string to display as the title. Doesn't change as the function executes.

`-> startOfDay` The hour that the hour list displays at its top. To see earlier hours, the user can scroll the list up. The value must be between 0 to 12, inclusive.

Result Returns `true` if the user selects OK and `false` otherwise. If `true` is returned, the values in `hour` and `minute` have probably been changed.

Comments

NOTE: Obsolete functions are provided ONLY for backward compatibility; for example, so a 1.0 application will work on 3.x OS releases. New code should not call these routines!

See Also `SelectDay`, `SelectOneTime`

8

Fields

This chapter provides the following information about field objects:

- Field Data Structures
- Field Resources
- Field Functions

The header file `Field.h` declares the API that this chapter describes. For more information on fields, see the section "Fields" in the *Palm OS Programmer's Companion*.

Field Data Structures

FieldAttrType

The `FieldAttrType` bit field defines the visible characteristics of the field. The functions `FldGetAttributes` and `FldSetAttributes` return and set these values. There are other functions that retrieve or set individual attributes defined here. Those functions are noted below.

```
typedef struct {
  UInt16 usable       :1;
  UInt16 visible      :1;
  UInt16 editable     :1;
  UInt16 singleLine   :1;
  UInt16 hasFocus     :1;
  UInt16 dynamicSize  :1;
  UInt16 insPtVisible :1;
  UInt16 dirty        :1;
  UInt16 underlined   :2;
  UInt16 justification:2;
  UInt16 autoShift    :1;
  UInt16 hasScrollBar :1;
  UInt16 numeric      :1;
```

```
} FieldAttrType;
```

Field Descriptions

`usable`	If not set, the field object is not considered part of the current interface of the application, and it doesn't appear on screen. The function `FldSetUsable` sets this value, but it is better to use `FrmShowObject`.
`visible`	Set or cleared internally when the field object is drawn or erased with `FldDrawField` or `FrmShowObject`.
`editable`	If not set, the field object doesn't accept Graffiti® input or editing commands and the insertion point cannot be positioned with the pen. The text can still be selected and copied.
`singleLine`	If set, the field is a single line of text high and doesn't expand to accommodate more text. If not set, the field can grow to multiple lines.
`hasFocus`	Set internally when the field has the current focus. The blinking insertion point appears in the field that has the current focus. Use the function `FrmSetFocus` and `FldReleaseFocus` to set this value.
`dynamicSize`	If set, the height of the field expands as characters are entered into the field and contracts as characters are deleted from the field. Note that a scrolling multiline field with `dynamicSize` set to `false` will expand the field height as necessary, but it does not contract as you delete characters.
`insPtVisible`	If set, the insertion point is scrolled into view. This attribute is set and cleared internally.

`dirty`	If set, the user has modified the field. The functions `FldDirty` and `FldSetDirty` retrieve this field's value.

`underlined`	If set each line of the field, including blank lines, is underlined. Possible values are defined by the `UnderlineModeType` defined in `Window.h`:
	
	`noUnderline`
	`grayUnderline`
	`solidUnderline`
	
	Editable text fields generally use `grayUnderline` as the value.
	
	The `solidUnderline` value is only valid for Palm OS 3.1 and higher.

`justification`	Specifies the text alignment. Possible values are `leftAlign` and `rightAlign`. (left or right justification only; `centerAlign` justification is not supported).
`autoShift`	If set, Graffiti auto-shift rules are applied.
`hasScrollBar`	If set, the field has a scrollbar. The system sends more frequent `fldChangedEvent`s so the application can adjust the height appropriately.
`numeric`	If set, only characters in the range of 0 through 9 are allowed in the field. Exactly one decimal separator (either . or ,) is also allowed per numeric field.

FieldPtr

The `FieldPtr` type defines a pointer to a `FieldType` structure.

```
typedef FieldType* FieldPtr;
```

You pass the `FieldPtr` as an argument to all field functions. You can obtain the `FieldPtr` using the function `FrmGetObjectPtr` in this way:

```
fldPtr = FrmGetObjectPtr(frm,
  FrmGetObjectIndex(frm, fldID));
```

where `fldID` is the resource ID assigned when you created the field.

FieldType

The `FieldType` structure represents a field.

```
typedef struct {
  UInt16            id;
  RectangleType     rect;
  FieldAttrType     attr;
  Char              *text;
  MemHandle         textHandle;
  LineInfoPtr       lines;
  UInt16            textLen;
  UInt16            textBlockSize;
  UInt16            maxChars;
  UInt16            selFirstPos;
  UInt16            selLastPos;
  UInt16            insPtXPos;
  UInt16            insPtYPos;
  FontID            fontID;
  UInt8             reserved;
} FieldType;
```

Your code should treat the `FieldType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Field Descriptions

`id`	ID value you specified when you created the field resource. This ID value is included as part of the event data of `fldEnterEvent`.
`rect`	Position and size of the field object. The functions `FldGetBounds`, `FrmGetObjectBounds`, `FldSetBounds`, and `FrmSetObjectBounds` retrieve and set this value.
`attr`	Field object attributes. (See `FieldAttrType`.)
`text`	Pointer to the `NULL`-terminated string that is displayed by the field object. The functions `FldGetTextPtr` and `FldSetTextPtr` retrieve and set this value (see below). Never set the value of this field directly using a function such as `StrCopy`.
`textHandle`	Handle to the stored text or to a database record containing the stored text. The functions `FldGetTextHandle` and `FldSetTextHandle` retrieve and set this value. If `textHandle` is defined, the field calculates the `text` pointer when it locks the handle. In general, you should only use `FldGetTextPtr` and `FldSetTextPtr` on text fields that aren't editable. On editable text fields, use `FldGetTextHandle` and `FldSetTextHandle`. Also note that editable text fields allocate the text handle as necessary. If a user starts typing in a field that doesn't have a text handle allocated, the field will allocate one. The field also resizes the text's memory block as necessary when the user adds more text.

`lines`	Pointer to an array of `LineInfoType` structures. There is one entry in this array for each visible line of the text. (See `LineInfoType`.) The field code maintains this array internally; you should never change the `lines` array yourself.
`textLen`	Length in bytes of the string currently displayed by the field object; the null terminator is excluded. You can retrieve this value with `FldGetTextLength`.
`textBlockSize`	Allocated size of the memory block that holds the field object's text string. You can retrieve this value with `FldGetTextAllocatedSize`. Fields allocate memory for the field text as needed, several bytes at a time. Note that `textBlockSize` may be different from the size of the chunk pointed to by `textHandle`. The `textHandle` may point to a database record that contains, in part, the text displayed by the field. If you called `MemHandleSize` on such a `textHandle`, the number returned may be greater than `textBlockSize`.
`maxChars`	Maximum number of bytes the field object accepts. The functions `FldGetMaxChars` and `FldSetMaxChars` retrieve and set this value. Note the difference between `textLen`, `textBlockSize`, and `maxChars`. `textLen` is the size of the characters that `text` actually holds. `textBlockSize` is the amount of memory currently allocated for the text (which must be greater than or equal to `textLen`), and `maxChars` sets the maximum value that `textBlockSize` and `textLen` can expand to.

For example, if you've created a text field for users to enter their first names in, you might specify that the maximum length of this field is 20 characters. If a user enters "John" in this field, `textLen` is 4, `textBlockSize` is 16, and `maxChars` is 20.

`selFirstPos` Starting character offset in bytes of the current selection. Use `FldGetSelection` and `FldSetSelection` to retrieve and set this value and the `selLastPos` value.

`selLastPos` Ending character offset in bytes of the current selection. When `selFirstPos` equals `selLastPos`, there is no selection.

`insPtXPos` Horizontal location of the insertion point, given as the offset in bytes into the line indicated by `insPtYPos`. The functions `FldGetInsPtPosition` and `FldSetInsPtPosition` retrieve and set this value.

`insPtYPos` Vertical location of the insertion point, given as the display line where the insertion point is positioned. The first display line is zero. The first display line may be different from the first line of text in the field if the field has been scrolled.

`fontID` Font ID for the field. See `Font.h` for more information. The functions `FldGetFont` and `FldSetFont` retrieve and set this value.

`reserved` Reserved for future use.

LineInfoPtr

The `LineInfoPtr` type defines a pointer to the `LineInfoType`.

```
typedef LineInfoType* LineInfoPtr;
```

LineInfoType

The `LineInfoType` structure defines an element in the field's `lines` array. The `lines` array contains the field's word wrapping information. There is one element in the array per visible line in the field, including visible lines that contain no text. The field code maintains this array internally; you should never change the `lines` array yourself.

The functions `FldCalcFieldHeight`, `FldGetVisibleLines`, `FldRecalculateField`, and `FldGetNumberOfBlankLines` retrieve or set information in this structure. The scrolling functions `FldGetScrollPosition`, `FldGetScrollValues`, `FldScrollField`, and `FldSetScrollPosition` also retrieve or set information in this structure.

```
typedef struct {
  UInt16      start;
  UInt16      length;
} LineInfoType;
```

Field Descriptions

`start` The byte offset into the `FieldType`'s `text` field of the first character displayed by this line. If the line is blank, `start` is equal to `textLen` and `length` is 0.

`length` The length in bytes of the portion of the string displayed on this line. If the line is blank, the length is 0.

Field Resources

The Field Resource (tFLD) represents a field on screen.

Field Functions

FldCalcFieldHeight

Purpose Determine the height of a field for a string.

Prototype
```
UInt16 FldCalcFieldHeight (const Char* chars,
UInt16 maxWidth)
```

Parameters

-> `chars`	Pointer to a null-terminated string.
-> `maxWidth`	Maximum line width in pixels.

Result Returns the total number of lines needed to draw the string passed.

Comments The width of a field is contained in the `rect` member of the `FieldType` structure. You can retrieve this value in the following way:

```
FrmGetObjectBounds(frm,
  FrmGetObjectIndex(frm, fldID),
  &myRect);
fieldWidth = myRect.extent.x;
FldCalcFieldHeight(myString, fieldWidth);
```

See Also `FldWordWrap`

FldCompactText

Purpose Compact the memory block that contains the field's text to release any unused space.

Prototype
```
void FldCompactText (FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns nothing.

Comments As characters are added to the field's text, the block that contains the text is grown. The block is expanded several bytes at a time so that it doesn't have to expand each time a character is added. This expansion may result in some unused space in the text block.

Applications should call this function on field objects that edit data records in place before the field is unlocked, or at any other time when a compact field is desirable; for example, before writing to the storage heap.

See Also `FldGetTextAllocatedSize`, `FldSetTextAllocatedSize`

FldCopy

Purpose Copy the current selection to the text clipboard.

Prototype

```
void FldCopy (const FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns nothing.

Comments This function leaves the current selection highlighted.

This function replaces anything previously in the text clipboard if there is text to copy. If no text is selected, the function beeps and the clipboard remains intact.

See Also `FldCut`, `FldPaste`

FldCut

Purpose Copy the current selection to the text clipboard, delete the selection from the field, and redraw the field.

Prototype
```
void FldCut (FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns nothing.

Comments If text is selected, the text is removed from the field, the field's dirty attribute is set, and anything previously in the text clipboard is replaced by the selected text.

If there is no selection or if the field is not editable, this function beeps.

See Also `FldCopy`, `FldPaste`, `FldUndo`

FldDelete

Purpose Delete the specified range of characters from the field and redraw the field.

Prototype
```
void FldDelete (FieldType* fldP, UInt16 start,
UInt16 end)
```

Parameters

-> `fldP`	Pointer to the field object (`FieldType` structure) to delete from.
-> `start`	The beginning of the range of characters to delete given as a valid byte offset into the field's `text` string.

-> `end`	The end of the range of characters to delete given as a valid byte offset into the field's `text` string. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.

Result Returns nothing.

Comments This function deletes all characters from the starting offset up to the ending offset and sets the field's dirty attribute. It does not delete the character at the ending offset.

If `start` or `end` point to an intra-character boundary, `FldDelete` attempts to move the offset backward, toward the beginning of the text, until the offset points to an inter-character boundary (i.e., the start of a character).

`FldDelete` posts a `fldChangedEvent` to the event queue. If you call this function repeatedly, you may overflow the event queue with `fldChangedEvents`. An alternative is to remove the text handle from the field, change the text, and then set the field's handle again. See `FldGetTextHandle` for a code example.

See Also `FldInsert`, `FldEraseField`, `TxtCharBounds`

FldDirty

Purpose Return `true` if the field has been modified since the text value was set.

Prototype

```
Boolean FldDirty (const FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns `true` if the field has been modified either by the user or through calls to certain functions such as `FldInsert` and `FldDelete`, `false` if the field has not been modified.

See Also `FldSetDirty`, `FieldAttrType`

FldDrawField

Purpose Draw the text of the field.

Prototype `void FldDrawField (FieldType* fldP)`

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns nothing.

Comments The field's `usable` attribute must be `true` or the field won't be drawn.

This function doesn't erase the area behind the field before drawing.

If the field has the focus, the blinking insertion point is displayed in the field.

See Also `FldEraseField`

FldEraseField

Purpose Erase the text of a field and turn off the insertion point if it's in the field.

Prototype `void FldEraseField (FieldType* fldP)`

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns nothing.

Comments You rarely need to call this function directly. Instead, use `FrmHideObject`, which calls `FldEraseField` for you.

This function visibly erases the field from the display, but it doesn't modify the contents of the field or free the memory associated with it.

If the field has the focus, the blinking insertion point is turned off.

This function sets the `visible` attribute to `false`. (See `FieldAttrType`.)

See Also `FldDrawField`

FldFreeMemory

Purpose Release the handle-based memory allocated to the field's text and the associated word-wrapping information.

Prototype

```
void FldFreeMemory (FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns nothing. May raise a fatal error message if the text associated with the field is actually a record in a database.

Comments This function releases

- The memory allocated to the text of a field—the memory block that the `textHandle` member of the `FieldType` data structure points to.

  If the field's `textHandle` is `NULL` but there is a `text` string associated with that field (which is often the case with noneditable text fields), the text string is not freed.
- The memory allocated to hold the word-wrapping information—the memory block that the `lines` member of the `FieldType` data structure points to.

This function doesn't affect the display of the field. Fields allocate memory for the text string as needed, so it is not an error to call this function while the field is still displayed. That is, if `text` is `NULL` and the user starts typing in the field, the field simply allocates memory for text and continues.

FldGetAttributes

Purpose Return the attributes of a field.

Prototype
```
void FldGetAttributes (const FieldType* fldP,
FieldAttrPtr attrP)
```

Parameters

-> `fldP`	Pointer to a `FieldType` structure.
<- `attrP`	Pointer to the `FieldAttrType` structure.

Result Returns the field's attributes in the `attrP` parameter.

See Also `FldSetAttributes`

FldGetBounds

Purpose Return the current bounds of a field.

Prototype
```
void FldGetBounds (const FieldType* fldP,
RectanglePtr rect)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
<- `rect`	Pointer to a `RectangleType` structure.

Result Returns nothing. Stores the field's bounds in the `RectangleType` structure reference by `rect`.

Comments Returns the `rect` field of the `FieldType` structure.

See Also `FldSetBounds`, `FrmGetObjectBounds`

FldGetFont

Purpose Return the ID of the font used to draw the text of a field.

Prototype
```
FontID FldGetFont (const FieldType* fldP)
```

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns the ID of the font.

See Also `FldSetFont`

FldGetInsPtPosition

Purpose Return the insertion point position within the string.

Prototype
```
UInt16 FldGetInsPtPosition (const FieldType* fldP)
```

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns the byte offset of the insertion point.

Comments The insertion point is to the left of the byte offset that this function returns. That is, if this function returns 0, the insertion point is to the left of the first character in the string. In multiline fields, line feeds are counted as a single character in the string, and the byte offset after the line feed character is the beginning of the next line.

See Also `FldSetInsPtPosition`

FldGetMaxChars

Purpose Return the maximum number of bytes the field accepts.

Prototype
```
UInt16 FldGetMaxChars (const FieldType* fldP)
```

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns the maximum length in bytes of characters the user is allowed to enter. This is the `maxChars` field in `FieldType`.

See Also `FldSetMaxChars`

FldGetNumberOfBlankLines

Purpose Return the number of blank lines that are displayed at the bottom of a field.

Prototype
```
UInt16 FldGetNumberOfBlankLines
(const FieldType* fldP)
```

Parameters -> `fldP` Pointer to a `FieldType` structure.

Result Returns the number of blank lines visible.

Comments This routine is useful for updating a scroll bar after characters have been removed from the text in a field. See the `NoteViewScroll` function in the Address sample application for an example.

Compatibility Implemented only if 2.0 New Feature Set is present.

FldGetScrollPosition

Purpose Return the offset of the first character in the first visible line of a field.

Prototype

```
UInt16 FldGetScrollPosition
(const FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns the offset of the first visible character.

See Also `FldSetScrollPosition`, `LineInfoType`

FldGetScrollValues

Purpose Return the values necessary to update a scroll bar.

Prototype

```
void FldGetScrollValues (const FieldType* fldP,
UInt16* scrollPosP, UInt16* textHeightP,
UInt16* fieldHeightP)
```

Parameters

-> `fldP`	Pointer to a `FieldType` structure.
<- `scrollPosP`	The line of text that is the topmost visible line. Line numbering starts with 0.
<-`textHeightP`	The number of lines needed to display the field's text, given the width of the field.
<-`fieldHeightP`	The number of visible lines in the field.

Result Returns nothing. Stores the position, text height, and field height in the parameters passed in.

Comments Use the values returned by this function to calculate the values you send to `SclSetScrollBar` to update the scroll bar. For example:

```
FldGetScrollValues (fldP, &scrollPos,
  &textHeight,  &fieldHeight);
```

```
if (textHeight > fieldHeight)
  maxValue = textHeight - fieldHeight;
else if (scrollPos)
  maxValue = scrollPos;
else
  maxValue = 0;

SclSetScrollBar (bar, scrollPos, 0, maxValue,
  fieldHeight-1);
}
```

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FldSetScrollPosition`

FldGetSelection

Purpose Return the current selection of a field.

Prototype

```
void FldGetSelection (const FieldType* fldP,
UInt16* startPosition, UInt16* endPosition)
```

Parameters

-> `fldP` Pointer to a field object (`FieldType` structure).

<- `startPosition` Pointer to the start of the selected characters range, given as the byte offset into the field's text.

<- `endPosition` Pointer to end of the selected characters range given as the byte offset into the field's text.

Result Returns the starting and ending byte offsets in `startPosition` and `endPosition`.

Comments The first character in a field is at offset zero.

If the user has selected the first five characters of a field, `startPosition` will contain the value 0 and `endPosition` the value 5, assuming all characters are a single byte long.

See Also `FldSetSelection`

FldGetTextAllocatedSize

Purpose Return the number of bytes allocated to hold the field's text string. Don't confuse this number with the actual length of the text string displayed in the field.

Prototype

```
UInt16 FldGetTextAllocatedSize
(const FieldType* fldP)
```

Parameters -> `fldP` Pointer to a field object.

Result Returns the number of bytes allocated for the field's text. This is the `textBlockSize` field in `FieldType`.

See Also `FldSetTextAllocatedSize`

FldGetTextHandle

Purpose Return a handle to the block that contains the text string of a field.

Prototype

```
MemHandle FldGetTextHandle (const FieldType* fldP)
```

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns the handle to the text string of a field or `NULL` if no handle has been allocated for the field pointer.

Comments The handle returned by this function is not necessarily the handle to the start of the string. If you've used `FldSetText` to set the field's text to a string that is part of a database record, the text handle points to the start of that record. You'll need to compute the offset from the start of the record to the start of the string. You can either

store the offset that you passed to `FldSetText` or you can compute the offset by performing pointer arithmetic on the pointer you get by locking this handle and the pointer returned by `FldGetTextPtr`.

If you are obtaining the text handle so that you can edit the field's text, you must remove the handle from the field before you do so. If you change the text while it is being used by a field, the field's internal structures specifying the text length, allocated size, and word wrapping information can become out of sync. To avoid this problem, remove the text handle from the field, change the text, and then set the field's text handle again. For example:

```
/* Get the handle for the string and unlock */
/* it by removing it from the field. */
textH = FldGetTextHandle(fldP);
FldSetTextHandle (fldP, NULL);

/* Insert code that modifies the string here.*/
/* The basic steps are: */
/* resize the chunk if necessary,*/
/* lock the chunk, write to it, and then */
/* unlock the chunk. If the text is in a */
/* database record, use Data Manager calls. */

/* Update the text in the field. */
FldSetTextHandle (fldP, textH);
FldDrawField(fldP);
```

See Also `FldSetTextHandle`, `FldGetTextPtr`

FldGetTextHeight

Purpose Return the height in pixels of the number of lines that are not empty.

Prototype
```
UInt16 FldGetTextHeight (const FieldType* fldP)
```

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns the height in pixels of the number of lines that are not empty.

Comments Empty lines are all of the lines in the field following the last byte of text. Note that lines that contain only a linefeed are not empty.

See Also `FldCalcFieldHeight`

FldGetTextLength

Purpose Return the length in bytes of the field's text.

Prototype
```
UInt16 FldGetTextLength (const FieldType* fldP)
```

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns the length in bytes of a field's text, not including the terminating null character. This is the `textLen` field of `FieldType`.

FldGetTextPtr

Purpose Return a locked pointer to the field's text string.

Prototype
```
Char* FldGetTextPtr (FieldType* fldP)
```

Parameters -> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns a locked pointer to the field's text string or `NULL` if the field is empty.

Comments The pointer returned by this function can become invalid if the user edits the text after you obtain the pointer.

Do not modify the contents of the pointer yourself. If you change the text while it is being used by a field, the field's internal structures specifying the text length, allocated size, and word wrapping information can become out of sync. To avoid this problem, follow the instructions given under `FldGetTextHandle`.

See Also `FldSetTextPtr`, `FldGetTextHandle`

FldGetVisibleLines

Purpose Return the number of lines that can be displayed within the visible bounds of the field.

Prototype

```
UInt16 FldGetVisibleLines (const FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns the number of lines the field displays. (This is the size of the `lines` array in the `FieldType` structure.)

See Also `FldGetNumberOfBlankLines`, `FldCalcFieldHeight`

FldGrabFocus

Purpose Turn the insertion point on (if the specified field is visible) and position the blinking insertion point in the field.

Prototype

```
void FldGrabFocus (FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns nothing.

Comments You rarely need to call this function directly. Instead, use `FrmSetFocus`, which calls `FldGrabFocus` for you.

One instance where you need to call `FldGrabFocus` directly is to programmatically set the focus in a field that is contained in a table cell.

This function sets the field attribute `hasFocus` to `true`. (See `FieldAttrType`.)

See Also `FrmSetFocus`, `FldReleaseFocus`

FldHandleEvent

Purpose Handles events that affect the field, including the following: `keyDownEvent`, `penDownEvent`, and `fldEnterEvent`.

Prototype

```
Boolean FldHandleEvent (FieldType* fldP,
EventType* eventP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `eventP`	Pointer to an event (`EventType` data structure).

Result Returns `true` if the event was handled.

Comments When a `keyDownEvent` occurs in an editable text field, the keystroke appears in the field if it's a printable character or manipulates the insertion point if it's a "movement" character. The field is automatically updated.

When a `penDownEvent` occurs, the field sends a `fldEnterEvent` to the event queue.

When a `fldEnterEvent` occurs, the field grabs the focus. If the user has tapped twice in the current location, the word at that location is selected. If the user has tapped three times, the entire line is selected. Otherwise, the insertion point is placed in the specified position.

When a `menuCmdBarOpenEvent` occurs, the field adds paste, copy, cut, and undo buttons to the command toolbar. These buttons are only added if they make sense in the current context. That is, the cut button is only added if the field is editable, the paste button is only

added if there is text on the clipboard and the field is editable, and the undo button is only added if there is an action to undo.

If the event alters the contents of the field, this function visually updates the field.

This function doesn't handle any events if the field is not editable or usable.

Compatibility Double-tapping to select a word and triple-tapping to select a line are only supported if 3.5 New Feature Set is present.

`FldHandleEvent` only handles the `menuCmdBarOpenEvent` if 3.5 New Feature Set is present.

FldInsert

Purpose Replace the current selection if any with the specified string and redraw the field.

Prototype

```
Boolean FldInsert (FieldType* fldP,
const Char* insertChars, UInt16 insertLen)
```

Parameters

-> `fldP`	Pointer to the field object (`FieldType` structure) to insert to.
-> `insertChars`	Text string to be inserted.
-> `insertLen`	Length in bytes of the text string to be inserted, not counting the trailing null character.

Result Returns `true` if string was successfully inserted. Returns `false` if:

- The `insertLen` parameter is 0.
- The field is not editable.
- Adding the text would exceed the field's size limit (the `maxChars` value).
- More memory must be allocated for the field, and the allocation fails.

Comments If there is no current selection, the string passed is inserted at the position of the insertion point.

This function sets the field's dirty attribute and posts a `fldChangedEvent` to the event queue. If you call this function repeatedly, you may overflow the event queue with `fldChangedEvents`. An alternative is to remove the text handle from the field, change the text, and then set the field's handle again. See `FldGetTextHandle` for a code example.

See Also `FldPaste`, `FldDelete`, `FldCut`, `FldCopy`

FldMakeFullyVisible

Purpose Cause a dynamically resizable field to expand its height to make its text fully visible.

Prototype

```
Boolean FldMakeFullyVisible (FieldType* fldP)
```

Parameters

-> `fldP` Pointer to a field object (`FieldType` structure).

Result Returns `true` if the field is dynamically resizable and was not fully visible; `false` otherwise.

Comments Use this function on a field whose `dynamicSize` attribute is `true` (see `FieldAttrType`).

This function does not actually resize the field. Instead, it computes how big the field should be to be fully visible and then posts this information to the event queue in a `fldHeightChangedEvent`.

If the field is contained in a table, the table's code handles the `fldHeightChangedEvent`. If the field is directly on a form, your application code should handle the `fldHeightChangedEvent` itself. The form code does not handle the event for you. Note that the constant `maxFieldLines` defines the maximum number of lines a field can expand to if the field is using the standard font.

See Also `TblHandleEvent`

FldNewField

Purpose Create a new field object dynamically and install it in the specified form.

Prototype

```
FieldType *FldNewField (void **formPP, UInt16 id,
Coord x, Coord y, Coord width, Coord height,
FontID font, UInt32 maxChars, Boolean editable,
Boolean underlined, Boolean singleLine,
Boolean dynamicSize,
JustificationType justification,
Boolean autoShift, Boolean hasScrollBar,
Boolean numeric)
```

Parameters

<-> `formPP`	Pointer to the pointer to the form in which the new field is installed. This value is not a handle; that is, the old form pointer value is not necessarily valid after this function returns. In subsequent calls, always use the new form pointer value returned by this function.
-> `id`	Symbolic ID of the field, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
-> `x`	Horizontal coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.
-> `y`	Vertical coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.
-> `width`	Width of the field, expressed in pixels.
-> `height`	Height of the field, expressed in pixels.
-> `font`	Font to use to draw the field's text.
-> `maxChars`	Maximum number of bytes held by the field this function creates.

-> `editable`	Pass `true` to create a field in which the user can edit text. Pass `false` to create a field that cannot be edited.
-> `underlined`	Pass `noUnderline` for no underline, or `grayUnderline` to have the field underline the text it displays. On Palm OS® version 3.1 and higher, pass `solidUnderline` to use a solid underline instead of a dotted underline.
-> `singleLine`	Pass `true` to create a field that can display only a single line of text.
-> `dynamicSize`	Pass `true` to create a field that resizes dynamically according to the amount of text it displays.
-> `justification`	Pass either of the values `leftAlign` or `rightAlign` to specify left justification or right justification, respectively. The `centerAlign` value is not supported.
-> `autoShift`	Pass `true` to specify the use of Palm OS 2.0 (and later) auto-shift rules.
-> `hasScrollBar`	Pass `true` to attach a scroll bar control to the field this function creates.
-> `numeric`	Pass `true` to specify that only characters in the range of 0 through 9 are allowed in the field.

Result Returns a pointer to the new field object or `NULL` if there wasn't enough memory to create the field. Out of memory situations could be caused by memory fragmentation.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmValidatePtr`, `WinValidateHandle`, `CtlValidatePointer`, `FrmRemoveObject`

FldPaste

Purpose Replace the current selection in the field, if any, with the contents of the text clipboard.

Prototype

```
void FldPaste (FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns nothing

Comments The function performs these actions:

- Scrolls the field, if necessary, so the insertion point is visible.
- Inserts the clipboard text at the position of the insertion point if there is no current selection.
- Positions the insertion point after the last character inserted.
- Doesn't delete the current selection if there is no text in the clipboard.

See Also `FldInsert`, `FldDelete`, `FldCut`, `FldCopy` `FldUndo`

FldRecalculateField

Purpose Update the structure that contains the word-wrapping information for each visible line.

Prototype

```
void FldRecalculateField (FieldType* fldP,
Boolean redraw)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `redraw`	If `true`, redraws the field. Currently, this parameter must be set to `true` to update the word-wrapping information.

Result Returns nothing.

Comments If necessary, this function reallocates the memory block that contains the displayed lines information, the `LineInfoType` structure pointed to by the `lines` member of the `FieldType` data structure.

Call this function if the field's data structure is modified in a way that invalidates the visual appearance of the field (for example, if you update a field's text with `FldSetTextPtr`). However, many of the field functions, such as `FldSetTextHandle`, `FldInsert`, and `FldDelete`, recalculate the word-wrapping information for you.

FldReleaseFocus

Purpose Turn the blinking insertion point off if the field is visible and has the current focus, reset the Graffiti state, and reset the undo state.

Prototype

```
void FldReleaseFocus (FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

Result Returns nothing.

Comments This function sets the field attribute `hasFocus` to `false`. (See `FieldAttrType`.)

Usually, you don't need to call this function. If the field is in a form or in a table that doesn't use custom drawing functions, the field code releases the focus for you when the focus changes to some other control. If your field is in any other type of object, such as a table that uses custom drawing functions or a gadget, you must call `FldReleaseFocus` when the focus moves away from the field.

See Also `FldGrabFocus`

FldScrollable

Purpose Return `true` if the field is scrollable in the specified direction.

Prototype

```
Boolean FldScrollable (const FieldType* fldP,
WinDirectionType direction)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `direction`	The direction to test. `DirectionType` is defined in `Window.h`. It is an enum defining the constants `up` and `down`.

Result Returns `true` if the field is scrollable in the specified direction; `false` otherwise.

See Also `FldScrollField`

FldScrollField

Purpose Scroll a field up or down by the number of lines specified.

Prototype

```
void FldScrollField (FieldType* fldP,
UInt16 linesToScroll, WinDirectionType direction)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `linesToScroll`	Number of lines to scroll.
-> `direction`	The direction to scroll. `DirectionType` is defined in `Window.h`. It is an enum defining the constants `up` and `down`.

Result Returns nothing.

Comments This function can't scroll horizontally, that is, right or left.

The field object is redrawn if it's scrolled; however, the scrollbar is not updated. Use `SclSetScrollBar` to update the scrollbar. For example:

```
FldScrollField (fldP, linesToScroll,
direction);

// Update the scroll bar.
SclGetScrollBar (bar, &value, &min, &max,
  &pageSize);

if (direction == up)
  value -= linesToScroll;
else
  value += linesToScroll;

SclSetScrollBar (bar, value, min, max,
  pageSize);
```

If the field is not scrollable in the direction indicated, this function returns without performing any work. You can use `FldScrollable` before calling this function to see if the field can be scrolled.

See Also `FldScrollable`, `FldSetScrollPosition`

FldSendChangeNotification

Purpose Send a `fldChangedEvent` to the event queue.

Prototype

```
void FldSendChangeNotification
(const FieldType* fldP)
```

Parameters

-> `fldP`	Pointer to a field object.

Result Returns nothing.

Comments This function is used internally by the field code. You normally never call it in application code.

FldSendHeightChangeNotification

Purpose Send a `fldHeightChangedEvent` to the event queue.

Prototype

```
void FldSendHeightChangeNotification
(const FieldType* fldP, UInt16 pos,
Int16 numLines)
```

Parameters

-> `fldP`	Pointer to a field object.
-> `pos`	Character position of the insertion point.
-> `numLines`	New number of lines in the field.

Result Returns nothing.

Comments This function is used internally by the field code. You normally never call it in application code.

FldSetAttributes

Purpose Set the attributes of a field.

Prototype

```
void FldSetAttributes (FieldType* fldP,
const FieldAttrPtr attrP)
```

Parameters

-> `fldP`	Pointer to a `FieldType` structure.
-> `attrP`	Pointer to the attributes.

Result Returns nothing.

Comments This function does not do anything to make the new attribute values take effect. For example, if you use this function to change the value of the underline attribute, you won't see its effect until you call `FldDrawField`.

You usually do not have to modify field attributes at runtime, so you rarely need to call this function.

See Also FldGetAttributes, FieldAttrType

FldSetBounds

Purpose Change the position or size of a field.

Prototype

```
void FldSetBounds (FieldType* fldP,
const RectangleType* rP)
```

Parameters

-> fldP	Pointer to a field object (FieldType structure).
-> rP	Pointer to a RectangleType structure that contains the new bounds of the display.

Result Returns nothing. May raise a fatal error message if the memory block that contains the word-wrapping information needs to be resized and there is not enough space to do so.

Comments If the field is visible, the field is redrawn within its new bounds.

NOTE: You can change the height or location of the field while it's visible, but do not change the width.

The memory block that contains the word-wrapping information (see LineInfoType) will be resized if the number of visible lines is changed. The insertion point is assumed to be off when this routine is called.

Make sure that rect is at least as tall as a single line in the current font. (You can determine this value by calling FntLineHeight.) If it's not, results are unpredictable.

See Also FldGetBounds, FrmSetObjectBounds

FldSetDirty

Purpose Set whether the field has been modified.

Prototype
```
void FldSetDirty (FieldType* fldP, Boolean dirty)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `dirty`	`true` if the text is modified.

Result Returns nothing.

Comments You typically call this function when you want to clear the dirty attribute. The dirty attribute is set when the user enters or deletes text in the field. It is also set by certain field functions, such as `FldInsert` and `FldDelete`.

See Also `FldDirty`

FldSetFont

Purpose Set the font used by the field, update the word-wrapping information, and draw the field if the field is visible.

Prototype
```
void FldSetFont (FieldType* fldP, FontID fontID)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `fontID`	ID of new font.

Result Returns nothing.

See Also `FldGetFont`, `FieldAttrType`

FldSetInsertionPoint

Purpose Set the location of the insertion point based on a specified string position.

Prototype
```
void FldSetInsertionPoint (FieldType* fldP,
UInt16 pos)
```

Parameters

-> `fldP`	Pointer to a `FieldType` structure.
-> `pos`	New location of the insertion point, given as a valid offset in bytes into the field's text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

Result Nothing.

Comments This routine differs from `FldSetInsPtPosition` in that it doesn't make the character position visible. `FldSetInsertionPoint` also doesn't make the field the current focus of input if it was not already.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `TxtCharBounds`

FldSetInsPtPosition

Purpose Set the location of the insertion point for a given string position.

Prototype
```
void FldSetInsPtPosition (FieldType* fldP,
UInt16 pos)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).

-> `pos`	New location of the insertion point, given as a valid offset in bytes into the field's text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

Result Returns nothing.

Comments If the position is beyond the visible text, the field is scrolled until the position is visible.

See Also `FldGetInsPtPosition`, `TxtCharBounds`

FldSetMaxChars

Purpose Set the maximum number of bytes the field accepts (the `maxChars` value).

Prototype

```
void FldSetMaxChars (FieldType* fldP,
UInt16 maxChars)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `maxChars`	Maximum size in bytes of the characters the user may enter. You may specify any value up to `maxFieldTextLen`.

Result Returns nothing.

Comments Line feed characters are counted when the length of characters is determined.

See Also `FldGetMaxChars`

FldSetScrollPosition

Purpose Scroll the field such that the character at the indicated offset is the first character on the first visible line. Redraw the field if necessary.

Prototype

```
void FldSetScrollPosition (FieldType* fldP,
UInt16 pos)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `pos`	Byte offset into the field's text string of first character to be made visible. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

Result Returns nothing.

Comments This function scrolls the field but does not update the field's scrollbar. You should update the scrollbar after calling this function. To do so, first call `FldGetScrollValues` to determine the values to use, and then call `SclSetScrollBar`.

See Also `FldGetScrollPosition`, `FldScrollField`, `TxtCharBounds`

FldSetSelection

Purpose Set the current selection in a field and highlight the selection if the field is visible.

Prototype

```
void FldSetSelection (FieldType* fldP,
UInt16 startPosition, UInt16 endPosition)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `startPosition`	Starting offset of the character range to highlight, given as a byte offset into the field's text.

-> `endPosition`	Ending offset of the character range to highlight. The ending offset should be greater than or equal to the starting offset. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.

Result Returns nothing.

Comments To cancel a selection, set both `startPosition` and `endPosition` to the same value. If `startPosition` equals `endPosition`, then the current selection is unhighlighted.

If either `startPosition` or `endPosition` point to an intra-character boundary, `FldSetSelection` attempts to move that offset backward, toward the beginning of the string, until the offset points to an inter-character boundary (i.e., the start of a character).

See Also `TxtCharBounds`

FldSetText

Purpose Set the text value of the field without updating the display.

Prototype

```
void FldSetText (FieldType* fldP,
MemHandle textHandle, UInt16 offset, UInt16 size)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `textHandle`	Unlocked handle of a block containing a null-terminated text string. Pass `NULL` for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.
-> `offset`	Offset from start of block to start of the text string.

-> `size`	Allocated size of text string, **not** the string length.

Result Returns nothing.

Comments This function allows applications to perform editing in place in memory. You can use it to point the field to a string in a database record so that you can edit that string directly using field routines.

The handle that you pass to this function is assumed to contain a null-terminated string starting at `offset` bytes in the memory chunk. The string should be between 0 and `size` - 1 bytes in length. The field does not make a copy of the memory chunk or the string data; instead, it stores the handle to the record in its structure.

`FldSetText` updates the word-wrapping information and places the insertion point after the last visible character, but it does not update the display. You must call `FldDrawField` after calling this function to update the display.

`FldSetText` increments the lock count for `textHandle` and decrements the lock count of its previous text handle (if any).

Because `FldSetText` (and `FldSetTextHandle`) may be used to edit database records, they do not free the memory associated with the previous text handle. If the previous text handle points to a string on the dynamic heap and you want to free it, use `FldGetTextHandle` to obtain the handle before using `FldSetText` and then free that handle after using `FldSetText`. (See `FldSetTextHandle` for a code example.)

If the field points to a database record, you want the memory associated with the text handle to persist; however, this memory and all other memory associated with the field is freed when the field itself is freed, which happens when the form is closed. If you don't want the memory associated with the text handle freed when the field is freed, use `FldSetText` and pass `NULL` for the text handle immediately before the form is closed. Passing `NULL` removes the association between the field and the text handle that you want retained. That text handle is unlocked as a result of the

`FldSetText` call, and when the field is freed, there is no text handle to free with it.

See Also `FldSetTextPtr`, `FldSetTextHandle`

FldSetTextAllocatedSize

Purpose Set the number of bytes allocated to hold the field's text string. Don't confuse this with the actual length of the text string displayed in the field.

Prototype

```
void FldSetTextAllocatedSize (FieldType* fldP,
UInt16 allocatedSize)
```

Parameters

-> `fldP` Pointer to a field object (`FieldType` structure).

-> `allocatedSize` Number of bytes to allocate for the text.

Result Returns nothing.

Comments This function generally is not used. It does not resize the field's allocated memory for the text string; it merely sets the `textBlockSize` field of the `FieldType` structure. The value of this field is computed and maintained internally by the field, so you should not have to call `FldSetTextAllocatedSize` directly.

See Also `FldGetTextAllocatedSize`, `FldCompactText`

FldSetTextHandle

Purpose Set the text value of a field to the string associated with the specified handle. Does not update the display.

Prototype

```
void FldSetTextHandle (FieldType* fldP,
MemHandle textHandle)
```

Parameters

-> `fldP` Pointer to a field object (`FieldType` structure).

-> `textHandle`	Unlocked handle of a field's text string. Pass `NULL` for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.

Result Returns nothing.

Comments This function differs from `FldSetText` in that it uses the entire memory chunk pointed to by `textHandle` for the string. In fact, this function simply calls `FldSetText` with an offset of 0 and a size equal to the entire length of the memory chunk. Use it to have the field edit a string in a database record if the entire record consists of that string, or use it to have the field edit a string in the dynamic heap.

`FldSetTextHandle` updates the word-wrapping information and places the insertion point after the last visible character, but it does not update the display. You must call `FldDrawField` after calling this function to update the display.

`FldSetTextHandle` increments the lock count for `textHandle` and decrements the lock count of its previous text handle (if any).

Because `FldSetTextHandle` (and `FldSetText`) may be used to edit database records, they do not free the memory associated with the previous text handle. If the previous text handle points to a string on the dynamic heap and you want to free it, use `FldGetTextHandle` to obtain the handle before using `FldSetText` and then free that handle after using `FldSetText`. For example:

```
/* get the old text handle */
oldTxtH = FldGetTextHandle(fldP);

/* change the text and update the display */
FldSetTextHandle(fldP, txtH);
FldDrawField(fldP);

/* free the old text handle */
if (oldTxtH != NULL)
  MemHandleFree(oldTxtH);
```

If the field points to a database record, you want the memory associated with the text handle to persist; however, this memory and all other memory associated with the field is freed when the field itself is freed, which happens when the form is closed. If you don't want the memory associated with the text handle freed when the field is freed, use `FldSetTextHandle` and pass `NULL` for the text handle immediately before the form is closed. Passing `NULL` removes the association between the field and the text handle that you want retained. That text handle is unlocked as a result of the `FldSetTextHandle` call, and when the field is freed, there is no text handle to free with it.

See Also `FldSetTextPtr`, `FldSetText`

FldSetTextPtr

Purpose Set a noneditable field's text to point to the specified text string.

Prototype

```
void FldSetTextPtr (FieldType* fldP, Char* textP)
```

Parameters

-> `fldP`	Pointer to a field object (`FieldType` structure).
-> `textP`	Pointer to a null-terminated string.

Result Returns nothing. May display an error message if passed an editable text field.

Comments Do not call `FldSetTextPtr` with an editable text field. Instead, call `FldSetTextHandle` for editable text fields. `FldSetTextPtr` is intended for displaying noneditable text in the user interface.

If the field has more than one line, use `FldRecalculateField` to recalculate the word wrapping.

This function does **not** visually update the field. Use `FldDrawField` to do so.

The field never frees the string that you pass to this function, even when the field itself is freed. You must free the string yourself. Before you free the string, make sure the field is not still displaying it. Set the field's string pointer to some other string or call

`FldSetTextPtr(fldP, NULL)` before freeing a string you have passed using this function.

See Also `FldSetTextHandle`, `FldGetTextPtr`

FldSetUsable

Purpose Set a field to usable or nonusable.

Prototype

```
void FldSetUsable (FieldType* fldP,
Boolean usable)
```

Parameters

`fldP`	Pointer to a `FieldType` structure.
`usable`	`true` to set usable; `false` to set nonusable.

Result Returns nothing.

Comments A nonusable field doesn't display or accept input.

Use `FrmHideObject` and `FrmShowObject` instead of using this function.

See Also `FldEraseField`, `FldDrawField`, `FieldAttrType`

FldUndo

Purpose Undo the last change made to the field object, if any. Changes include typing, backspaces, delete, paste, and cut.

Prototype

```
void FldUndo (FieldType* fldP)
```

Parameters

`fldP`	Pointer to the field (`FieldType` structure) that has the focus.

Result Returns nothing.

See Also `FldPaste`, `FldCut`, `FldDelete`, `FldInsert`

FldWordWrap

Purpose Given a string and a width, return the number of bytes of characters that can be displayed using the current font.

Prototype

```
UInt16 FldWordWrap (const Char* chars,
Int16 maxWidth)
```

Parameters

-> `chars`	Pointer to a null-terminated string.
-> `maxWidth`	Maximum line width in pixels.

Result Returns the length in bytes of the characters that can be displayed.

See Also `FntWordWrap`

9

Find

This chapter provides reference material for the global find facility. The API for the find facility is defined in the header file `Find.h`.

Find Functions

FindDrawHeader

Purpose Draw the header line that separates, by database, the list of found items.

Prototype

```
Boolean FindDrawHeader (FindParamsPtr findParams,
Char const* title)
```

Parameters

`findParams`	Handle of `FindParamsPtr`.
`title`	Description of the database (for example Memos).

Result Returns `true` if Find screen is filled up. Applications should exit from the search if this occurs.

FindGetLineBounds

Purpose Returns the bounds of the next available line for displaying a match in the Find Results dialog.

Prototype

```
void FindGetLineBounds
(const FindParamsType *findParams, RectanglePtr r)
```

Parameters

`findParams`	Handle of `FindParamsPtr`.

`r`	Pointer to a structure to hold the bounds of the next results line.

Result Returns nothing.

FindSaveMatch

Purpose Saves the record and position within the record of a text search match. This information is saved so that it's possible to later navigate to the match.

Prototype

```
Boolean FindSaveMatch (FindParamsPtr findParams,
UInt16 recordNum, UInt16 pos, UInt16 fieldNum,
UInt32 appCustom, UInt16 cardNo, LocalID dbID)
```

Parameters

`findParams`	Handle of `FindParamsPtr`.
`recordNum`	Record index.
`pos`	Offset of the match string from start of record.
`fieldNum`	Field number the string was found in.
`appCustom`	Extra data the application can save with a match.
`cardNo`	Card number of the database that contains the match.
`dbID`	Local ID of the database that contains the match.

Result Returns `true` if the maximum number of displayable items has been exceeded

Comments Called by application code when it gets a match.

FindStrInStr

Purpose Perform a case-blind partial word search for a string in another string. This function assumes that the string to find has already been normalized for searching.

Prototype

```
Boolean FindStrInStr (Char const *strToSearch,
Char const *strToFind, UInt16 *posP)
```

Parameters

`strToSearch`	String to search.
`strToFind`	Normalized version of the text string to be found.
`posP`	Pointer to offset in search string of the match.

Result Returns `true` if the string was found. `FindStrInStr` matches the beginnings of words only; that is, `strToFind` must be a prefix of one of the words in `strToSearch` for `FindStrInStr` to return `true`.

Comment Don't use this function on systems that support the Text Manager. Instead, use `TxtFindString`, which performs searches on strings that contain multi-byte characters and returns the length of the matching text.

On systems that don't support the Text Manager, use `TxtGlueFindString`, found in the PalmOSGlue library. For more information, see Chapter 62, "PalmOSGlue Library."

The method by which a search string is normalized varies depending on the version of Palm OS® and the character encoding supported by the device. The string passed to your application in the `strToFind` field of the `sysAppLaunchCmdFind` launch code parameter block has already been normalized. It can be passed directly to `FindStrInStr`, `TxtFindString`, or `TxtGlueFindString`. If you have to create your own normalized search string, use `TxtGluePrepFindString`, also in the PalmOSGlue library.

10

Forms

This chapter provides the following information about form objects:

- Form Data Structures
- Form Constants
- Form Resources
- Form Functions
- Application-Defined Functions

The header file `Form.h` declares the API that this chapter describes. For more information on forms, see the section "Forms, Windows, and Dialogs" in the *Palm OS Programmer's Companion*.

Form Data Structures

FormAttrType

The `FormAttrType` bit field defines the visible characteristics of the form.

```
typedef struct {
  UInt16 usable     :1;
  UInt16 enabled    :1;
  UInt16 visible    :1;
  UInt16 dirty      :1;
  UInt16 saveBehind :1;
  UInt16 graffitiShift:1;
  UInt16 globalsAvailable : 1;
  UInt16 doingDialog : 1;
  UInt16 exitDialog : 1;
  UInt16 reserved   :7;
  UInt16 reserved2;
} FormAttrType;
```

Your code should treat the `FormAttrType` bit field as opaque. Do not attempt to change bit field member values directly.

Field Descriptions

`usable`	Not set if the form is not considered part of the current interface of the application, and it doesn't appear on screen.
`enabled`	Not used.
`visible`	Set or cleared internally when the field object is drawn or erased.
`dirty`	Not used.
`saveBehind`	Set if the bits behind the form are saved when the form is drawn.
`graffitiShift`	Set if the graffiti shift indicator is supported.
`globalsAvailable`	System use only.
`doingDialog`	System use only.
`exitDialog`	System use only.
`reserved`	Reserved for system use.
`reserved2`	Reserved for system use.

Compatibility

The `globalsAvailable`, `doingDialog`, and `exitDialog` flags are present only if 3.5 New Feature Set is present.

FormBitmapType

The `FormBitmapType` structure defines the visible characteristics of a bitmap on a form.

```
typedef struct {
  FormObjAttrType attr;
  PointType       pos;
```

```
    UInt16          rscID;
  } FormBitmapType;
```

Field Descriptions

`attr`	See `FormObjAttrType`.
`pos`	Location of the bitmap.
`rscID`	Resource ID of the bitmap. If you use `DmGetResource` with this value as the resource ID, it returns a pointer to a `BitmapType` structure.

FormFrameType

The `FormFrameType` structure defines a frame that appears on the form.

```
  typedef struct {
    UInt16          id;
    FormObjAttrType attr;
    RectangleType   rect;
    UInt16          frameType;
  } FormFrameType;
```

Field Descriptions

`id`	ID of the frame.
`attr`	See `FormObjAttrType`.
`rect`	Location and size of the frame.
`frameType`	The type of frame.

FormGadgetAttrType

The `FormGadgetAttrType` bit field defines a gadget's attributes.

```
  typedef struct {
    UInt16 usable : 1;
    UInt16 extended : 1;
    UInt16 visible : 1;
    UInt16 reserved : 13;
  } FormGadgetAttrType;
```

Your code should treat the `FormGadgetAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Field Descriptions

`usable` Not set if the gadget is not considered part of the current interface of the application, and it doesn't appear on screen. This is set by `FrmShowObject` and cleared by `FrmHideObject`.

`extended` If set, the gadget is an extended gadget. Extended gadgets are supported if 3.5 New Feature Set is present. An extended gadget has the `handler` field defined in its `FormGadgetType`. If not set, the gadgets is a standard gadget compatible with all releases of Palm OS®.

`visible` Set or cleared when the gadget is drawn or erased. `FrmHideObject` clears this value. You should set it explicitly in the gadget's callback function (if it has one) in response to a draw request.

`reserved` Reserved for future use.

Compatibility

This type is defined only if 3.5 New Feature Set is present.

FormGadgetType

The `FormGadgetType` structure defines a gadget object that appears on a form.

```
typedef struct{
  UInt16                id;
  FormGadgetAttrType    attr;
  RectangleType         rect;
  const void *          data;
  FormGadgetHandlerType *handler;
}FormGadgetType;
```

Your code should treat the `FormGadgetType` structure as opaque. Use the functions specified in the descriptions below to retrieve and

set each value. Do not attempt to change structure member values directly.

Field Descriptions

`id`	ID of the gadget resource.
`attr`	See `FormGadgetAttrType`.
`rect`	Location and size of the object.
`data`	Pointer to any specific data that needs to be stored. You can set and retrieve the value of this field with `FrmGetGadgetData` and `FrmSetGadgetData`.
`handler`	Pointer to a callback function that controls the gadget's behavior and responds to events. You can set this field with `FrmSetGadgetHandler`.

Compatibility

In Palm OS® releases prior to 3.5, the `attr` field was of type `FormObjAttrType` and the `handler` field did not exist.

FormLabelType

The `FormLabelType` structure defines a label that appears on a form.

```
typedef struct {
  UInt16          id;
  PointType       pos;
  FormObjAttrType attr;
  FontID          fontID;
  UInt8           reserved;
  Char *          text;
} FormLabelType;
```

Your code should treat the `FormLabelType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`id`	Resource ID of the label.
`pos`	Location of the label.
`attr`	See `FormObjAttrType`.
`fontID`	Font ID of the font used for the label.
`reserved`	Reserved for future use.
`text`	Text of the label.

FormLineType

The `FormLineType` structure defines a line appearing on a form.

```
typedef struct {
  FormObjAttrType   attr;
  PointType         point1;
  PointType         point2;
} FormLineType;
```

Your code should treat the `FormLineType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`attr`	See `FormObjAttrType`.
`point1`	Starting point of the line.
`point2`	Ending point of the line.

FormObjAttrType

The `FormObjAttrType` bit field defines a form object's attributes.

```
typedef struct {
  UInt16 usable : 1;
  UInt16 reserved : 15;
} FormObjAttrType;
```

Your code should treat the `FormObjAttrType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`usable`	Not set if the object is not considered part of the current interface of the application, and it doesn't appear on screen.
`reserved`	Reserved for future use.

FormObjectKind

The `FormObjectKind` enum specifies values for the `objectType` field of the `FormObjListType`. It specifies how to interpret the `object` field.

```
enum formObjects {
  frmFieldObj,
  frmControlObj,
  frmListObj,
  frmTableObj,
  frmBitmapObj,
  frmLineObj,
  frmFrameObj,
  frmRectangleObj,
  frmLabelObj,
  frmTitleObj,
  frmPopupObj,
  frmGraffitiStateObj,
  frmGadgetObj,
  frmScrollbarObj,
};
typedef enum formObjects FormObjectKind;
```

Value Descriptions

`frmFieldObj`	Text field
`frmControlObj`	Control
`frmListObj`	List
`frmTableObj`	Table
`frmBitmapObj`	Form bitmap

`frmLineObj`	Line
`frmFrameObj`	Frame
`frmRectangleObj`	Rectangle
`frmLabelObj`	Label
`frmTitleObj`	Form title
`frmPopupObj`	Popup list
`frmGraffitiStateObj`	Graffiti® state indicator
`frmGadgetObj`	Gadget (custom object)
`frmScrollbarObj`	Scrollbar

FormObjectType

The `FormObjectType` union points to the C structure for a user interface object that appears on the form.

```
typedef union {
  void *                    ptr;
  FieldType*                field;
  ControlType*              control;
  GraphicControlType *      graphicControl;
  SliderControlType *       sliderControl;
  ListType*                 list;
  TableType*                table;
  FormBitmapType*           bitmap;
  FormLabelType *           label;
  FormTitleType*            title;
  FormPopupType*            popup;
  FormGraffitiStateType*    grfState;
  FormGadgetType*           gadget;
  ScrollBarType*            scrollBar;
} FormObjectType;
```

Your code should treat the `FormObjectType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`ptr`	Used when the object's type is not one of those specified below.
`field`	Text field's structure. See `FieldType`.
`control`	Control's structure. See `ControlType`.
`graphicControl`	Graphic button structure. See `GraphicControlType`.
`sliderControl`	Slider control structure. See `SliderControlType`.
`list`	List object's structure. See `ListType`.
`table`	Table structure. See `TableType`.
`bitmap`	Form bitmap's structure. See `FormBitmapType`.
`label`	Label's structure. See `FormLabelType`.
`title`	Form title's structure. See `FormTitleType`.
`popup`	Popup list's structure. See `FormPopupType`.
`grfState`	Graffiti shift indicator's structure. See `FrmGraffitiStateType`.
`gadget`	Gadget (custom UI resource) structure. See `FormGadgetType`.
`scrollbar`	Scroll bar's structure. See `ScrollBarType`.

Compatibility

The `graphicControl` and `sliderControl` fields are only defined if 3.5 New Feature Set is present.

FormObjListType

The `FormObjectListType` structure specifies a user interface object that appears on the form.

```
typedef struct {
  FormObjectKind  objectType;
  UInt8           reserved;
  FormObjectType  object;
} FormObjListType;
```

Your code should treat the `FormObjListType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`objectType`	Specifies the type of the object (control, field, etc.). See `FormObjectKind`.
`reserved`	Reserved for future use.
`object`	The C data structure that defines the object. See `FormObjectType`.

FormPopupType

The `FormPopupType` structure defines a popup list that appears on a form.

```
typedef struct {
  UInt16          controlID;
  UInt16          listID;
} FormPopupType;
```

Your code should treat the `FormPopupType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`controlID`	Resource ID of the popup trigger control that triggers the list's display.
`listID`	Resource ID of the list object that defines the popup list.

FormPtr

The `FormPtr` type defines a pointer to a `FormType` structure.

```
typedef FormType * FormPtr;
```

FormRectangleType

The `FormRectangleType` structure defines a rectangle that appears on the form.

```
typedef struct {
  FormObjAttrType attr;
  RectangleType   rect;
} FormRectangleType;
```

Your code should treat the `FormRectangleType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`attr`	See `FormObjAttrType`.
`rect`	Location and size of the rectangle.

FormTitleType

The `FormTitleType` structure defines the title of the form.

```
typedef struct {
  RectangleType rect;
  char *        text;
} FormTitleType;
```

Your code should treat the `FormTitleType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`rect`	The location and size of the title area.
`text`	Text of the title.

FormType

The `FormType` structure and supporting structures are defined as follows:

```
typedef struct {
  WindowType                  window;
  UInt16                      formId;
  FormAttrType                attr;
  WinHandle                   bitsBehindForm;
  FormEventHandlerType *      handler;
  UInt16                      focus;
  UInt16                      defaultButton;
  UInt16                      helpRscId;
  UInt16                      menuRscId;
  UInt16                      numObjects;
  FormObjListType *           objects;
} FormType;
```

Your code should treat the `FormType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`window`	Structure of the window object that corresponds to the form. See `WindowType`.
`formId`	ID number of the form, specified by the application developer. This ID value is part of the event data of `frmOpenEvent`. The ID should match the form's resource ID.
`attr`	Form object attributes. See `FormAttrType`.
`bitsBehindForm`	Used to save all the bits behind the form so the screen can be properly refreshed when the form is closed. Use this attribute for modal forms.
`handler`	Routine called when the form needs to handle an event. You typically set this in your application's event handling function.
`focus`	Index of a field or table object within the form that contains the focus. Any `keyDownEvent` is passed to the object that has the focus. Set to `noFocus` if no object has the focus.

`defaultButton`	Resource ID of the object defined as the default button. This value is used by the routine `FrmDoDialog`.
`helpRscId`	Resource ID number of the help resource. The help resource is a String resource (type tSTR).
`menuRscId`	ID number of a menu bar to use if the form has a menu, or zero if the form doesn't have a menu.
`numObjects`	Number of objects contained within the form.
`objects`	Pointer to the array of objects contained within the form. See `FormObjListType`.

FrmGraffitiStateType

The `FrmGraffitiStateType` structure defines the graffiti shift indicator.

```
typedef struct{
  PointerType   pos;
}FrmGraffitiStateType;
```

Your code should treat the `FrmGraffitiStateType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`pos`	Location of the graffiti shift indicator.

Form Constants

The following form constants are defined:

Constant	Value	Description
`noFocus`	0xffff	No form object has the focus
`frmRedrawUpdateCode`	0x8000	Indicates that the form should be redrawn; flag in a `frmUpdateEvent`.

Constant	Value	Description
`frmNoSelectedControl`	0xff	Returned by `FrmGetControlGroupSelection` if no control is selected.
`frmResponseCreate`	1974	Passed to `FormCheckResponseFunc` to indicate that the function should perform initialization.
`frmResponseQuit`	0xBEEF	Passed to `FormCheckResponseFunc` to indicate that the function should perform cleanup.

Form Resources

The following resources are associated with forms and with the objects on a form whose data structures are defined above:

- Form—Form Resource (tFRM)
- Alert dialog— Alert Resource (Talt)
- Bitmap—Form Bitmap Resource (tFBM)
- Button—Button Resource (tBTN)
- Check box—Check Box Resource (tCBX)
- Field—Field Resource (tFLD)
- Gadget (custom object)— Gadget Resource (tGDT)
- Graffiti shift indicator —Graffiti Shift Indicator Resource (tGSI)
- Label—Label Resource (tLBL)
- List—List Resource (tLST)
- Popup trigger—Popup Trigger Resource (tPUT)
- Push button—Push Button Resource (tPBN)
- Repeating button—Repeating Button Resource (tREP)
- Scrollbar—Scroll Bar Resource (tSCL)
- Selector trigger—Selector Trigger Resource (tSLT)
- Table—Table Resource (tTBL)

Form Functions

FrmAlert

Purpose Create a modal dialog from an alert resource and display it until the user selects a button in the dialog.

Prototype
```
UInt16 FrmAlert (UInt16 alertId)
```

Parameters -> `alertId` ID of the alert resource.

Result Returns the item number of the button the user selected. A button's item number is determined by its order in the alert dialog; the first button has the item number 0 (zero).

See Also `FrmDoDialog`, `FrmCustomAlert`, `FrmCustomResponseAlert`

FrmCloseAllForms

Purpose Send a `frmCloseEvent` to all open forms.

Prototype
```
void FrmCloseAllForms (void)
```

Parameters None.

Result Returns nothing.

Comments Applications can call this function to ensure that all forms are closed cleanly before exiting `PilotMain()`; that is, before termination.

See Also `FrmSaveAllForms`

FrmCopyLabel

Purpose Copy the passed string into the data structure of the specified label object in the active form.

Prototype

```
void FrmCopyLabel (FormType *formP,
UInt16 labelID, const Char * newLabel)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `labelID`	ID of form label object.
-> `newLabel`	Pointer to a `NULL`-terminated string.

Result Returns nothing.

Comments The size of the new label **must not** exceed the size of the label defined in the resource. When defining the label in the resource, specify an initial size at least as big as any of the strings that will be assigned dynamically. This function redraws the label if the form's `usable` attribute and the label's `visible` attribute are set.

This function redraws the label but does not erase the old one first. If the new label is shorter than the old one, the end of the old label will still be visible. To avoid this, you can hide the label using `FrmHideObject`, then show it using `FrmShowObject`, after using `FrmCopyLabel`.

See Also `FrmGetLabel`

FrmCopyTitle

Purpose Copy a new title over the form's current title. If the form is visible, the new title is drawn.

Prototype

```
void FrmCopyTitle (FormType *formP,
const Char *newTitle)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `newTitle`	Pointer to the new title string.

Result Returns nothing.

Comments The size of the new title **must not** exceed the title size defined in the resource. When defining the title in the resource, specify an initial size at least as big as any of the strings to be assigned dynamically.

See Also `FrmGetTitle`, `FrmSetTitle`

FrmCustomAlert

Purpose Create a modal dialog from an alert resource and display the dialog until the user taps a button in the alert dialog.

Prototype

```
UInt16 FrmCustomAlert (UInt16 alertId,
const Char *s1, const Char *s2, const Char *s3)
```

Parameters

-> `alertId`	Resource ID of the alert.
-> `s1, s2, s3`	Strings to replace ^1, ^2, and ^3 (see Comments).

Result Returns the number of the button the user tapped (the first button is zero).

Comments A button's item number is determined by its order in the alert template; the first button has the item number zero.

Up to three strings can be passed to this routine. They are used to replace the variables ^1, ^2 and ^3 that are contained in the message string of the alert resource.

If the variables ^1, ^2, and ^3 occur in the message string, do not pass `NULL` for the arguments `s1`, `s2`, and `s3`. If you want an argument to be ignored, pass the empty string (`""`). In Palm OS 2.0 or below, pass a string containing a space (`" "`) instead of the empty string.

See Also `FrmAlert`, `FrmDoDialog`, `FrmCustomResponseAlert`

FrmCustomResponseAlert

Purpose Create a modal dialog with a text field from an alert resource and display it until the user taps a button in the alert dialog.

Prototype

```
UInt16 FrmCustomResponseAlert (UInt16 alertId,
const Char *s1, const Char *s2,
const Char *s3, Char *entryStringBuf,
Int16 entryStringBufLength,
FormCheckResponseFuncPtr callback)
```

Parameters

-> `alertId` Resource ID of the alert.

-> `s1, s2, s3` Strings to replace ^1, ^2, and ^3. See the Comments in `FrmCustomAlert` for more information.

<- `entryStringBuf` The string the user entered in the text field.

-> `entryStringBufLength` The maximum length for the string in `entryStringBuf`.

-> `callback` A callback function that processes the string. See `FormCheckResponseFunc`. Pass `NULL` if there is no callback.

Result Returns the number of the button the user tapped (the first button is zero).

Comments This function differs from FrmCustomAlert in these ways:

- The dialog it displays contains a text field for user entry. The text that the user enters is returned in the `entryStringBuf` parameter.
- When the user taps a button, the `callback` function is called and is passed the button number and `entryStringBuf`. The dialog is only dismissed if the callback returns `true`. This behavior allows you to perform error checking on the string that the user entered and give the user a chance to re-enter the string.

  The callback function is also called with special constants when the alert dialog is being initialized and when it is being deallocated. This allows the callback to perform any necessary initialization and cleanup.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also FrmAlert, FrmDoDialog

FrmDeleteForm

Purpose Release the memory occupied by a form. Any memory allocated to objects in the form is also released.

Prototype

```
void FrmDeleteForm (FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (FormType structure).

Result Returns nothing.

Comments This function doesn't modify the display.

Compatibility If 3.5 New Feature Set is present and the form contains an extended gadget, this function calls the gadget's callback with `formGadgetDeleteCmd`. See FormGadgetHandler.

See Also FrmInitForm, FrmReturnToForm

FrmDispatchEvent

Purpose Dispatch an event to the application's handler for the form.

Prototype

```
Boolean FrmDispatchEvent (EventType *eventP)
```

Parameters

-> `eventP` Pointer to an event.

Result Returns the Boolean value returned by the form's event handler or `FrmHandleEvent`. (If the form's event handler returns `false`, the event is passed to `FrmHandleEvent`.) This function also returns false if the form specified in the event is invalid.

Comments The event is dispatched to the current form's handler unless the form ID is specified in the event data, as, for example, with `frmOpenEvent` or `frmGotoEvent`. A form's event handler (`FormEventHandler`) is registered by `FrmSetEventHandler`.

Note that if the form does not have a registered event handler, this function causes a fatal error.

FrmDoDialog

Purpose Display a modal dialog until the user taps a button in the dialog.

Prototype

```
UInt16 FrmDoDialog (FormType *formP)
```

Parameters

-> `formP` Pointer to the form object (`FormType` structure).

Result Returns the resource ID of the button the user tapped.

See Also `FrmInitForm`, `FrmCustomAlert`, `FrmCustomResponseAlert`

FrmDrawForm

Purpose Draw all objects in a form and the frame around the form.

Prototype

```
void FrmDrawForm (FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

Result Returns nothing.

Comments If the `saveBehind` form attribute is set and the form is visible, this function saves the bits behind the form using the `bitsBehindForm` field in the `FormType` structure.

You should call this function in response to a `frmOpenEvent`.

If you do any custom drawing, you should do so after you call this function not before. If you do custom drawing, respond to `frmUpdateEvent` as well as `frmOpenEvent`, and be sure to return true to specify that the `frmUpdateEvent` was handled. The default event handler for `frmUpdateEvent` calls `FrmDrawForm`, so if you allow the event to fall through by returning `false`, your custom drawing is erased.

Compatibility If 3.5 New Feature Set is present, `FrmDrawForm` erases the form's window before performing any drawing. Thus, it is especially important to do any custom drawing after this function call on Palm OS 3.5 and higher.

If 3.5 New Feature Set is present and the form contains an extended gadget, this function calls the gadget's callback with `formGadgetDrawCmd`. See `FormGadgetHandler`.

See Also `FrmEraseForm`, `FrmInitForm`

FrmEraseForm

Purpose Erase a form from the display.

Prototype
```
void FrmEraseForm (FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

Result Returns nothing.

Comments If the region obscured by the form was saved by `FrmDrawForm`, this function restores that region.

FrmGetActiveForm

Purpose Return the currently active form.

Prototype
```
FormType *FrmGetActiveForm (void)
```

Parameters None.

Result Returns a pointer to the form object of the active form.

See Also `FrmGetActiveFormID`, `FrmSetActiveForm`

FrmGetActiveFormID

Purpose Return the ID of the currently active form.

Prototype
```
UInt16 FrmGetActiveFormID (void)
```

Parameters None.

Result Returns the active form's ID number.

See Also `FrmGetActiveForm`

FrmGetControlGroupSelection

Purpose Return the item number of the control selected in a group of controls.

Prototype

```
UInt16 FrmGetControlGroupSelection
(FormType *formP, UInt8 groupNum)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `groupNum`	Control group number.

Result Returns the item number of the selected control; returns `frmNoSelectedControl` if no item is selected.

Comments The item number is the index into the form object's data structure.

> **NOTE:** `FrmSetControlGroupSelection` sets the selection in a control group based on an object ID, **not** its index, which `FrmGetControlGroupSelection` returns.

Compatibility On versions prior to 3.5, this function returned a `Byte` instead of `UInt16`.

See Also `FrmGetObjectId`, `FrmGetObjectPtr`, `FrmSetControlGroupSelection`

FrmGetControlValue

Purpose Return the current value of a control.

Prototype

```
Int16 FrmGetControlValue (const FormType *formP,
UInt16 controlID)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

-> `controlID`	Index of the control object in the form object's data structure. You can obtain this by using `FrmGetObjectIndex`.

Result Returns the current value of the control. For most controls the return value is either 0 (off) or 1 (on). For sliders, this function returns the value of the `value` field.

Comments The caller must specify a valid index. This function is valid only for push button and check box control objects.

See Also `FrmSetControlValue`

FrmGetFirstForm

Purpose Return the first form in the window list.

Prototype

```
FormType *FrmGetFirstForm (void)
```

Parameters None.

Result Returns a pointer to a form object, or `NULL` if there are no forms.

Comments The window list is a LIFO stack. The last window created is the first window in the window list.

FrmGetFocus

Purpose Return the item (index) number of the object that has the focus.

Prototype

```
UInt16 FrmGetFocus (const FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

Result Returns the index of the object (UI element) that has the focus, or returns `noFocus` if none does. To convert the object index to an ID, use `FrmGetObjectId`.

See Also `FrmGetObjectPtr`, `FrmSetFocus`

FrmGetFormBounds

Purpose Return the visual bounds of the form; the region returned includes the form's frame.

Prototype

```
void FrmGetFormBounds (const FormType *formP,
RectangleType *rP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
<- `rP`	Pointer to a `RectangleType` structure where the bounds is returned.

Result Returns nothing. The bounds of the form are returned in `r`.

FrmGetFormId

Purpose Return the resource ID of a form.

Prototype
```
UInt16 FrmGetFormId (FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

Result Returns form resource ID.

See Also `FrmGetFormPtr`

FrmGetFormPtr

Purpose Return a pointer to the form that has the specified ID.

Prototype
```
FormType *FrmGetFormPtr (UInt16 formId)
```

Parameters

-> `formId`	Form ID number.

Result Returns a pointer to the form object, or `NULL` if the form is not in memory.

See Also `FrmGetFormId`

FrmGetGadgetData

Purpose Return the value stored in the `data` field of the gadget object.

Prototype
```
void *FrmGetGadgetData (const FormType *formP,
UInt16 objIndex)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

-> `objIndex`	Index of the gadget object in the form object's data structure. You can obtain this by using `FrmGetObjectIndex`.

Result Returns a pointer to the custom gadget's data.

Comments Gadget objects provide a way for an application to attach custom gadgetry to a form. In general, the `data` field of a gadget object contains a pointer to the custom object's data structure.

See Also `FrmSetGadgetData`, `FrmSetGadgetHandler`

FrmGetLabel

Purpose Return pointer to the text of the specified label object in the specified form.

Prototype

```
const Char *FrmGetLabel (FormType *formP,
UInt16 labelID)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `labelID`	ID of the label object.

Result Returns a pointer to the label string.

Comments Does not make a copy of the string; returns a pointer to the string. The object must be a label.

See Also `FrmCopyLabel`

FrmGetNumberOfObjects

Purpose Return the number of objects in a form.

Prototype

```
UInt16 FrmGetNumberOfObjects
(const FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

Result Returns the number of objects in the specified form.

See Also `FrmGetObjectPtr`, `FrmGetObjectId`

FrmGetObjectBounds

Purpose Retrieve the bounds of an object given its form and index.

Prototype

```
void FrmGetObjectBounds (const FormType *formP,
UInt16 ObjIndex, RectangleType *rP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `ObjIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.
<- `rP`	Pointer to a `RectangleType` structure where the object bounds are returned. The bounds are in window-relative coordinates.

Result Returns nothing. The object's bounds are returned in `r`.

See Also `FrmGetObjectPosition`, `FrmSetObjectPosition`

FrmGetObjectId

Purpose Return the ID of the specified object.

Prototype
```
UInt16 FrmGetObjectId (const FormType *formP,
UInt16 objIndex)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.

Result Returns the ID number of an object or `frmInvalidObjectId` if the `objIndex` parameter is invalid.

See Also `FrmGetObjectPtr`

FrmGetObjectIndex

Purpose Return the index of an object in the form's objects list.

Prototype
```
UInt16 FrmGetObjectIndex (const FormType *formP,
UInt16 objID)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objID`	ID of an object in the form.

Result Returns the index of the object (the index of the first object is 0).

Comments Bitmaps use a different mechanism for IDs than the rest of the form objects. When finding a bitmap with `FrmGetObjectIndex`, you need to pass the bitmap's resource ID, not the ID of the form bitmap object. (Passing the ID of the form bitmap object may or may not give you the right object back, depending on how you created the objects.)

This means that if you've got the same bitmap in two different form bitmap objects on the same form, you won't be able to use `FrmGetObjectIndex` to get at the second one; it'll always return the first.

See Also `FrmGetObjectPtr`, `FrmGetObjectId`

FrmGetObjectPosition

Purpose Return the coordinates of the specified object relative to the form.

Prototype

```
void FrmGetObjectPosition (const FormType *formP,
UInt16 objIndex, Coord *x, Coord *y)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.
<- `x, y`	Pointers where the window-relative x and y positions of the object are returned. These locate the top-left corner of the object.

Result Returns nothing.

See Also `FrmGetObjectBounds`, `FrmSetObjectPosition`

FrmGetObjectPtr

Purpose Return a pointer to the data structure of an object in a form.

Prototype

```
void *FrmGetObjectPtr (const FormType *formP,
UInt16 objIndex)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.

Result Returns a pointer to an object in the form.

See Also FrmGetObjectId

FrmGetObjectType

Purpose Return the type of an object.

Prototype

```
FormObjectKind FrmGetObjectType
(const FormType *formP, UInt16 objIndex)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.

Result Returns `FormObjectKind` of the item specified. See `FormObjectKind`.

FrmGetTitle

Purpose Return a pointer to the title string of a form.

Prototype

```
const Char *FrmGetTitle (const FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

Result Returns a pointer to title string, or `NULL` if there is no title string or there is an error finding it.

Comments This is a pointer to the internal structure itself, **not** to a copy.

See Also `FrmCopyTitle`, `FrmSetTitle`

FrmGetWindowHandle

Purpose Return the window handle of a form.

Prototype

```
WinHandle FrmGetWindowHandle
(const FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

Result Returns the handle of the memory block that contains the form data structure. Since the form structure begins with the `WindowType`, this is also a `WinHandle`.

FrmGotoForm

Purpose Send a `frmCloseEvent` to the current form; send a `frmLoadEvent` and a `frmOpenEvent` to the specified form.

Prototype

```
void FrmGotoForm (UInt16 formId)
```

Parameters

-> `formId`	ID of the form to display.

Result Returns nothing.

Comments The default form event handler (`FrmHandleEvent`) erases and disposes of a form when it receives a `frmCloseEvent`.

See Also `FrmPopupForm`

FrmHandleEvent

Purpose Handle the event that has occurred in the form.

Prototype

```
Boolean FrmHandleEvent (FormType *formP,
EventType *eventP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `eventP`	Pointer to the event data structure (`EventType`).

Result Returns `true` if the event was handled.

Comments Never call this function directly. Call `FrmDispatchEvent` instead. `FrmDispatchEvent` passes events to a form's custom event handler and then, if the event was not handled, to this function.

Table 10.1 provides an overview of how `FrmHandleEvent` handles different events.

Table 10.1 FrmHandleEvent Actions

When FrmHandleEvent receives...	FrmHandleEvent performs these actions...
`ctlEnterEvent`	Passes the event and a pointer to the object the event occurred in to `CtlHandleEvent`. The object pointer is obtained from the event data. If the control is part of an exclusive control group, it deselects the currently selected control of the group first.
`ctlRepeatEvent`	Passes the event and a pointer to the object the event occurred in to `CtlHandleEvent`. The object pointer is obtained from the event data.

Table 10.1 FrmHandleEvent Actions *(continued)*

When FrmHandleEvent receives...	FrmHandleEvent performs these actions...
ctlSelectEvent	Checks if the control is a Popup Trigger Control. If it is, the list associated with the popup trigger is displayed until the user makes a selection or touches the pen outside the bounds of the list. If a selection is made, a popSelectEvent is added to the event queue.
fldEnterEvent or fldHeightChangedEvent	Checks if a field object or a table object has the focus and passes the event to the appropriate handler (FldHandleEvent or TblHandleEvent). The table object is also a container object, which may contain a field object. If TblHandleEvent receives a field event, it passes the event to the field object contained within it.
frmCloseEvent	Erases the form and releases any memory allocated for it.
frmGadgetEnterEvent	Passes the event to the gadget's callback function if the gadget has one. See FormGadgetHandler.
frmGadgetMiscEvent	Passes the event to the gadget's callback function if the gadget has one. See FormGadgetHandler.
frmTitleEnterEvent	Tracks the pen until it is lifted. If it is lifted within the bounds of the form title, adds a frmTitleSelectEvent event to the event queue.
frmTitleSelectEvent	Adds a keyDownEvent with the vchrMenu character to the event queue.
frmUpdateEvent	Calls FrmDrawForm to redraw the form.

Table 10.1 FrmHandleEvent Actions *(continued)*

When FrmHandleEvent receives...	FrmHandleEvent performs these actions...
keyDownEvent	Passes the event to the handler for the object that has the focus. If no object has the focus, the event is ignored.
lstEnterEvent	Passes the event and a pointer to the object the event occurred in to LstHandleEvent. The object pointer is obtained from the event data.
menuCmdBarOpenEvent	Checks if a field object or a table object has the focus and passes the event to the appropriate handler (FldHandleEvent or TblHandleEvent), broadcasts the notification sysNotifyMenuCmdBarOpenEvent, and then displays the command toolbar.
menuEvent	Checks if the menu command is one of the system edit menu commands. The system provides a standard edit menu that contains the commands Undo, Cut, Copy, Paste, Select All, and Keyboard. FrmHandleEvent responds to these commands.
penDownEvent; pen position in the bounds of the form object	Checks the list of objects contained by the form to determine if the pen is within the bounds of one. If it is, the appropriate handler is called to handle the event, for example, if the pen is in a control, CtlHandleEvent is called. If the pen isn't within the bounds of an object, the event is ignored by the form. If the pen is within the bounds of the help icon, it is tracked until it is lifted, and if it's still within the help icon bounds, the help dialog is displayed.
popSelectEvent	Sets the label of the popup trigger to the current selection of the popup list.

Table 10.1 FrmHandleEvent Actions *(continued)*

When FrmHandleEvent receives...	FrmHandleEvent performs these actions...
sclEnterEvent or sclRepeatEvent	Passes the event and a pointer to the object the event occurred in to SclHandleEvent.
tblEnterEvent	Passes the event and a pointer to the object the event occurred in to TblHandleEvent. The object pointer is obtained from the event data.

Compatibility FrmHandleEvent only handles frmTitleSelectEvent, menuCmdBarOpenEvent, frmGadgetEnterEvent, and frmGadgetMiscEvent if 3.5 New Feature Set is present.

See Also FrmDispatchEvent

FrmHelp

Purpose Display the specified help message until the user taps the Done button in the help dialog.

Prototype

```
void FrmHelp (UInt16 helpMsgId)
```

Parameters -> helpMsgId Resource ID of help message string.

Result Returns nothing.

Comments The help message is displayed in a modal dialog that has a vertical scrollbar, if necessary.

FrmHideObject

Purpose Erase the specified object and set its attribute data (`usable` bit) so that it does not redraw or respond to the pen.

Prototype

```
void FrmHideObject (FormType *formP,
UInt16 objIndex)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.

Result Returns nothing.

Comments This function does not affect lists or tables.

Compatibility Prior to OS version 3.2, this function did not set the `usable` bit of the object attribute data to `false`. On an OS version prior to 3.2 you can work around this bug by directly setting this bit to `false` yourself.

If 3.5 New Feature Set is present and the object is an extended gadget, this function calls the gadget's callback with `formGadgetEraseCmd`. See `FormGadgetHandler`.

See Also `FrmShowObject`

FrmInitForm

Purpose Load and initialize a form resource.

Prototype

```
FormType *FrmInitForm (UInt16 rscID)
```

Parameters

-> `rscID`	Resource ID of the form.

Result Returns a pointer to the form data structure.

Displays an error message if the form has already been initialized.

Comments This function does not affect the display (use FrmDrawForm to draw the form) nor make the form active (use FrmSetActiveForm to make it active).

For each initialized form, you must call FrmDeleteForm to release the form memory when you are done with the form. Alternatively, you can free the active form by calling FrmReturnToForm.

See Also FrmDoDialog, FrmDeleteForm, FrmReturnToForm

FrmNewBitmap

Purpose Create a new form bitmap dynamically.

Prototype

```
FormBitmapType *FrmNewBitmap (FormType **formPP,
UInt16 ID, UInt16 rscID, Coord x, Coord y)
```

Parameters

<-> formPP	Pointer to a pointer to the form in which the new bitmap is installed. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new formPP value returned by this function.
-> ID	Symbolic ID of the bitmap, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
-> rscID	Numeric value identifying the resource that provides the bitmap. This value must be unique within the application scope.
-> x	Horizontal coordinate of the upper-left corner of the bitmap's boundaries, relative to the window in which it appears.

-> `y`	Vertical coordinate of the upper-left corner of the bitmap's boundaries, relative to the window in which it appears.

Result Returns a pointer to the new bitmap, or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmRemoveObject`

FrmNewForm

Purpose Create a new form object dynamically.

Prototype

```
FormType *FrmNewForm (UInt16 formID,
const Char *titleStrP, Coord x, Coord y,
Coord width, Coord height, Boolean modal,
UInt16 defaultButton, UInt16 helpRscID,
UInt16 menuRscID)
```

Parameters

-> `formID`	Symbolic ID of the form, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
-> `titleStrP`	Pointer to a string that is the title of the form.
-> `x`	Horizontal coordinate of the upper-left corner of the form's boundaries, relative to the window in which it appears.
-> `y`	Vertical coordinate of the upper-left corner of the form's boundaries, relative to the window in which it appears.
-> `width`	Width of the form, expressed in pixels. Valid values are 1 -160.
-> `height`	Height of the form, expressed in pixels.Valid values are 1 -160.

-> `modal`	`true` specifies that the form ignores pen events outside its boundaries.
-> `defaultButton`	Symbolic ID of the button that provides the form's default action, specified by the developer.
-> `helpRscID`	Symbolic ID of the resource that provides the form's online help, specified by the developer. Only modal dialogs can have help resources.
-> `menuRscID`	Symbolic ID of the resource that provides the form's menus, specified by the developer.

Result Returns a pointer to the new form object, or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmValidatePtr`, `WinValidateHandle`, `FrmRemoveObject`

FrmNewGadget

Purpose Create a new gadget dynamically and install it in the specified form.

Prototype

```
FormGadgetType *FrmNewGadget (FormType **formPP,
UInt16 id, Coord x, Coord y, Coord width,
Coord height)
```

Parameters

<-> `formPP`	Pointer to a pointer to the form in which the new gadget is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.
-> `id`	Symbolic ID of the gadget, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

-> `x`	Horizontal coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.
-> `y`	Vertical coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.
-> `width`	Width of the gadget, expressed in pixels. Valid values are 1 - 160.
-> `height`	Height of the gadget, expressed in pixels.Valid values are 1 - 160.

Result Returns a pointer to the new gadget object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Comments A gadget is a custom user interface object. For more information, see "Gadget Resource" on page 90.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmRemoveObject`

FrmNewGsi

Purpose Create a new Graffiti shift indicator dynamically and install it in the specified form.

Prototype

```
FrmGraffitiStateType *FrmNewGsi
(FormType **formPP, Coord x, Coord y)
```

Parameters

<-> `formPP`	Pointer to a pointer to the form in which the new Graffiti shift indicator is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.

-> `x`	Horizontal coordinate of the upper-left corner of the Graffiti shift indicator's boundaries, relative to the window in which it appears.
-> `y`	Vertical coordinate of the upper-left corner of the Graffiti shift indicator's boundaries, relative to the window in which it appears.

Result Returns a pointer to the new gadget object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Comments In normal operation, the Graffiti shift indicator is drawn in the lower-right portion of the screen when the user enters the shift keystroke. You use this function if the Graffiti shift indicator needs to be drawn in a nonstandard location. For example, the form manager uses it to draw the shift indicator in a custom alert dialog that contains a text field (`FrmCustomResponseAlert`).

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `FrmRemoveObject`

FrmNewLabel

Purpose Create a new label object dynamically and install it in the specified form.

Prototype

```
FormLabelType *FrmNewLabel (FormType **formPP,
UInt16 ID, const Char *textP, Coord x, Coord y,
FontID font)
```

Parameters

<-> `formPP`	Pointer to a pointer to the form in which the new label is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.

-> `ID`	Symbolic ID of the label, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
-> `textP`	Pointer to a string that provides the label text. This string is copied into the label structure.
-> `x`	Horizontal coordinate of the upper-left corner of the label's boundaries, relative to the window in which it appears.
-> `y`	Vertical coordinate of the upper-left corner of the label's boundaries, relative to the window in which it appears.
-> `font`	Font with which to draw the label text.

Result Returns a pointer to the new label object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `CtlValidatePointer`, `FrmRemoveObject`

FrmPointInTitle

Purpose Check if a coordinate is within the bounds of the form's title.

Prototype

```
Boolean FrmPointInTitle (const FormType *formP,
Coord x, Coord y)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `x, y`	Window-relative x and y coordinates.

Result Returns `true` if the specified coordinate is in the form's title.

Compatibility Implemented only if 2.0 New Feature Set is present.

FrmPopupForm

Purpose Queues a `frmLoadEvent` and a `frmOpenEvent` for the specified form.

Prototype

```
void FrmPopupForm (UInt16 formId)
```

Parameters

-> `formID`	Resource ID of form to open.

Result Returns nothing.

Comments This routine differs from `FrmGotoForm` in that the current form is not closed. You can call `FrmReturnToForm` to close a form opened by `FrmPopupForm`.

FrmRemoveObject

Purpose Remove the specified object from the specified form.

Prototype

```
Err FrmRemoveObject (FormType **formPP,
UInt16 objIndex)
```

Parameters

<-> `formPP`	Pointer to a pointer to the form from which this function removes an object. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns. In subsequent calls, always use the new `formPP` value returned by this function.
-> `objIndex`	The object to remove, specified as an index into the list of objects installed in the form. You can use the `FrmGetObjectIndex` function to discover this value.

Result Returns 0 if no error.

Comments You can use this function to remove any form object (a bitmap, control, list, and so on) and free the memory allocated to it within the form data structure. The data structures for most form objects

are embedded within the form data structure memory chunk. This function frees that memory and moves the other objects, if necessary, to close up the memory "hole" and decrease the size of the form chunk.

Note that this function does not free memory outside the form data structure that may be allocated to an object, such as the memory allocated to the string in an editable field object.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmNewBitmap`, `FrmNewForm`, `FrmNewGadget`, `FrmNewLabel`, `CtlNewControl`, `FldNewField`, `LstNewList`

FrmRestoreActiveState

Purpose Macro that restores the active window and form state.

Prototype

```
FrmRestoreActiveState (stateP)
```

Parameters

-> `stateP`	A pointer to the `FormActiveStateType` structure that you passed to `FrmSaveActiveState` when you saved the state.

Result Returns zero on success.

Comments Use this function to restore the state of displayed forms to the state that existed before you dynamically showed a new modal form. You must have previously called `FrmSaveActiveState` to save the state.

Compatibility Implemented only if 3.0 New Feature Set is present.

FrmReturnToForm

Purpose Erase and delete the currently active form and make the specified form the active form.

Prototype

```
void FrmReturnToForm (UInt16 formId)
```

Parameters

-> `formID` Resource ID of the form to return to.

Result Returns nothing.

Comments It is assumed that the form being returned to is already loaded into memory and initialized. Passing a form ID of 0 returns to the first form in the window list, which is the last form to be loaded.

`FrmReturnToForm` does not generate a `frmCloseEvent` when called from a modal form's event handler. It assumes that you have already handled cleaning up your form's variables since you are explicitly calling `FrmReturnToForm`.

See Also `FrmGotoForm`, `FrmPopupForm`

FrmSaveActiveState

Purpose Macro that saves the active window and form state.

Prototype

```
FrmSaveActiveState (stateP)
```

Parameters

<-> `stateP` A pointer to a `FormActiveStateType` structure that is used to save the state. Pass the same pointer to `FrmRestoreActiveState` to restore the state. Treat the structure like a black box; that is, don't attempt to read it or write to it.

Result Returns zero on success.

Comments Use this function to save the state of displayed forms before dynamically showing a new modal form. Call

FrmRestoreActiveState to restore the state after you remove the modal form.

Compatibility Implemented only if 3.0 New Feature Set is present.

FrmSaveAllForms

Purpose Send a frmSaveEvent to all open forms.

Prototype

```
void FrmSaveAllForms (void)
```

Parameters None.

Result Returns nothing.

See Also FrmCloseAllForms

FrmSetActiveForm

Purpose Set the active form. All input (key and pen) is directed to the active form and all drawing occurs there.

Prototype

```
void FrmSetActiveForm (FormType *formP)
```

Parameters

-> formP	Pointer to the form object (FormType structure).

Result Returns nothing.

Comments A penDownEvent outside the form but within the display area is ignored.

Compatibility In Palm OS releases earlier than 3.5, this function generated a winEnterEvent for the new form immediately following the winExitEvent for the old form. Starting in Palm OS 3.5, FrmSetActiveForm does not generate the winEnterEvent. The

`winEnterEvent` does not occur until the newly active form is drawn.

See Also `FrmGetActiveForm`

FrmSetCategoryLabel

Purpose Set the category label displayed on the title line of a form. If the form's `visible` attribute is set, redraw the label.

Prototype

```
void FrmSetCategoryLabel (FormType *formP,
UInt16 objIndex, Char *newLabel)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.
-> `newLabel`	Pointer to the name of the new category.

Result Returns nothing.

Comments The pointer to the new label (`newLabel`) is saved in the object.

FrmSetControlGroupSelection

Purpose Set the selected control in a group of controls.

Prototype

```
void FrmSetControlGroupSelection
(const FormType *formP, UInt8 groupNum,
UInt16 controlID)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `groupNum`	Control group number.

-> `controlID`	ID of control to set.

Result Returns nothing.

Comments This function unsets all the other controls in the group. The display is updated.

> **NOTE:** `FrmGetControlGroupSelection` returns the selection in a control group as an object index, **not** as an object ID, which `FrmSetControlGroupSelection` uses to set the selection.

See Also `FrmGetControlGroupSelection`

FrmSetControlValue

Purpose Set the current value of a control. If the control is visible, it's redrawn.

Prototype

```
void FrmSetControlValue (const FormType *formP,
UInt16 objIndex, Int16 newValue)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of the control in the form. You can obtain this by using `FrmGetObjectIndex`.
-> `newValue`	New value to set for the control. For sliders, specify a value between the slider's minimum and maximum. For graphical controls, push buttons, or check boxes, specify 0 for off, nonzero for on.

Result Returns nothing.

Comments This function works only with graphical controls, sliders, push buttons, and check boxes. If you set the value of any other type of control, the behavior is undefined.

See Also FrmGetControlValue

FrmSetEventHandler

Purpose Registers the event handler callback routine for the specified form.

Prototype

```
void FrmSetEventHandler (FormType *formP,
FormEventHandlerType *handler)
```

Parameters

-> formP	Pointer to the form object (FormType structure).
-> handler	Address of the form event handler function, FormEventHandler.

Result Returns nothing.

Comments FrmDispatchEvent calls this handler whenever it receives an event for a specific form.

FrmSetEventHandler must be called right after a form resource is loaded. The callback routine it registers is the mechanism for dispatching events to an application. The tutorial explains how to use callback routines.

FrmSetFocus

Purpose Set the focus of a form to the specified object.

Prototype

```
void FrmSetFocus (FormType *formP,
UInt16 fieldIndex)
```

Parameters

-> formP	Pointer to the form object (FormType structure).

-> `fieldIndex`	Index of the object to get the focus in the form. You can obtain this by using `FrmGetObjectIndex`. You can pass the constant `noFocus` so that no object has the focus.

Result Returns nothing.

Comments You can set the focus to a field or table object. If the focus is set to a field object, this function turns on the insertion point in the field by calling `FldGrabFocus` internally.

See Also FrmGetFocus

FrmSetGadgetData

Purpose Store a data value in the `data` field of the gadget object.

Prototype

```
void FrmSetGadgetData (FormType *formP,
UInt16 objIndex, const void *data)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.
-> `data`	Application-defined value. This value is stored into the `data` field of the gadget data structure (`FormGadgetType`).

Result Returns nothing.

Comments Gadget objects provide a way for an application to attach custom gadgetry to a form. Typically, the `data` field of a gadget object contains a pointer to the custom object's data structure.

See Also `FrmGetGadgetData`, `FrmSetGadgetHandler`

FrmSetGadgetHandler

Purpose Registers the gadget event handler callback routine for the specified gadget on the specified form.

Prototype

```
void FrmSetGadgetHandler (FormType *formP,
UInt16 objIndex, FormGadgetHandlerType *attrP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of a gadget object in the form. You can obtain this by using `FrmGetObjectIndex`.
-> `attrP`	Address of the callback function. See `FormGadgetHandler`.

Result Returns nothing.

Comments This function sets the application-defined function that controls the specified gadget's behavior. This function is called when the gadget needs to be drawn, erased, deleted, or needs to handle an event.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `FrmGetGadgetData`, `FrmSetGadgetData`

FrmSetMenu

Purpose Change a form's menu bar and make the new menu active.

Prototype

```
void FrmSetMenu (FormType *formP,
UInt16 menuRscID)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `menuRscID`	Resource ID of the menu.

Result Returns nothing.

Compatibility Implemented only if 2.0 New Feature Set is present.

FrmSetObjectBounds

Purpose Set the bounds or position of an object.

Prototype

```
void FrmSetObjectBounds (FormType *formP,
UInt16 objIndex, const RectangleType *bounds)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.
-> `bounds`	Window-relative bounds. For the following objects, this sets only the position of the top-left corner: label, bitmap, and Graffiti state indicator.

Result Returns nothing.

Comments Doesn't update the display.

Compatibility Implemented only if 2.0 New Feature Set is present.

FrmSetObjectPosition

Purpose Set the position of an object.

Prototype

```
void FrmSetObjectPosition (FormType *formP,
UInt16 objIndex, Coord x, Coord y)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.
-> `x`	Window-relative horizontal coordinate.

-> `y`	Window-relative vertical coordinate.

Result Returns nothing.

See Also `FrmGetObjectPosition`, `FrmGetObjectBounds`

FrmSetTitle

Purpose Set the title of a form. If the form is visible, draw the new title.

Prototype

```
void FrmSetTitle (FormType *formP, Char *newTitle)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `newTitle`	Pointer to the new title string.

Result Returns nothing.

Comments This function draws the title if the form is visible.

This function saves the pointer passed in `newTitle`; it does **not** make a copy. Don't pass a pointer to a stack-based object in `newTitle`.

This function redraws the title but does not erase the old one first. If the new title is shorter than the old one, the end of the old title will still be visible. To avoid this, you can hide the title using `FrmHideObject`, then show it using `FrmShowObject`, after using `FrmSetTitle`.

See Also `FrmGetTitle`, `FrmCopyTitle`, `FrmCopyLabel`

FrmShowObject

Purpose Set a form object as usable. If the form is visible, draw the object.

Prototype

```
void FrmShowObject (FormType *formP,
UInt16 objIndex)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `objIndex`	Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.

Result Returns nothing.

Compatibility If 3.5 New Feature Set is present and the object is an extended gadget, this function calls the gadget's callback with `formGadgetDrawCmd`. See `FormGadgetHandler`.

See Also `FrmHideObject`

FrmUpdateForm

Purpose Send a `frmUpdateEvent` to the specified form.

Prototype

```
void FrmUpdateForm (UInt16 formId,
UInt16 updateCode)
```

Parameters

-> `formId`	Resource ID of form to update.
-> `updateCode`	An application-defined code that can be used to indicate what needs to be updated. Specify the code `frmRedrawUpdateCode` to indicate that the whole form should be redrawn.

Result Returns nothing.

Comments If the `frmUpdateEvent` posted by this function is handled by the default form event handler, `FrmHandleEvent`, the `updateCode` parameter is ignored. `FrmHandleEvent` always redraws the form.

If you handle the `frmUpdateEvent` in a custom event handler, you can use the `updateCode` parameter any way you want. For example, you might use it to indicate that only a certain part of the form needs to be redrawn. If you do handle the `frmUpdateEvent`, be sure to return `true` from your event handler so that the default form handler does not also redraw the whole form.

If you do handle the `frmUpdateEvent` in a custom event handler, be sure to handle the case where `updateCode` is set to `frmRedrawUpdateCode`, and redraw the whole form. This event (and code) is sent by the system when the whole form needs to be redrawn because the display needs to be refreshed.

FrmUpdateScrollers

Purpose Visually update (show or hide) the field scroll arrow buttons.

Prototype

```
void FrmUpdateScrollers (FormType *formP,
UInt16 upIndex, UInt16 downIndex,
Boolean scrollableUp, Boolean scrollableDown)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).
-> `upIndex`	Index of the up-scroller button. You can obtain this by using `FrmGetObjectIndex`.
-> `downIndex`	Index of the down-scroller button. You can obtain this by using `FrmGetObjectIndex`.
-> `scrollableUp`	Set to `true` to make the up scroll arrow active (shown), or `false` to hide it.
-> `scrollableDown`	Set to `true` to make the down scroll arrow active (shown), or `false` to hide it.

Result Returns nothing.

FrmValidatePtr

Purpose Return `true` if the specified pointer references a valid form.

Prototype

```
Boolean FrmValidatePtr (const FormType *formP)
```

Parameters

-> `formP`	Pointer to be tested.

Result Returns `true` if the specified pointer is a non-`NULL` pointer to an object having a valid form structure.

Comments This function is intended for debugging purposes only. Do not include it in released code.

To distinguish between a window and a form in released code, instead of using this function, look at the flag `windowFlags.dialog` in the `WindowType` structure. This flag is `true` if the window is a form.

Compatibility Implemented only if 3.0 New Feature Set is present.

FrmVisible

Purpose Return `true` if the form is visible (is drawn).

Prototype

```
Boolean FrmVisible (const FormType *formP)
```

Parameters

-> `formP`	Pointer to the form object (`FormType` structure).

Result Returns `true` if the form is visible; `false` if it is not visible.

See Also `FrmDrawForm`, `FrmEraseForm`

Application-Defined Functions

FormCheckResponseFunc

Purpose Callback function for `FrmCustomResponseAlert`.

Prototype

```
Boolean FormCheckResponseFuncType (Int16 button,
Char *attempt)
```

Parameters

-> `button`	The ID of the button that the user tapped.
-> `attempt`	The string that the user entered in the alert dialog.

Result Return `true` if the dialog should be dismissed. Return `false` if the dialog should not be dismissed.

Comments This function is called at these times during the `FrmCustomResponseAlert` routine:

- At the beginning of `FrmCustomResponseAlert`, this function is called with a button ID of `frmResponseCreate`. This constant indicates that the dialog is about to be displayed, and your function should perform any necessary initialization. For example, on a Japanese system, a password dialog might need to disable the Japanese FEP. So it would call `TsmSetFepMode(NULL, tsmFepModeOff)` in this function.
- When the user has tapped a button on the dialog. The function should process the `attempt` string. If the string is valid input, the function should return `true`. If not, it should return `false` to give the user a chance to re-enter the string.
- At the end of `FrmCustomResponseAlert`, this function is called with a button ID of `frmResponseQuit`. This gives the callback a change to perform any cleanup, such as re-enabling the Japanese FEP.

Compatibility Implemented only if 3.5 New Feature Set is present.

FormEventHandler

Purpose The event handler callback routine for a form.

Prototype

```
Boolean FormEventHandlerType (EventType *eventP)
```

Parameters

-> `eventP`	Pointer to the form event (`FormType` structure).

Result Must return `true` if this routine handled the event, otherwise `false`.

Comments `FrmDispatchEvent` calls this handler whenever it receives an event for the form.

This callback routine is the mechanism for dispatching events to particular forms in an application. The callback is registered by the routine `FrmSetEventHandler`.

FormGadgetHandler

Purpose The event handler callback for an extended gadget.

Prototype

```
Boolean (FormGadgetHandlerType)
(struct FormGadgetType *gadgetP, UInt16 cmd,
void *paramP)
```

Parameters

-> `gadgetP`	Pointer to the gadget structure. See `FormGadgetType`.
-> `cmd`	A constant that specifies what action the handler should take. This can be one of the following: `formGadgetDeleteCmd` Sent by `FrmDeleteForm` to indicate that the gadget is being deleted and must clean up any memory it has allocated or perform other cleanup tasks.

	`formGadgetDrawCmd`
	Sent by `FrmDrawForm` and `FrmShowObject` to indicate that the gadget must be drawn or redrawn.
	
	`formGadgetEraseCmd`
	Sent by `FrmHideObject` to indicate that the gadget is going to be erased. `FrmHideObject` clears the `visible` and `usable` flags for you. If you return `false`, it also calls `WinEraseRectangle` to erase the gadget's bounds.
	
	`formGadgetHandleEventCmd`
	Sent by `FrmHandleEvent` to indicate that a gadget event has been received. The `paramP` parameter contains the pointer to the `EventType` structure.

-> `paramP`	`NULL` except if `cmd` is `formGadgetHandleEventCmd`. In that case, this parameter holds the pointer to the `EventType` structure containing the event.

Result Return `true` if the event was handed successfully; `false` otherwise.

Comments If this function performs any drawing in response to the `formGadgetDrawCmd`, it should set the gadget's `visible` attribute flag. (`gadgetP->attr.visible = true`). This flag indicates that the gadget appears on the screen. If you don't set the `visible` flag, the gadget won't be erased when `FrmHideObject` is called. (`FrmHideObject` immediately returns if the object's `visible` flag is `false`.)

Note that if the function receives the `formGadgetEraseCmd`, it may simply choose to perform any necessary cleanup and return `false`. If the function returns `false`, `FrmHideObject` erases the gadget's bounding rectangle. If the function returns `true`, it must erase the gadget area itself.

If this function receives a `formGadgetHandleEventCmd`, `paramP` points one of two events: `frmGadgetEnterEvent` or

`frmGadgetMiscEvent`. The `frmGadgetEnterEvent` is passed when there is a `penDownEvent` within the gadget's bounds. This function should track the pen and perform any necessary highlighting. The `frmGadgetMiscEvent` is never sent by the system. Your application may choose to use it if at any point it needs to send data to the extended gadget. In this case, the event has one or both of these fields defined: `selector`, an unsigned integer, and `dataP`, a pointer to data.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `FrmSetGadgetHandler`

11

Graffiti Shift

This chapter provides reference material for the Graffiti® Shift facility, declared in the header file `GraffitiShift.h`.

GraffitiShift Functions

GsiEnable

Purpose Enable or disable the Graffiti-shift state indicator.

Prototype

```
void GsiEnable (const Boolean enableIt)
```

Parameters

`enableIt` `true` to enable, `false` to disable.

Result Returns nothing.

Comments Enabling the indicator makes it visible, disabling it makes the insertion point invisible.

GsiEnabled

Purpose Return `true` if the Graffiti-shift state indicator is enabled, or `false` if it's disabled.

Prototype

```
Boolean GsiEnabled (void)
```

Parameters None.

Result `true` if enabled, `false` if not.

GsiInitialize

Purpose Initialize the global variables used to manage the Graffiti-shift state indicator.

Prototype
```
void GsiInitialize (void)
```

Parameters None.

Result Returns nothing.

GsiSetLocation

Purpose Set the display-relative position of the Graffiti-shift state indicator.

Prototype
```
void GsiSetLocation (const Int16 x, const Int16 y)
```

Parameters

`x, y`	Coordinate of left side and top of the indicator.

Result Returns nothing.

Comments The indicator is not redrawn by this routine.

GsiSetShiftState

Purpose Set the Graffiti-shift state indicator.

Prototype

```
void GsiSetShiftState (const UInt16 lockFlags,
const UInt16 tempShift)
```

Parameters

`lockFlags`	`glfCapsLock` or `glfNumLock`.
`tempShift`	The current temporary shift.

Result Returns nothing.

Comment This function affects only the state of the UI element, not the underlying Graffiti engine.

See Also `GrfSetState`

12

Insertion Point

This chapter provides reference material for the insertion point API, declared in the header file `InsPoint.h`.

For more information on the insertion point, see the section "Insertion Point" in the *Palm OS Programmer's Companion*.

Insertion Point Functions

InsPtEnable

Purpose Enable or disable the insertion point. When the insertion point is disabled, it's invisible; when it's enabled, it blinks.

Prototype

```
void InsPtEnable (Boolean enableIt)
```

Parameters

`enableIt`	`true` = enable; `false` = disable

Result Returns nothing.

Comments This function is called by the Form functions when a text field loses or gains the focus, and by the Windows function when a region of the display is copied (`WinCopyRectangle`).

See Also InsPtEnabled

InsPtEnabled

Purpose Return `true` if the insertion point is enabled or `false` if the insertion point is disabled.

Prototype

```
Boolean InsPtEnabled (void)
```

Parameters None.

Result Returns `true` if the insertion point is enabled (blinking); returns `false` if the insertion point is disabled (invisible).

See Also InsPtEnable

InsPtGetHeight

Purpose Return the height of the insertion point.

Prototype

```
Int16 InsPtGetHeight (void)
```

Parameters None.

Result Returns the height of the insertion point, in pixels.

InsPtGetLocation

Purpose Return the screen-relative position of the insertion point.

Prototype

```
void InsPtGetLocation (Int16 *x, Int16 *y)
```

Parameters

`x, y`	Pointer to top-left position of insertion point's x and y coordinate.

Result Returns nothing. Stores the location in `x` and `y`.

Comments This function is called by the Field functions. An application would not normally call this function.

InsPtSetHeight

Purpose Set the height of the insertion point.

Prototype
```
void InsPtSetHeight (const Int16 height)
```

Parameters

`height`	Height of the insertion point in pixels.

Result Returns nothing.

Comments Set the height of the insertion point to match the character height of the font used in the field that the insertion point is in. When the current font is changed, the insertion point height should be set to the line height of the new font.

If the insertion point is visible when its height is changed, it's erased and redrawn with its new height.

See Also InsPtGetHeight

InsPtSetLocation

Purpose Set the screen-relative position of the insertion point.

Prototype
```
void InsPtSetLocation (const Int16 x,
const Int16 y)
```

Parameters

`x, y`	Number of pixels from the left side (top) of the display.

Result Returns nothing.

Comments The position passed to this function is the location of the top-left corner of the insertion point.

This function should be called only by the Field functions.

See Also `InsPtGetLocation`

13

Lists

This chapter provides information about list objects by discussing these topics:

- List Data Structures
- List Resources
- List Functions
- Application-Defined Function

The header file `List.h` declares the API that this chapter describes. For more information on lists, see the section "Lists" in the *Palm OS Programmer's Companion*.

List Data Structures

ListAttrType

The `ListAttrType` bit field defines the visible characteristics of the list.

```
typedef struct {
  UInt16 usable     :1;
  UInt16 enabled    :1;
  UInt16 visible    :1;
  UInt16 poppedUp   :1;
  UInt16 hasScrollBar:1.
  UInt16 search     :1;
  UInt16 reserved   :2;
} ListAttrType;
```

Field Descriptions

`usable`	If not set, the form is not considered part of the current interface of the application, and it doesn't appear on screen.
`enabled`	If set, the user can interact with the list.
`visible`	Set or cleared internally when the field object is drawn or erased.
`poppedUp`	If set, choices are displayed in a popup window. This attribute is set and cleared internally.
`hasScrollBar`	If set, the list has a scroll bar.
`search`	If set, incremental search is enabled.
`reserved`	Reserved for system use.

ListType

The `ListType` structure is defined as follows:

```
typedef struct {
  UInt16              id;
  RectangleType       bounds;
  ListAttrType        attr;
  Char **             itemsText;
  Int16               numItems;
  Int16               currentItem;
  Int16               topItem;
  FontID              font;
  UInt8               reserved;
  WinHandle           popupWin;
  ListDrawDataFuncPtr drawItemCallback;
} ListType;
```

Field Descriptions

`id`	ID value, specified by the application developer. This ID value is part of the event data of `lstEnterEvent` and `lstSelectEvent`.
`bounds`	Bounds of the list, relative to the window.
`attr`	List attributes. See `ListAttrType`.
`itemsText`	Pointer to an array of pointers to the text of the choices.
`numItems`	Number of choices in the list.
`currentItem`	Currently-selected list choice (0 = first choice).
`topItem`	First choice displayed in the list.
`font`	ID of the font used to draw all list text strings.
`reserved`	Reserved for future use.
`popupWin`	Handle of the window created when a list is displayed if the `poppedUp` attribute is set.
`drawItemCallback`	Function used to draw an item in the list. If `NULL`, the default drawing routine is used instead. See Application-Defined Function.

List Resources

The List Resource (tLST), and Popup Trigger Resource (tPUT) are used together to represent an active list.

List Functions

LstDrawList

Purpose Draw the list object if it's usable. Set its `visible` attribute to `true`.

Prototype
```
void LstDrawList (ListType *listP)
```

Parameters

`listP`	Pointer to list object (ListType).

Result Returns nothing.

Comments If there are more choices than can be displayed, this function ensures that the current selection is visible. If possible, the current selection is displayed at the top. The current selection is highlighted.

If the list is disabled, it's drawn grayed-out (strongly discouraged). If it's empty, nothing is drawn. If it's not usable, nothing is drawn.

See Also `FrmGetObjectPtr`, `LstPopupList`, `LstEraseList`

LstEraseList

Purpose Erase a list object.

Prototype
```
void LstEraseList (ListType *listP)
```

Parameters

`listP`	Pointer to a list object (ListType).

Result Returns nothing.

Comments The `visible` attribute is set to `false` by this function.

See Also `FrmGetObjectPtr`, `LstDrawList`

LstGetNumberOfItems

Purpose Return the number of items in a list.

Prototype
```
Int16 LstGetNumberOfItems (const ListType *listP)
```

Parameters

`listP`	Pointer to a list object (ListType).

Result Returns the number of items in a list.

See Also `FrmGetObjectPtr`, `LstSetListChoices`

LstGetSelection

Purpose Return the currently selected choice in the list.

Prototype
```
Int16 LstGetSelection (const ListType *listP)
```

Parameters

`listP`	Pointer to list object.

Result Returns the item number of the current list choice. The list choices are numbered sequentially, starting with 0; Returns `noListSelection` if none of the items are selected.

See Also `FrmGetObjectPtr`, `LstSetListChoices`, `LstSetSelection`, `LstGetSelectionText`

LstGetSelectionText

Purpose Return a pointer to the text of the specified item in the list, or `NULL` if no such item exists.

Prototype
```
Char * LstGetSelectionText (const ListType *listP,
Int16 itemNum)
```

Parameters

`listP`	Pointer to list object.

`itemNum`	Item to select (0 = first item in list).

Result Returns a pointer to the text of the current selection, or `NULL` if out of bounds.

Comments This is a pointer within ListType, not a copy.

See Also `FrmGetObjectPtr`, `LstSetListChoices`

LstGetVisibleItems

Purpose Return the number of visible items.

Prototype

```
Int16 LstGetVisibleItems (const ListType *listP)
```

Parameters

`listP`	Pointer to list object.

Result The number of items visible.

Compatibility Implemented only if 2.0 New Feature Set is present.

LstHandleEvent

Purpose Handle event in the specified list; the list object must have its `usable` and `visible` attribute set to `true`.This routine handles two type of events, `penDownEvent` and `lstEnterEvent`; see Comments.

Prototype

```
Boolean LstHandleEvent (ListType *listP,
const EventType *eventP)
```

Parameters

`listP`	Pointer to a list object (ListType).
`eventP`	Pointer to an `EventType` structure.

Result Return `true` if the event was handled. The following cases will result in a return value of `true`:

- A `penDownEvent` within the bounds of the list

- A `lstEnterEvent` with a list ID value that matches the list ID in the list data structure

Comments When this routine receives a `penDownEvent`, it checks if the pen position is within the bounds of the list object. If it is, this routine tracks the pen until the pen comes up. If the pen comes up within the bounds of the list, a `lstEnterEvent` is added to the event queue, and the routine is exited.

When this routine receives a `lstEnterEvent`, it checks that the list ID in the event record matches the ID of the specified list. If there is a match, this routine creates and displays a popup window containing the list's choices and the routine is exited.

If a `penDownEvent` is received while the list's popup window is displayed and the pen position is outside the bounds of the popup window, the window is dismissed. If the pen position is within the bounds of the window, this routine tracks the pen until it comes up. If the pen comes up outside the list object, a `lstEnterEvent` is added to the event queue.

LstMakeItemVisible

Purpose Make an item visible, preferably at the top. If the item is already visible, make no changes.

Prototype

```
void LstMakeItemVisible (ListType *listP,
Int16 itemNum)
```

Parameters

`listP`	Pointer to a list object (ListType).
`itemNum`	Item to select (0 = first item in list).

Result Returns nothing.

Comments Does ***not*** visually update the list. You must call `LstDrawList` to update it.

See Also `FrmGetObjectPtr`, `LstSetSelection`, `LstSetTopItem`, `LstDrawList`

LstNewList

Purpose Create a new list object dynamically and install it in the specified form.

Prototype

```
Err LstNewList (void **formPP, UInt16 id, Coord x,
Coord y, Coord width, Coord height, FontID font,
Int16 visibleItems, Int16 triggerId)
```

Parameters

<--> `formPP`	Pointer to the pointer to the form in which the new list is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns. In subsequent calls, always use the new `formPP` value returned by this function.
`id`	Symbolic ID of the list, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
`x`	Horizontal coordinate of the upper-left corner of the list's boundaries, relative to the window in which it appears.
`y`	Vertical coordinate of the upper-left corner of the list's boundaries, relative to the window in which it appears.
`width`	Width of the list, expressed in pixels. Valid values are 1 – 160.
`height`	Height of the list, expressed in pixels.Valid values are 1 – 160.
`visibleItems`	Number of list items that can be viewed together.
`triggerId`	Symbolic ID of the popup trigger associated with the new list. This ID is specified by the developer; by convention, this ID should match the resource ID (not mandatory).

Result Returns 0 if no error.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `LstDrawList`, `FrmRemoveObject`

LstPopupList

Purpose Display a modal window that contains the items in the list.

Prototype

```
Int16 LstPopupList (ListType *listP)
```

Parameters

`listP`	Pointer to list object.

Result Returns the list item selected, or -1 if no item was selected.

Comments Saves the previously active window. Creates and deletes the new popup window.

See Also FrmGetObjectPtr

LstScrollList

Purpose Scroll the list up or down a number of times.

Prototype

```
Boolean LstScrollList (ListType *listP,
WinDirectionType direction, Int16 itemCount)
```

Parameters

`listP`	Pointer to list object.
`direction`	Direction to scroll.
`itemCount`	Items to scroll in direction.

Result Returns `true` when the list is actually scrolled, `false` otherwise. May return `false` if a scroll past the end of the list is requested.

Compatibility Implemented only if 2.0 New Feature Set is present.

LstSetDrawFunction

Purpose Set a callback function to draw each item instead of drawing the item's text string.

Prototype

```
void LstSetDrawFunction (ListType *listP,
ListDrawDataFuncPtr func)
```

Parameters

`listP`	Pointer to list object.
`func`	Pointer to function which draws items.

Result Returns nothing.

Comments This function also adjusts `topItem` to prevent a shrunken list from being scrolled down too far. Use this function for custom draw functionality.

See Also `FrmGetObjectPtr`, `LstSetListChoices`

LstSetHeight

Purpose Set the number of items visible in a list.

Prototype

```
void LstSetHeight (ListType *listP,
Int16 visibleItems)
```

Parameters

`listP`	Pointer to list object.
`visibleItems`	Number of choices visible at once.

Result Returns nothing.

Comments This function doesn't redraw the list if it's already visible.

See Also FrmGetObjectPtr

LstSetListChoices

Purpose Set the items of a list to the array of text strings passed to this function. This function doesn't affect the display of the list. If the list is visible, erases the old list items.

Prototype

```
void LstSetListChoices (ListType *listP,
Char **itemsText, UInt16 numItems)
```

Parameters

`listP`	Pointer to a list object.
`itemsText`	Pointer to an array of text strings.
`numItems`	Number of choices in the list.

Result Returns nothing.

See Also `FrmGetObjectPtr`, `LstSetSelection`, `LstSetTopItem`, `LstDrawList`, `LstSetHeight`, `LstSetDrawFunction`

LstSetPosition

Purpose Set the position of a list.

Prototype

```
void LstSetPosition (ListType *listP, Coord x,
Coord y)
```

Parameters

`listP`	Pointer to a list object
`x, y`	Left and top bound.

Result Returns nothing.

Comments List is not redrawn. Don't call this function when the list is visible.

See Also FrmGetObjectPtr

LstSetSelection

Purpose Set the selection for a list.

Prototype
```
void LstSetSelection (ListType *listP,
Int16 itemNum)
```

Parameters

`listP`	Pointer to a list object.
`itemNum`	Item to select (0 = first item in list; -1 = none).

Result Returns nothing.

Comments The old selection, if any, is unselected. If the list is visible, the selected item is visually updated. The list is scrolled to the selection, if necessary.

See Also `FrmGetObjectPtr`

LstSetTopItem

Purpose Set the item visible. The item cannot become the top item if it's on the last page.

Prototype
```
void LstSetTopItem (ListType *listP,
Int16 itemNum)
```

Parameters

`listP`	Pointer to list object.
`itemNum`	Item to select (0 = first item in list).

Result Returns nothing.

Comments Does ***not*** update the display.

See Also `FrmGetObjectPtr`, `LstSetSelection`, `LstMakeItemVisible`, `LstDrawList`, `LstEraseList`

Application-Defined Function

If you need to perform special drawing for items in the list, call LstSetDrawFunction to set the list drawing callback function. The callback function's prototype is:

```
void ListDrawDataFuncType (Int16 itemNum,
RectangleType *bounds, Char **itemsText)
```

14

Menus

This chapter describes the menu API as declared in the header file `Menu.h`. It discusses the following topics:

- Menu Data Structures
- Menu Constants
- Menu Resources
- Menu Functions

For more information on menus, see the section "Menus" on page 99 in the *Palm OS Programmer's Companion*.

Menu Data Structures

MenuBarAttrType

The `MenuBarAttrType` bit field defines some characteristics of the menu bar.

```
typedef struct {
  UInt16 visible : 1;
  UInt16 commandPending : 1;
  UInt16 insPtEnabled : 1;
  UInt16 needsRecalc : 1;
} MenuBarAttrType;
```

Your code should treat the `MenuBarAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Field Descriptions

`visible`	If set, the menu bar is drawn and visible on the screen. This attribute is set as part of `MenuDrawMenu`, which is called when the menu is drawn.
`commandPending`	If set, a menu command shortcut is in progress. This bit is set during `MenuHandleEvent` if the menu shortcut keystroke is received. If the next key is received before the timeout value is reached, the key is examined to see if it is a valid menu command.
`insPtEnabled`	Stores the state of the insertion point at the time the menu was drawn so that it can be restored when the menu is erased.
`needsRecalc`	If set, recalculate menu dimensions.

Compatibility

The `needsRecalc` constant is present only if 3.5 New Feature Set is present.

MenuCmdBarButtonType

The `MenuCmdBarButtonType` struct defines a button to be displayed on the command toolbar. The `buttonsData` field of the `MenuCmdBarType` struct contains an array of structures of this type.

```
typedef struct {
  UInt16  bitmapId;
  Char    name[menuCmdBarMaxTextLength];
  MenuCmdBarResultType resultType;
  UInt8   reserved;
  UInt32  result;
} MenuCmdBarButtonType;
```

Your code should treat the `MenuCmdBarButtonType` structure as opaque. Do not attempt to change structure member values directly. Instead, use `MenuCmdBarAddButton` to add a button to the

display. For the most part, the parameters to `MenuCmdBarAddButton` are the same as the fields in the `MenuCmdBarButtonType`, so there should be no need to alter these fields directly.

Field Descriptions

`bitmapId`	Resource ID of the bitmap to display on the button. This bitmap should be 13 pixels high by 16 pixels wide.
`name`	Text to display in the status message when the user taps the button.
`resultType`	Specifies what type of data is contained in the `result` field. See `MenuCmdBarResultType`.
`reserved`	Reserved for future use.
`result`	Specifies the data to send when the user clicks the button. The data is interpreted as specified by the `resultType` field. The result can be a shortcut character to enqueue in a `keyDownEvent`, a menu item ID to enqueue in a `menuEvent`, or a notification to be broadcast.

Compatibility

This structure is defined only if 3.5 New Feature Set is present.

MenuCmdBarResultType

The `MenuCmdBarResultType` enum specifies how the `result` field in the `MenuCmdBarButtonType` structure should be interpreted.

```
typedef enum {
  menuCmdBarResultNone,
  menuCmdBarResultChar,
  menuCmdBarResultMenuItem,
  menuCmdBarResultNotify
} MenuCmdBarResultType;
```

Value Descriptions

`menuCmdBarResultNone`	Send nothing.
`menuCmdBarResultChar`	The result is a character to send in a `keyDownEvent`.
`menuCmdBarResultMenuItem`	The result is the ID of the menu item to send in a `menuEvent`.
`menuCmdBarResultNotify`	The result is a notification constant to be broadcast using `SysNotifyBroadcastDeferred`.

Compatibility

This enum is defined only if 3.5 New Feature Set is present.

MenuCmdBarType

The `MenuCmdBarType` struct defines the command toolbar. This command toolbar is allocated and displayed when the user draws the shortcut stroke in the Graffiti® area. It is deallocated when `MenuEraseStatus` is called, which occurs most frequently when the timeout value has elapsed.

```
typedef struct MenuCmdBarType {
  WinHandle   bitsBehind;
  Int32       timeoutTick;
  Coord       top;
  Int16       numButtons;
  Boolean     insPtWasEnabled;
  Boolean     gsiWasEnabled;
  Boolean     feedbackMode;
  MenuCmdBarButtonType *buttonsData;
} MenuCmdBarType;
```

Your code should treat the `MenuCmdBarType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`bitsBehind`	Handle for the window that contains the region obscured by the command toolbar.
`timeoutTick`	Timeout value given in system ticks. If the user hasn't specified a command after this many ticks, the command toolbar is erased from the screen and deallocated from memory. This value also specifies how long the status message is displayed after the user successfully enters a command.
`top`	The top bound of the command toolbar given in display-relative coordinates. The command toolbar is as wide as the screen and displays at the bottom of the screen.
`numButtons`	Number of buttons displayed on the command toolbar.
`insPtWasEnabled`	If `true`, the insertion point was enabled before the command toolbar was displayed and should be re-enabled when the command toolbar is erased. If `false`, the insertion point was disabled.
`gsiWasEnabled`	If `true`, the Graffiti shift indicator was enabled before the command toolbar was displayed and should be re-enabled when the command toolbar is erased. If `false`, the Graffiti shift indicator was disabled.
`feedbackMode`	If `true`, the command toolbar is currently displaying a status message. The status message is displayed to tell the user what command is being performed. If `false`, the command toolbar is awaiting input.
`buttonsData`	The list of buttons to display on the command toolbar. See `MenuCmdBarButtonType`. Buttons are stored in this list sequentially with the rightmost button at index 0.

Compatibility

This structure is defined only if 3.5 New Feature Set is present.

MenuBarPtr

The `MenuBarPtr` type defines a pointer to a `MenuBarType`.

```
typedef MenuBarType *MenuBarPtr;
```

MenuBarType

The `MenuBarType` structure defines the menu bar. There is one menu bar per form.

```
typedef struct {
  WinHandle       barWin;
  WinHandle       bitsBehind;
  WinHandle       savedActiveWin;
  WinHandle       bitsBehindStatus;
  MenuBarAttrType attr;
  Int16           curMenu;
  Int16           curItem;
  Int32           commandTick;
  Int16           numMenus;
  MenuPullDownPtr menus;
} MenuBarType;
```

Your code should treat the `MenuBarType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`barWin`	Handle for the window that contains the menu bar.
`bitsBehind`	Handle for the window that contains the region obscured by the menu bar.
`savedActiveWin`	Handle where the currently active window is saved so that it can be restored when the menu is erased.

`bitsBehindStatus`	Handle where the bits behind the status message are saved so that when the message display terminates, the bits can be restored. The status message is displayed when the user activates the menu through the use of the command keystroke.
`attr`	Menu bar attributes. See `MenuBarAttrType`.
`curMenu`	Menu number for the currently visible menu. Menus are numbered sequentially starting with 0. The value is preserved when the menu bar is dismissed. A value of `noMenuSelection` indicates that there is no current pull-down menu.
`curItem`	Item number of the currently highlighted menu item. The items in each menu are numbered sequentially, starting with zero. A value of `noMenuItemSelection` indicates that there is no current item selected.
`commandTick`	Tick count at which the status message should be erased.
`numMenus`	Number of pull-down menus on the menu bar.
`menus`	Array of `MenuPullDownType` structures.

Compatibility

If 3.5 New Feature Set is present, the `bitsBehindStatus` and `commandTick` fields are defined but are not used. Instead, the `bitsBehind` and `timeoutTick` fields in `MenuCmdBarType`

define the save-behind window and the timeout value for the command toolbar.

MenuItemType

The `MenuItemType` structure defines a specific item within a menu. The `items` array in the `MenuPullDownType` structure contains one `MenuItemType` structure for each menu item in the pull-down menu.

If 3.5 New Feature Set is present, you can add a menu item to a pull-down menu programmatically using `MenuAddItem`.

```
typedef struct {
  UInt16  id;
  Char    command;
  UInt8   hidden: 1;
  UInt8   reserved: 7;
  Char *  itemStr;
} MenuItemType;
```

Field Descriptions

`id`	ID value you specified when you created the menu item. This ID value is included as part of the event data of a `menuEvent`.
`command`	Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.
`hidden`	If `true`, the menu item is hidden. If `false`, it is displayed. You can set and clear this value using `MenuHideItem` and `MenuShowItem`.
`reserved`	Reserved for future use.
`itemStr`	Pointer to the text to display for this menu item, including the shortcut key. To include a shortcut key, begin the string with the item's text, then type a tab character, and then the item's shortcut key. To create a separator bar, create a one-character string containing the `MenuSeparatorChar` constant.

Compatibility

The `hidden` and `reserved` fields are defined only if 3.5 New Feature Set is present.

MenuPullDownPtr

The `MenuPullDownPtr` type defines a pointer to a `MenuPullDownType`.

```
typedef MenuPullDownType * MenuPullDownPtr;
```

MenuPullDownType

The `MenuPullDownType` structure defines a specific menu accessed from the menu bar. The menus array in the `MenuBarType` structure contains one `MenuPullDownType` structure for each pull-down menu associated with the menu bar.

```
typedef struct {
  WinHandle        menuWin;
  RectangleType    bounds;
  WinHandle        bitsBehind;
  RectangleType    titleBounds;
  Char *           title;
  UInt16           hidden : 1;
  UInt16           numItems : 15;
  MenuItemType     *items;
} MenuPullDownType;
```

Field Descriptions

`menuWin`	Handle for the window that contains the menu.
`bounds`	Position and size, in pixels, of the pull-down menu.
`bitsBehind`	Handle of a window that contains the region obscured by the menu.
`title`	The menu title (null-terminated string) displayed in the menu bar.

`titleBounds`	Position and size, in pixels, of the title in the menu bar.
`hidden`	If `true`, the menu is hidden; if `false`, it is displayed. This field is not currently used.
`numItems`	Number of items in the menu. Separators count as items.
`items`	Array of `MenuItemType` structures.

Compatibility

The `hidden` field is defined only if 3.5 New Feature Set is present.

Menu Constants

Constant	Value	Description
`noMenuSelection`	-1	The `curMenu` field of `MenuBarType` is set to this when there is no currently selected menu.
`noMenuItemSelection`	-1	The `curItem` field of `MenuBarType` is set to this when there is no currently selected menu item.
`separatorItemSelection`	-2	The `curItem` field of `MenuBarType` is set to this when a menu separator item is selected.
`MenuSeparatorChar`	'–'	Special character indicating that the menu item is a bar used to separate groups of related menu items. The first character of the `itemStr` string in `MenuItemType` is set to this.

Menu Resources

The menu bar (MBAR) and pull-down menu (MENU) resources are used jointly to represent a menu object on screen. See "Menus and Menu Bars" in Chapter 2, "Palm OS Resources."

Menu Functions

MenuAddItem

Purpose Add an item to the currently active menu.

Prototype

```
Err MenuAddItem (UInt16 positionId, UInt16 id,
Char cmd, const Char *textP)
```

Parameters

-> `positionId`	ID of an existing menu item. The new menu item is added after this menu item.
-> `id`	ID value to use for the new menu item.
-> `cmd`	Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.
-> `textP`	Pointer to the text to display for this menu item, including the shortcut key. To include a shortcut key, begin the string with the item's text, then type a tab character, and then the item's shortcut key. To create a separator bar, create a one-character string containing the `MenuSeparatorChar` constant.

Result Returns 0 upon success or one of the following if an error occurs:

`menuErrNoMenu`	The `textP` parameter is `NULL`.
`menuErrSameId`	The menu already contains an item with the ID `id`.
`menuErrNotFound`	The menu doesn't contain an item with the ID `positionId`.

May display a fatal error message if there is no current menu.

Comments This function creates a new `MenuItemType` structure and adds it to the `MenuBarType`'s item list.

You should call this function only in response to a `menuOpenEvent`. This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a `keyDownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu` is called to change the form's menu. Palm OS® user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

Compatibility Implemented only if 3.5 New Feature Set is present.

MenuCmdBarAddButton

Purpose Define a button to be displayed on the command toolbar.

Prototype

```
Err MenuCmdBarAddButton (UInt8 where,
UInt16 bitmapId, MenuCmdBarResultType resultType,
UInt32 result, Char *nameP)
```

Parameters

-> `where`	Either `menuCmdBarOnLeft` to add the button to the left of the other buttons on the command toolbar, `menuCmdBarOnRight` to add it to the right of the other buttons, or a number indicating the exact position of the button. Button positions are numbered from right to left, and the rightmost position is number 1.
-> `bitmapId`	Resource ID of the bitmap to display on the button. The bitmap's dimensions should be 13 pixels high by 16 pixels wide.
-> `resultType`	The type of data contained in the `result` parameter. See `MenuCmdBarResultType`.
-> `result`	The data to send when the user taps this button. This can be a character, a menu item ID, or a notification constant.

`-> nameP` Pointer to the text to display in the status message if the user taps the button. If `NULL`, the text is taken from the menu item that matches the ID or shortcut character contained in `result`, if a match is found.

If you supply a text buffer for this parameter, `MenuCmdBarAddButton` makes a copy of the buffer.

Result Returns 0 upon success, or one of the following error codes:

`menuErrOutOfMemory`
There is not enough memory available to perform the operation.

`menuErrTooManyItems`
The command toolbar already has the maximum number of buttons allowed (currently 8).

Comments Call this function in response to a `menuCmdBarOpenEvent` or to the notification `sysNotifyMenuCmdBarOpenEvent`. Both of these signal that the user has entered the command keystroke and the command toolbar is about to open. Your response should be to add buttons to the toolbar and to return `false`, indicating that you have not completely handled the event.

The `sysNotifyMenuCmdBarOpenEvent` notification is intended to be used only by shared libraries, system extensions, and other code resources that do not use an event loop. If you're writing an application, always respond to the event instead of the notification; an application should only add buttons to the toolbar if it is the current application. If you register for the notification, you receive it each time the command toolbar is displayed, whether your application is active or not.

Note that the command toolbar is allocated each time it is opened and is deallocated when it is erased from the screen.

There is a limited amount of space in which to display buttons on the command toolbar. You should limit the number of buttons to four or five. The maximum allowed by the system is eight, but you

should leave space for the status message that appears after the user chooses an action. Buttons should be contextual; for example, the field code only displays a paste button if there is text on the clipboard. Bitmaps for the buttons should be 16 X 13 pixels.

If a field has focus when the command toolbar is opened, the field manager adds buttons for cut, copy, paste, and undo. If your application does not want this default behavior, set the `preventFieldButtons` field in the `menuCmdBarOpenEvent` structure to `true`. (Note that there is no way to prevent the field buttons from being drawn from within a notification handler.)

The following bitmaps for command toolbar buttons are defined in `UIResources.h`. The system and the built-in applications use these bitmaps to represent the commands listed in the table. Your application should also use them if it performs the same actions. If you use any of these buttons, add them in the order shown from right to left. (For example, `BarDeleteBitmap`, if used, should always be the rightmost button.)

Bitmap	Command
`BarDeleteBitmap`	Delete record.
`BarPasteBitmap`	Paste clipboard contents at insertion point.
`BarCopyBitmap`	Copy selection.
`BarCutBitmap`	Cut selection.
`BarUndoBitmap`	Undo previous action.
`BarSecureBitmap`	Show Security dialog.
`BarBeamBitmap`	Beam current record.
`BarInfoBitmap`	Show Info dialog (Launcher).

It is best to add buttons on the left side. If you add buttons to the right, this function moves all existing buttons over one position to the left. You can also specify an exact position for the `where` parameter. The positions are numbered from right to left with the rightmost position being 1. If you specify an exact position, the function leaves space for the other buttons. For example, if you

specify position 3 and there are no buttons displayed at positions 1 and 2, there will be blank spots to the right of your button.

The `result` and `resultType` parameters specify what the result should be if the user taps the button. `result` contains the actual data, and `resultType` contains a constant that specifies the type of data in `result`. Typically, the result is to enqueue a `menuEvent`. In this case, `resultType` is `menuCmdBarResultMenuItem` and the `result` is the ID of the menu item that should included in the event.

You may also specify the shortcut character instead of the menu ID; however, doing so is inefficient. When `result` is a shortcut character, the `MenuHandleEvent` function enqueues a `keyDownEvent` with the character in `result`. During the next cycle of the event loop, `MenuHandleEvent` enqueues a `menuEvent` in response to the `keyDownEvent`. Thus, it is better to have your button enqueue the `menuEvent` directly.

If you call `MenuCmdBarAddButton` outside of an application, you might not know of any menu items in the active menu (unless your code has added one using `MenuAddItem`). In this case, specify a notification to be broadcast. The notification is broadcast at the top of the next event loop, and it must contain no custom data. (Applications may also use the notification result type.)

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `MenuCmdBarDisplay`, `MenuCmdBarGetButtonData`

MenuCmdBarDisplay

Purpose Display the command toolbar.

Prototype

```
void MenuCmdBarDisplay (void)
```

Parameters None

Result Returns nothing.

Comments This function displays the command toolbar when the user enters the command keystroke. You normally do not call this function in your own code. The form manager calls it at the end of its handling of `menuCmdBarOpenEvent`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `MenuCmdBarAddButton`, `MenuCmdBarGetButtonData`

MenuCmdBarGetButtonData

Purpose Get the data for a given command button.

Prototype

```
Boolean MenuCmdBarGetButtonData
(Int16 buttonIndex, UInt16 *bitmapIdP,
MenuCmdBarResultType *resultTypeP,
UInt32 *resultP, Char *nameP)
```

Parameters

-> `buttonIndex`	Index of the button for which you want to obtain information. Buttons are ordered from right to left, with the rightmost button at index 0.
<- `bitmapIdP`	The resource ID of the bitmap displayed on the button. Pass `NULL` if you don't want to retrieve this value.
<- `resultTypeP`	The type of action this button takes. Pass `NULL` if you don't want to retrieve this value.
<- `resultP`	The result of tapping the button. Pass `NULL` if you don't want to retrieve this information.

<- `nameP`	The text displayed in the status message when this button is tapped. Pass `NULL` if you don't want to retrieve this information. If not `NULL`, `nameP` must point to a string of `menuCmdBarMaxTextLength` size.

Result Returns `true` if the information was retrieved successfully, `false` if there is no command toolbar or if there is no button at `buttonIndex`.

Comments You can use this function to retrieve information about the buttons that are displayed on the command toolbar. If the command toolbar has not yet been initialized, this function returns `false`.

Note that the command toolbar is allocated when the user enters the command keystroke and deallocated when `MenuEraseStatus` is called. Thus, the only logical place to call `MenuCmdBarGetButtonData` is in response to a `menuCmdBarOpenEvent` or `sysNotifyMenuCmdBarOpenEvent` notification.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `MenuCmdBarDisplay`, `MenuCmdBarAddButton`

MenuDispose

Purpose Release any memory allocated to the menu and the command status and restore any saved bits to the screen.

Prototype

```
void MenuDispose (MenuBarType *menuP)
```

Parameters

-> `menuP`	Pointer to the menu object to dispose. (See `MenuBarType`.) If `NULL`, this function returns immediately.

Result Returns nothing.

Comments Most applications do not need to call this function directly. `MenuDispose` is called by the system when the form that contains the menu is no longer the active form, when the form that contains the menu is freed, and when `FrmSetMenu` is called to change the form's menu bar.

See Also `MenuInit`, `MenuDrawMenu`

MenuDrawMenu

Purpose Draw the current menu bar and the last pull-down that was visible.

Prototype

```
void MenuDrawMenu (MenuBarType *menuP)
```

Parameters

-> `menuP`	Pointer to a `MenuBarType`. Must not be `NULL`.

Result Returns nothing.

Comments Most applications do not need to call this function directly. `MenuHandleEvent` calls `MenuDrawMenu` when the user taps the Menu silk-screen button (or taps the form's title on Palm OS 3.5 and higher).

The menu bar and the pull-down menu are drawn in front of all the application windows. The state of the insertion point, the bits that are obscured by the menu bar and the pull-down menu, and the currently active window are saved before the menu is drawn. These are all restored when the menu is erased.

A menu keeps track of the last pull-down menu displayed for as long as the menu is active. A menu loses its active status under these conditions:

- When `FrmSetMenu` is called to change the active menu on the form.
- When a new form, even a modal form or alert panel, becomes active.

Suppose a user selects your application's About item from the Options menu then clicks the OK button to return to the main form. When the About dialog is displayed, it becomes the active form,

which causes the main form's menu state to be erased. This menu state is not restored when the main form becomes active again. The next time `MenuDrawMenu` is called (that is, the next time the user taps the Menu silk-screen button), the menu bar is displayed as it was before, and the first pull-down menu listed in the menu bar is displayed instead of the Options pull-down menu.

See Also `MenuInit`, `MenuDispose`

MenuEraseStatus

Purpose Erase the menu command status.

Prototype

```
void MenuEraseStatus (MenuBarType *menuP)
```

Parameters

-> `menuP`	Pointer to a `MenuBarType`, or `NULL` for the current menu.

Result Returns nothing.

Comments When the user selects a menu command using the command keystroke, the command toolbar or status message is displayed at the bottom of the screen. `MenuEraseStatus` erases the toolbar or status message.

Under most circumstances, you do not need to call this function explicitly—just let the current menu command status remove itself automatically. Otherwise, you may cause text to be erased before the user has a chance to see it.

You need to call `MenuEraseStatus` explicitly only if the command toolbar covers something that is going to be changed by the menu command the user has selected. For example, if the user selects a command that displays a new form, call `MenuEraseStatus` before executing the command. Also, if the command performs some drawing in the lower portion of the window, call `MenuEraseStatus` before performing the drawing function.

Compatibility The exact behavior when a menu shortcut character is entered depends on which version of the operating system is running. In

versions prior to release 3.5, the system displays the string "Command:" in the lower-left portion of the screen when the user enters the Graffiti command keystroke.

In Palm OS 3.5 and higher, entering the Graffiti command keystroke displays the command toolbar. This toolbar is the entire width of the screen and it displays buttons that the user can tap instead of entering another keystroke. If the user taps a button or enters a character that matches a shortcut character for an item on the active menu, a status message is displayed in the toolbar while the command is executed. Calling `MenuEraseStatus` on Palm OS 3.5 and higher deallocates the command toolbar data structure as well as erasing the command toolbar from the screen.

Because the command toolbar takes up more of the display than the pre-Palm OS 3.5 status message, you may find you need to call `MenuEraseStatus` more frequently in Palm OS 3.5 than in earlier versions.

See Also `MenuInit`

MenuGetActiveMenu

Purpose Returns a pointer to the currently active menu.

Prototype

```
MenuBarType *MenuGetActiveMenu (void)
```

Parameters None.

Result Returns a pointer to the currently active menu, or `NULL` if there is none.

Comments An active menu is not necessarily visible on the screen. A menu might be active but not visible, for example, if a command shortcut has been entered. In general, a form's menu becomes active the first time a `keyDownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu` is called to change the form's menu.

If you want to know if the menu is visible rather than merely active, there are two options:

- You can check the `visible` attribute. For example:

  ```
  myMenu = MenuGetActiveMenu();
  if (myMenu && myMenu->attr.visible) {
    // menu is visible
  }
  ```

- You can check for `winEnterEvent` and `winExitEvent`.

  When the system draws a menu, the menu's window becomes the active drawing window. The system generates a `winExitEvent` for the previous active drawing window and a `winEnterEvent` for the menu's window. When the menu is erased, the system generates a `winExitEvent` for the menu's window and a `winEnterEvent` for the window that was active before the menu was drawn.

  It's common to want to check if the menu is visible in applications that perform custom drawing to a window. Such applications want to make sure that they don't draw on top of the menu. The recommended way to do this is to stop drawing when you receive a `winExitEvent` matching your drawing window and resume drawing when you receive the corresponding `winEnterEvent`. For example, the following code is excerpted from the Reptoids example application's main event loop:

  ```
  EvtGetEvent (&event, TimeUntillNextPeriod());

  if (event.eType == winExitEvent) {
    if (event.data.winExit.exitWindow ==
      (WinHandle) FrmGetFormPtr(MainView)) {
        // stop drawing.
    }
  }


  else if (event.eType == winEnterEvent) {
    if (event.data.winEnter.enterWindow ==
      (WinHandle) FrmGetFormPtr(MainView) &&
      event.data.winEnter.enterWindow ==
  ```

```
        (WinHandle) FrmGetFirstForm ()) {
          // start drawing
      }
    }
```

Note that the second method of checking to see if a menu is visible is preferred because it is not specific to menus—your application should stop drawing if any window obscures your drawing window, and it will do so if you check for `winEnterEvent` and `winExitEvent`.

See Also `MenuHandleEvent`, `MenuSetActiveMenu`

MenuHandleEvent

Purpose Handle events in the current menu. This routine handles two types of events, `penDownEvent` and `keyDownEvent`.

Prototype

```
Boolean MenuHandleEvent (MenuBarType *menuP,
EventType *event, Uint16 *error)
```

Parameters

-> `menuP`	Pointer to a `MenuBarType` data structure.
-> `event`	Pointer to an `EventType` structure.
-> `error`	Error (or 0 if no error).

Result Returns `true` if the event is handled; that is, if the event is a `penDownEvent` within the menu bar or the menu, or the event is a `keyDownEvent` that the menu supports. Returns `false` on any other event.

Comments When a `penDownEvent` is received in the menu bar, `MenuHandleEvent` tracks the pen until it comes up. If the pen comes up within the bounds of the menu bar, the selected title is inverted and the appropriate pull-down menu is drawn. Any previous pull-down menu is erased. If the pen comes up outside of the menu bar and pull-down menu, the menu is erased.

When a `penDownEvent` is received in a pull-down menu, `MenuHandleEvent` tracks the pen until it comes up. If the pen

comes up within the bounds of the menu, a menuEvent containing the resource ID of the selected menu item is added to the event queue. If the pen comes up outside of the bounds of the menu and menu bar, the menu is erased.

If a keyDownEvent is received with a vchrMenu, the menu is drawn if it is not visible and erased if it is visible.

If a keyDownEvent is received with a vchrCommand, a command shortcut is in progress. Command shortcuts are handled differently depending on which version of Palm OS is running. On versions earlier than 3.5, the next keyDownEvent is checked to see if it is a valid menu shortcut. If so, a menuEvent is added to the event queue.

If a keyDownEvent is received with a vchrCommand on Palm OS 3.5 and higher, MenuHandleEvent enqueues a menuCmdBarOpenEvent if the command toolbar is not already open. The menuCmdBarOpenEvent provides a chance for applications to add their own buttons to the command toolbar. The next event might be either a keyDownEvent with a character that completes the shortcut or a penDownEvent on one of the buttons on the toolbar. The keyDownEvent is processed as with the earlier releases— if it is a valid menu shortcut, a menuEvent is added to the event queue. If the next event is a penDownEvent, the pen is tracked until it comes up. If the pen comes up within the bounds of a button, the appropriate action is taken. See the description of MenuCmdBarAddButton for more information.

In Palm OS version 3.5 or higher, if either the vchrMenu or the vchrCommand event causes a menu to be activated and initialized for the first time, a menuOpenEvent containing the reason the menu was initialized (menuButtonCause for a vchrMenu or menuCommandCause for a vchrCommand) is added to the event queue, and then the current event is added after it.

MenuHideItem

Purpose Hide a menu item.

Prototype
```
Boolean MenuHideItem (UInt16 id)
```

Parameters -> `id` ID of the menu item to hide.

Result Returns `true` if the menu item was hidden; `false` otherwise.

Comments You should call this function only in response to a `menuOpenEvent`. This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a `keyDownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu` is called to change the form's menu. Palm OS user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `MenuShowItem`

MenuInit

Purpose Load a menu resource from a resource file.

Prototype
```
MenuBarType *MenuInit (Uint16 resourceId)
```

Parameters -> `resourceId` ID that identifies the menu resource.

Result Returns the pointer to a memory block allocated to hold the menu resource (a pointer to a `MenuBarType`).

Comments The menu is not usable until `MenuSetActiveMenu` is called.

Typically, you do not need to call this function directly. A form stores the resource ID of the menu associated with it and initializes

that menu as necessary. If you want to change the form's menu, call FrmSetMenu, which handles disposing of the form's current menu, associating the new menu with the form, and initializing when needed.

See Also MenuSetActiveMenu, MenuDispose

MenuSetActiveMenu

Purpose Set the current menu.

Prototype

```
MenuBarType *MenuSetActiveMenu
(MenuBarType *menuP)
```

Parameters

-> `menuP`	Pointer to the memory block that contains the new menu, or `NULL` for none.

Result Returns a pointer to the menu that was active before the new menu was set, or `NULL` if no menu was active.

Comments This function sets the active menu but does not associate it with a form. It's recommended that you call FrmSetMenu instead of `MenuSetActiveMenu`. `FrmSetMenu` sets the active menu, frees the old menu, and associates the newly active menu with the form, which means the menu will be freed when the form is freed.

See Also MenuGetActiveMenu

MenuSetActiveMenuRscID

Purpose Set the current menu by resource ID.

Prototype

```
void MenuSetActiveMenuRscID (Uint16 resourceId)
```

Parameters

-> `resourceId`	Resource ID of the menu to be made active.

Result Returns nothing.

Comments This function is similar to `MenuSetActiveMenu` except that you pass the menu's resource ID instead of a pointer to a menu structure. It is used as an optimization; with `MenuSetActiveMenu`, you must initialize the menu before making it active. Potentially, the application may exit before the menu is needed, making this memory allocation unnecessary. `MenuSetActiveMenuRscID` simply stores the resource ID. The next time a menu is requested, the menu is initialized from this resource.

It's recommended that you call `FrmSetMenu` instead of calling this function for the reasons given in `MenuSetActiveMenu`.

Compatibility Implemented only if 2.0 New Feature Set is present.

MenuShowItem

Purpose Display a menu item that is currently hidden.

Prototype

```
Boolean MenuShowItem (UInt16 id)
```

Parameters

-> `id` ID of the menu item to display.

Result Returns `true` if the menu item was successfully displayed, `false` otherwise.

Comments You should call this function only in response to a `menuOpenEvent`. This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a `keyDownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu` is called to change the form's menu. Palm OS user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `MenuHideItem`

15

Private Records

This chapter describes the private records API as declared in `PrivateRecords.h`. It discusses the following topics:

- Private Record Data Structures
- Private Record Functions

Private Record Data Structures

privateRecordViewEnum

The `privateRecordViewEnum` enumerated type provides the available choices for displaying private records.

```
typedef enum privateRecordViewEnum {
  showPrivateRecords = 0x00,
  maskPrivateRecords,
  hidePrivateRecords
} privateRecordViewEnum;
```

Value Descriptions

`showPrivateRecords`	Display private records in the user interface.
`maskPrivateRecords`	Show a shaded rectangle in place of a private record.
`hidePrivateRecords`	Hide private records and provide no indication in the user interface that they exist.

Private Record Functions

SecSelectViewStatus

Purpose Display a form that allows the user to select whether to hide, show, or mask private records.

Prototype

```
privateRecordViewEnum SecSelectViewStatus (void)
```

Parameters None

Result Returns a constant that indicates which option the user selected. See `privateRecordViewEnum`.

Comments This function displays a dialog that allows users to change the preference `prefShowPrivateRecords`, which controls how private records are displayed.

When the user taps the OK button in this dialog, `SecVerifyPW` is called to see if the user changed the preference setting and, if so, to prompt the user to enter the appropriate password.

After calling this function, your code should check the return value or the value of `prefShowPrivateRecords` and mask, display, or hide the private records accordingly. See the description of `TblSetRowMasked` for a partial example.

Compatibility Implemented only if 3.5 New Feature Set is present.

SecVerifyPW

Purpose Display a password dialog, verify the password, and change the private records preference.

Prototype

```
Boolean SecVerifyPW
(privateRecordViewEnum newSecLevel)
```

Parameters

-> `newSecLevel` The security level (display, hide, or mask) selected on the private records dialog.

Result Returns `true` if the `prefShowPrivateRecords` preference was successfully changed, `false` if not.

Comments This function checks `newSecLevel` against the current value for the preference. If the two values differ and `newSecLevel` indicates a decrease in security, a dialog is displayed prompting the user to enter a password. (Hidden is considered the most secure, followed by masked. Showing private records is considered the least secure.) If the password is entered successfully, the preference is changed.

This function also displays an alert message if the security level has changed to either hidden or masked.

Compatibility Implemented only if 3.5 New Feature Set is present.

16

Progress Manager

This chapter provides reference material for the Progress Manager.

- Progress Manager Functions
- Application-Defined Functions

The header file `Progress.h` declares the API that this chapter describes. For more information on the progress manager, see the section "Progress Dialogs" in the *Palm OS Programmer's Companion*.

Progress Manager Functions

PrgHandleEvent

Purpose Handles events related to the active progress dialog.

Prototype

```
Boolean PrgHandleEvent (ProgressPtr prgP,
EventType *eventP)
```

Parameters

-> `prgP`	Pointer to a progress structure created by `PrgStartDialog`.
-> `eventP`	Pointer to an event. You can pass a `NULL` event to cause this function to immediately call your `PrgCallbackFunc` function and then update the dialog (for example, after you call `PrgUpdateDialog`).

Result Returns `true` if the system handled the event. If it returns `false`, you should check if the user canceled the dialog by calling `PrgUserCancel`.

Comments Use this function instead of SysHandleEvent when you have a progress dialog. PrgHandleEvent internally calls SysHandleEvent as needed.

Note that the auto power-off feature of the system is automatically disabled when you use this function, unless the dialog is just displaying an error. This function also prevents appStopEvent events.

If an update to the dialog is pending (from a call to PrgUpdateDialog, for example) this function handles that event and causes the dialog to be updated. As part of this process, the textCallback function you specified in your call to PrgStartDialog is called. Your textCallback function should set the textP buffer in the PrgCallbackData structure with the new message to be displayed in the progress dialog. Optionally, you can also set the bitmapId field to include an icon in the update dialog. For more information about the textCallback function, see the section "Application-Defined Functions."

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also PrgStartDialog, PrgStopDialog, PrgUpdateDialog, PrgUserCancel

PrgStartDialog

Purpose Displays a progress dialog that can be updated.

Prototype

```
ProgressPtr PrgStartDialog (Char *title,
PrgCallbackFunc textCallback, void *userDataP)
```

Parameters

-> title	Pointer to a title for the progress dialog. Must be a NULL-terminated string that is no longer than progressMaxTitle (20).
-> textCallback	Pointer to a callback function that supplies the text and icons for the current progress state. See PrgCallbackFunc.

-> `userDataP`	A pointer to data that you need to access in the callback function. This address gets passed directly to your function.

Result A pointer to a progress structure. This pointer must be passed to other progress manager functions and **must** be released by calling `PrgStopDialog`. `NULL` is returned if the system is unable to allocate the progress structure.

Comments The dialog created by this function can be updated by another process via the `PrgUpdateDialog` function. The dialog can contain a Cancel or OK button. The initial dialog defaults to stage 0 and calls the `textCallback` function to get the initial text and icon data for the progress dialog.

This function saves the screen bits behind the progress dialog, and these are restored when you call `PrgStopDialog`. Because of this, you should minimize changes to the screen while the progress dialog is displayed, otherwise, the restored bits may not match with what is currently being displayed.

Compatibility This version of the function is available only if 3.2 New Feature Set is present. On earlier systems, `PrgStartDialog` has the prototype shown for `PrgStartDialogV31`.

See Also `PrgHandleEvent`, `PrgStopDialog`, `PrgUpdateDialog`, `PrgUserCancel`

PrgStartDialogV31

Purpose Displays a progress dialog that can be updated.

Prototype

```
ProgressPtr PrgStartDialogV31 (Char *title,
PrgCallbackFunc textCallback)
```

Parameters

-> `title`	Pointer to a title for the progress dialog. Must be a `NULL`-terminated string that is no longer than `progressMaxTitle` (20).

-> `textCallback`	Pointer to a callback function that supplies the text and icons for the current progress state. See `PrgCallbackFunc`.

Result A pointer to a progress structure. This pointer must be passed to other progress manager functions and **must** be released by calling `PrgStopDialog`. `NULL` is returned if the system is unable to allocate the progress structure.

Compatibility This function corresponds to version of `PrgStartDialog` available on Palm OS® 3.0 and Palm OS 3.1.

See Also `PrgHandleEvent`, `PrgStopDialog`, `PrgUpdateDialog`, `PrgUserCancel`

PrgStopDialog

Purpose Releases memory used by the progress dialog and restores the screen to its initial state.

Prototype

```
void PrgStopDialog (ProgressPtr prgP,
Boolean force)
```

Parameters

-> `prgP`	Pointer to a progress structure created by `PrgStartDialog`.
-> `force`	`true` removes the progress dialog immediately, `false` causes the system to wait until the user confirms an error, if one is displayed.

Result Returns nothing.

Comments If the progress dialog is in a state where it is displaying an error message to the user, this function normally waits for the user to confirm the dialog before it removes the dialog. If you specify `true` for the `force` parameter, this causes the system to remove the dialog immediately.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also PrgHandleEvent, PrgStartDialog, PrgUpdateDialog, PrgUserCancel

PrgUpdateDialog

Purpose Updates the status of the current progress dialog.

Prototype

```
void PrgUpdateDialog (ProgressPtr prgP,
UInt16 err, UInt16 stage, Char *messageP,
Boolean updateNow)
```

Parameters

-> prgP	Pointer to a progress structure created by PrgStartDialog.
-> err	If non-zero, causes the dialog to go into error mode, to display an error message with only an OK button.
-> stage	Current stage of progress. The callback function can use this to determine the correct string to display in the updated dialog.
-> messageP	Extra text that may be useful in displaying the progress for this stage. Used by the callback function, which can append it to the base message that is based on the stage.
-> updateNow	If true, the dialog is immediately updated. Otherwise, the dialog is updated on the next call to PrgHandleEvent.

Result Returns nothing.

Comments For more information about how the parameters are used and the callback function, see the section "Application-Defined Functions."

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `PrgHandleEvent`, `PrgStartDialog`, `PrgStopDialog`, `PrgUserCancel`

PrgUserCancel

Purpose Macro that returns `true` if the user cancelled the process via the progress dialog.

Prototype

```
PrgUserCancel (prgP)
```

Parameters

-> `prgP`	Pointer to a progress structure (`ProgressPtr`) created by `PrgStartDialog`.

Result Returns the value of the `cancel` field in the progress structure (as a `UInt16`).

Comments This is a macro you can use to check if the user cancelled the process. If the user did cancel, you can change the progress dialog text to something like "Cancelling," or "Disconnecting," or whatever is appropriate for your application. Then you should cancel the process, end the communication session, or do whatever processing is necessary.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `PrgHandleEvent`, `PrgStartDialog`, `PrgStopDialog`, `PrgUpdateDialog`

Application-Defined Functions

PrgCallbackFunc

Purpose Supplies the text and icons for the current progress state.

Prototype

```
Boolean (*PrgCallbackFunc)
(PrgCallbackDataPtr cbP)
```

Parameters

<-> `cbP`	A pointer to a `PrgCallbackData` structure. See below.

Result Returns `true` if the progress dialog should be updated using the values you specified in the `PrgCallbackData` structure. If you specify `false`, the dialog is still updated, but with default status messages. (Returning `false` is not recommended.)

Comments This is a callback function that you specify when you call `PrgStartDialog`. The callback function is called by `PrgHandleEvent` when it needs current progress information to display in the progress dialog.

The system passes this function one parameter, a pointer to a `PrgCallbackData` structure. Here are the important fields in that data structure (note that -> indicates you set the field in the `textCallback` function):

<- `UInt16 stage`	Current stage (passed from `PrgUpdateDialog`).
<-> `Char *textP`	Buffer to hold the text to display in the updated dialog. You might want to look up a message in a resource file, based on the value in the `stage` field. Also, you should append the additional text in the `message` field, to form the full string to display. Be sure to include a null terminator at the end of the string you return, and don't exceed the length in `textLen`.

`<- UInt16 textLen`
: Maximum length of the text buffer `textP`. Note that this value is set for you by the caller. Be careful not to exceed this length in `textP`.

`<- Char *message`
: Additional text to display in the dialog (from the `messageP` parameter to `PrgUpdateDialog`). This should be no longer than `progressMaxMessage` (128).

`<- Err error`
: Current error (passed from the `err` parameter to `PrgUpdateDialog`).

`-> UInt16 bitmapId`
: Resource ID of the bitmap to display in the progress dialog, if any.

`<- UInt16 canceled:1`
: `true` if user has pressed the cancel button.

`<- UInt16 showDetails:1`
: `true` if user pressed the down arrow button on the Palm device for more details. (Because this is a non-standard user interface technique, you shouldn't use this feature to display details that users need under normal conditions. It's more for debugging purposes.)

`-> UInt16 textChanged:1`
: If `true`, then update text (defaults to `true`). You can set this to `false` to avoid an update to the text.

`<- UInt16 timedOut:1`
: `true` if update caused by a timeout.

`<-> UInt32 timeout`
: Timeout in ticks to force next update. After this number of ticks, an update is automatically triggered (which sets the `timedOut` flag). You can use this feature to do a simple animation effect. Note that you must set the timeout for `EvtGetEvent` to a value that is equal to or less than this value, otherwise you won't get update events as frequently as you expect.

`-> UInt16 delay:1`
: If `true`, delay for one second after updating the dialog. Use this value when you are displaying the final progress message so that users have a chance to see the message before the dialog closes. This field is available only if 3.2 New Feature Set is present.

`<- void *userDataP`
: A pointer to any application-defined data that the function needs to access. You specify this pointer as a parameter to `PrgStartDialog` if the callback function needs to access some application data but does not have access to application globals. This field is available only if 3.2 New Feature Set is present.

In this function, you should set the value of the `textP` buffer to the string you want to display in the progress dialog when it is updated. You can use the value in the `stage` field to look up a message in a string resource. You also might want to append the text in the `message` field to your base string. Typically, the `message` field would contain more dynamic information that depends on a user selection, such as a phone number, device name, or network identifier, etc.

For example, the `PrgUpdateDialog` function might have been called with a `stage` of 1 and a `messageP` parameter value of a phone number string, "555-1212". Based on the stage, you might find the string "Dialing" in a string resource, and append the phone number, to form the final text "Dialing 555-1212" that you place in the text buffer `textP`.

Keeping the static strings corresponding to various stages in a resource makes it easier to localize your application. More dynamic information can be passed in via the `messageP` parameter to `PrgUpdateDialog`.

NOTE: This function is called only if the parameters passed to `PrgUpdateDialog` have changed from the last time it was called. If `PrgUpdateDialog` is called twice with exactly the same parameters, the `textCallback` function is called only once.

17

Scroll Bars

This chapter provides reference material for the scroll bar API.

- Scroll Bar Data Structures
- Scroll Bar Resources
- Scroll Bar Functions

The header file `ScrollBar.h` declares the API that this chapter describes. For more information on scroll bars, see the section "Scroll Bars" on page 112 in the *Palm OS Programmer's Companion*.

Scroll Bar Data Structures

ScrollBarAttrType

The `ScrollBarAttrType` bit field defines a scroll bar's visible characteristics.

```
typedef struct {
  UInt16 usable:      1;
  UInt16 visible:     1;
  UInt16 hilighted:   1;
  UInt16 shown:       1;
  UInt16 activeRegion:4;
} ScrollBarAttrType;
```

Field Descriptions

`usable`	If not set, the scroll bar object is not considered part of the current interface of the application, and it doesn't appear on screen.
`visible`	If set, the scroll bar is allowed to be displayed on the screen. If both `visible` and `shown` are `true`, then the scroll bar is actually displayed on the screen.

`hilighted`	`true` if either the up arrow or the down arrow is highlighted.
`shown`	Set if the scroll bar is visible and if `maxValue` > `minValue`. (See `ScrollBarType`.)
`activeRegion`	The region of the scroll bar that is receiving the pen down events. Possible values are:
	`sclUpArrow` — The up arrow.
	`sclDownArrow` — The down arrow.
	`sclUpPage` — The region between the scroll car and the up arrow.
	`sclDownPage` — The region between the scroll car and the down arrow.
	`sclCar` — The scroll car.

ScrollBarPtr

The `ScrollBarPtr` type defines a pointer to a `ScrollBarType` structure.

```
typedef ScrollBarType *ScrollBarPtr;
```

You pass the `ScrollBarPtr` as an argument to all scroll bar functions. You can obtain the `ScrollBarPtr` using the function `FrmGetObjectPtr` in this way:

```
scrollBarPtr = FrmGetObjectPtr(frm,
  FrmGetObjectIndex(frm, scrollBarID));
```

where `scrollBarID` is the resource ID assigned when you created the scroll bar.

ScrollBarType

The `ScrollBarType` represents a scroll bar.

```
typedef struct {
  RectangleType        bounds;
  UInt16               id;
```

```
    ScrollBarAttrType   attr;
    Int16               value;
    Int16               minValue;
    Int16               maxValue;
    Int16               pageSize;
    Int16               penPosInCar;
    Int16               savePos;
  } ScrollBarType;
```

Your code should treat the `ScrollBarType` structure as opaque. Use the functions described in this chapter to retrieve and set each value. Do not attempt to change structure member values directly.

Field Descriptions

`bounds` Position (using absolute coordinates) and size (in pixels) of the scroll bar on the screen.

`id` ID value you specified when you created the scroll bar object.

`attr` Scroll bar's attributes. See `ScrollBarAttrType`.

`value` Current value of the scroll bar. This value is used to determine where to position the scroll car (the dark region in the scroll bar that indicates the position in the document).

The number given is typically a number relative to `minValue` and `maxValue`. These values have nothing to do with any physical characteristics of the object that the scroll bar is attached to, such as the number of lines in the object.

This value is typically set to 0 initially and then adjusted programmatically with `SclSetScrollBar`.

`minValue` Minimum value. When `value` equals `minValue`, the scroll car is positioned at the very top of the scrolling region. This value is typically 0.

`maxValue`	Maximum value. When `value` equals `maxValue`, the scroll car is positioned at the very bottom of the scrolling region. This value is typically set to 0 initially and then adjusted programmatically with `SclSetScrollBar`.
`pageSize`	Number of lines to scroll when user scrolls one page.
`penPosInChar`	Used internally.
`savePos`	Used internally.

Scroll Bar Resources

The Scroll Bar Resource (tSCL) represents a scroll bar.

Scroll Bar Functions

SclDrawScrollBar

Purpose Draw a scroll bar.

Prototype

```
void SclDrawScrollBar (const ScrollBarPtr bar)
```

Parameters

-> `bar`	Pointer to a scroll bar structure (see `ScrollBarType`).

Result Returns nothing.

Comments This function is called internally by `SclSetScrollBar` and `FrmDrawForm`. You rarely need to call it yourself.

Compatibility Implemented only if 2.0 New Feature Set is present.

SclGetScrollBar

Purpose Retrieve a scroll bar's current position, its range, and the size of a page.

Prototype

```
void SclGetScrollBar (const ScrollBarPtr bar,
Int16 *valueP, Int16 *minP, Int16 *maxP,
Int16 *pageSizeP)
```

Parameters

-> `bar`	Pointer to a scroll bar structure (see `ScrollBarType`).
<- `valueP`	A value representing the scroll car's current position. (The scroll car is the dark region that indicates the position in the document.)
<-`minP`	A value representing the top of the user interface object.
<-`maxP`	A value representing the bottom of the user interface object.
<-`pageSizeP`	Pointer to size of a page (used when page scrolling).

Result Returns the scroll bar's current values in `valueP`, `minP`, `maxP`, and `pageSizeP`.

Comments You might use this function immediately before calling `SclSetScrollBar` to update the scroll bar. `SclGetScrollBar` returns the scroll bar's current values, which you can then adjust as necessary and pass to `SclSetScrollBar`.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `SclSetScrollBar`

SclHandleEvent

Purpose Handles events that affect a scroll bar.

Prototype

```
Boolean SclHandleEvent (const ScrollBarPtr bar,
const EventType *event)
```

Parameters

-> `bar`	Pointer to a scroll bar structure (see `ScrollBarType`).
-> `event`	Pointer to an event (EventType).

Result Returns `true` if the event was handled.

Comment When a `penDownEvent` occurs, the scroll bar sends an `sclEnterEvent` to the event queue.

When an `sclEnterEvent` occurs, the scroll bar determines what its new value should be based on which region of the scroll bar is receiving the pen down events. It then sends either an `sclRepeatEvent` or an `sclExitEvent` to the event queue.

When the user holds and drags the scroll bar with the pen, the scroll bar sends a `sclRepeatEvent`. Applications that implement dynamic scrolling should catch this event and move the text each time one arrives.

When the user releases the pen from the scroll bar, the scroll bar sends a `sclExitEvent`. Applications that implement non-dynamic scrolling should catch this event and move the text when `sclExitEvent` arrives. Applications that implement dynamic scrolling can ignore this event.

Compatibility Implemented only if 2.0 New Feature Set is present.

SclSetScrollBar

Purpose

Set the scroll bar's current position, its range, and the size of a page. If the scroll bar is visible and its minimum and maximum values are not equal, it's redrawn.

Prototype

```
void SclSetScrollBar (const ScrollBarPtr bar,
Int16 value, const Int16 min, const Int16 max,
const Int16 pageSize)
```

Parameters

-> `bar`	Pointer to a scroll bar structure (see `ScrollBarType`).
-> `value`	The position the scroll car should move to. (The scroll car is the dark region that indicates the position in the document.)
-> `min`	Minimum value.
-> `max`	Maximum value.
-> `pageSize`	Number of lines of text that can be displayed on a the screen at one time (used when page scrolling).

Result

Returns nothing. May display a fatal error message if the `min` parameter is greater than the `max` parameter.

Comments

Call this function when the user adds or deletes text in a field or when a table row is added or deleted.

For scrolling fields, your application should catch the `fldChangedEvent` and update the scroll bar at that time.

The `max` parameter is computed as:

number of lines of text – page size + overlap

where number of lines of text is the total number of lines or rows needed to display the entire object, page size is the number of lines or rows that can be displayed on the screen at one time, and overlap is the number of lines or rows from the bottom of one page to be visible at the top of the next page.

For example, if you have 100 lines of text and 10 lines show on a page, the `max` value would be 90 or 91, depending on the overlap. So if value is greater than or equal to 90 or 91, the scroll car is at the very bottom of the scrolling region.

You can use the `FldGetScrollValues` function to compute the values you pass for `value`, `min`, and `max`. For example:

```
FldGetScrollValues (fld, &scrollPos,
  &textHeight,  &fieldHeight);

if (textHeight > fieldHeight)
  maxValue = textHeight - fieldHeight;
else if (scrollPos)
  maxValue = scrollPos;
else
  maxValue = 0;

SclSetScrollBar (bar, scrollPos, 0, maxValue,
  fieldHeight-1);
```

In this case, `textHeight` is the number of lines of text and `fieldHeight` is the page size. No lines overlap when you scroll one page. Notice that if the page size is greater than the lines of text, then `max` equals `min`, which means that the scroll bar is not displayed.

For scrolling tables, there is no equivalent to `FldGetScrollValues`. Your application must scroll the table itself and keep track of the scroll values. See the `ListViewUpdateScrollers` function in the Memo example application (`MemoMain.c`) for an example of setting scroll bar values for a table.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `SclGetScrollBar`

18

System Dialogs

This chapter provides reference material for system dialogs declared in the header files `FatalAlert.h`, `Launcher.h`, `GraffitiReference.h`, and `GraffitiUI.h`.

System Dialog Functions

SysAppLauncherDialog

Purpose Display the launcher popup, get a choice, ask the system to launch the selected application, clean up, and leave. If there are no applications to launch, nothing happens.

Prototype

```
void SysAppLauncherDialog()
```

Parameters None.

Result The system may be asked to launch an application.

Comments Typically, this routine is called by the system as necessary. Most applications do not need to call this function themselves.

In Palm OS® version 3.0 and higher the launcher is an application, rather than a popup. This function remains available for compatibility purposes only.

See Also `SysAppLaunch`, the "Application Launcher" section in the *Palm OS Programmer's Companion*

SysFatalAlert

Purpose Display a fatal alert until the user taps a button in the alert.

Prototype

```
UInt16 SysFatalAlert (const Char *msg)
```

Parameters

`msg`	Message to display in the dialog.

Result The button tapped; first button is zero.

SysGraffitiReferenceDialog

Purpose Pop up the Graffiti® Reference Dialog.

Prototype

```
void SysGraffitiReferenceDialog
(ReferenceType referenceType)
```

Parameters

`referenceType`	Which reference to display. See `GraffitiReference.h` for more information.

Result Nothing returned.

19

Tables

This chapter describes the table API as declared in the header file `Table.h`. It discusses the following topics:

- Table Data Structures
- Table Constants
- Table Resource
- Table Functions
- Application-Defined Functions

For more information on tables, see the section "Tables" in the *Palm OS Programmer's Companion*.

Table Data Structures

TableAttrType

The `TableAttrType` bit field defines the visible characteristics of the table.

```
typedef struct {
  UInt16 visible:1;
  UInt16 editable:1;
  UInt16 editing:1;
  UInt16 selected:1;
  UInt16 hasScrollBar:1;
  UInt16 reserved:11;
} TableAttrType;
```

Your code should treat the `TableAttrType` bit field as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change member values directly.

Field Descriptions

`visible`	If set, the table is drawn on the screen. The value of this field is set by `TblDrawTable` and cleared by `TblEraseTable`.
`editable`	If set, the user can modify the table. You specify this when you create the table resource.
`editing`	If set, the table is in edit mode. The table is in edit mode while the user edits a text item. The value of this field is returned by `TblEditing`.
`selected`	If set, the current item (as identified by the `TableType` fields `currentRow` and `currentColumn`) is selected. Use `TblGetSelection` to retrieve this value.
`hasScrollBar`	If set, the table has a scroll bar. Note that this attribute can only be set programmatically. See `TblHasScrollBar`.

TableColumnAttrType

The `TableColumnAttrType` structure defines a column in a table.

```
typedef struct {
  Coord                  width;
  UInt16                 reserved1 : 5;
  UInt16                 masked : 1;
  UInt16                 editIndicator : 1;
  UInt16                 usable : 1;
  UInt16                 reserved2 : 8;
  Coord                  spacing;
  TableDrawItemFuncPtr   drawCallback;
  TableLoadDataFuncPtr   loadDataCallback;
  TableSaveDataFuncPtr   saveDataCallback;
} TableColumnAttrType;
```

Your code should treat the `TableColumnAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Field Descriptions

`width`	The column's width in pixels. See `TblGetColumnWidth` and `TblSetColumnWidth`.
`reserved1`	Reserved for future use.
`masked`	If `true` and the item's row also has a `masked` attribute of `true`, the table cell is drawn on the screen but is shaded to obscure the information that it contains. See `TblSetColumnMasked`.
`editIndicator`	If `true`, items in the column should be highlighted if selected while in edit mode. If `false`, items in the column should not be highlighted. By default, text field items are highlighted in edit mode, but all other types of items are not highlighted. The default can be overridden with `TblSetColumnEditIndicator`.
`usable`	If `false`, the column is not considered part of the current interface of the application, and it doesn't appear on screen. See `TblSetColumnUsable`.
`reserved2`	Reserved for future use.
`spacing`	The spacing in pixels between this column and the next column. See `TblGetColumnSpacing` and `TblSetColumnSpacing`.
`drawCallback`	Pointer to a function that draws custom items in the column. This function is called during `TblDrawTable` and `TblRedrawTable`. See `TblSetCustomDrawProcedure`.

`loadDataCallback`	Pointer to a function that loads data into the column. This function is called during `TblDrawTable` and `TblRedrawTable`. See `TblSetLoadDataProcedure`.
`saveDataCallback`	Pointer to a function that saves the data in the column. Called when the focus moves from one table cell to another and when the table loses focus entirely. See `TblSetSaveDataProcedure`.

Compatibility

The `masked` field is defined only if 3.5 New Feature Set is present.

TableItemPtr

A `TableItemPtr` points to a `TableItemType`.

```
typedef TableItemType *TableItemPtr;
```

TableItemType

The `TableItemType` structure defines an item, or cell, within the table.

```
typedef struct {
  TableItemStyleType  itemType;
  FontID              fontID;
  Int16               intValue;
  Char *              ptr;
} TableItemType;
```

Your code should treat the `TableItemType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

NOTE: None of the table items create memory that you need to free. The table manager handles all of the allocating and deallocating of memory for table items. The only memory you are responsible for freeing is the memory handle containing the text that you want displayed in editable text fields. (See `TableLoadDataFuncType`.)

Field Descriptions

`itemType` The type of the item, such as a control, a text label, and so on. `TblSetItemStyle` sets this value. The rest of the fields in this struct are either used or not used depending on the `itemType`. See Table 19.1.

`fontID` ID of the font used to display the item's text. `TblGetItemFont` and `TblSetItemFont` retrieve and set this value.

`intValue` Integer value of the item. `TblGetItemInt` and `TblSetItemInt` retrieve and set this value.

`ptr` Pointer to the item's text. `TblGetItemPtr` and `TblSetItemPtr` retrieve and set this value.

All text items have a maximum of `tableMaxTextItemSize`.

The following table lists the possible values for the `itemType` field, describes how each type is drawn, describes which other fields are used for each `itemType`, and provides special instructions for setting those fields. Note in particular that the `fontID` field is often not used. Instead, certain items are displayed in a standard font. These are noted in the last column of this table.

Table 19.1 TableItemType fields

itemType	Description	TableItemType Fields Used
`checkboxTableItem`	A checkbox control.	`intValue`
`customTableItem`	Application-defined cell.	None. Custom items are drawn using the custom drawing function that you implement. See `TableDrawItemFuncType`. If you want, you can store data in the `intValue` and `ptr` fields.

`dateTableItem`	Non-editable date in the form *month/day*, or a dash if the date value is -1. The date is followed by an exclamation point if it has past.	`intValue`
		The `intValue` field should be a value that can be cast to `DateType`. `DateType` is currently defined as a 16-bit number:
		`yyyyyyymmmmddddd`
		The first 7 bits are the year given as the offset since 1904, the next 4 bits are the month, and the next 5 bits are the day.
		Dates are always drawn in the current font.

`labelTableItem`	Non-editable text.	`ptr` Labels are displayed in the system's default font.
`numericTableItem`	Non-editable number.	`intValue` Numbers are displayed in the system's default bold font.

Table 19.1 TableItemType fields *(continued)*

itemType	Description	TableItemType Fields Used

`popupTriggerTableItem`	A list with a pop-up trigger.	`intValue`
		`ptr`
		`intValue` is the index of the list item that should be displayed.
		`ptr` is a pointer to the list.
		Lists are displayed in the system's default font.

`textTableItem`	Editable text field.	`fontID`
		`ptr`
		For this item type, implement the callback function `TableLoadDataFuncType` to load text into the table cell and implement the callback `TableSaveDataFuncType` to save data before the field is freed.

`textWithNoteTableItem`	Editable text field and a note icon to the right of the text.	`fontID`
		`ptr`
		For this item type, implement the callback function `TableLoadDataFuncType` to load text into the table cell and implement the callback `TableSaveDataFuncType` to save data before the field is freed.

Table 19.1 TableItemType fields *(continued)*

itemType	Description	TableItemType Fields Used
`timeTableItem`	Not implemented.	

`narrowTextTableItem`	Editable text with space reserved on the right side of the cell.	`fontID`
		`ptr`
		`intValue`
		
		`intValue` is the number of pixels to reserve on the right side of the cell.
		
		For this item type, implement the callback function `TableDrawItemFuncType` to draw in the space reserved on the right side of the cell, the `TableLoadDataFuncType` callback function to load text into the table cell, and the callback function `TableSaveDataFuncType` to save data before the field is freed.

TablePtr

The `TablePtr` type defines a pointer to a `TableType`.

```
typedef TableType * TablePtr;
```

You pass the table's pointer as an argument to all table functions. You can obtain the table's pointer using the function `FrmGetObjectPtr` in this way:

```
tblPtr = FrmGetObjectPtr(frm,
  FrmGetObjectIndex(frm, tblID));
```

where `tblID` is the resource ID assigned when you created the table.

TableRowAttrType

The `TableRowAttrType` structure defines a row in a table.

```
typedef struct {
  UInt16      id;
  Coord       height;
  UInt32      data;
  UInt16      reserved1 : 7;
  UInt16      usable : 1;
  UInt16      reserved2 : 4;
  UInt16      masked : 1;
  UInt16      invalid : 1;
  UInt16      staticHeight : 1;
  UInt16      selectable : 1;
  UInt16      reserved3;
} TableRowAttrType;
```

Your code should treat the `TableRowAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Field Descriptions

`id`	The ID of this row. See `TblFindRowID`, `TblGetRowID`, and `TblSetRowID`.
`height`	Height of the row in pixels. The functions `TblSetRowHeight` and `TblGetRowHeight` set and retrieve this value.
`data`	Any application-specific value you want to store in this row. For example, the Datebook and ToDo applications use this field to store the unique ID of the database record that is displayed in this table row. See `TblFindRowData`, `TblGetRowData`, and `TblSetRowData`.
`reserved1`	Reserved for future use.

`usable`	If `false`, the row is not considered part of the current interface of the application, and it doesn't appear on screen. Table rows have `usable` set to `false` when they are scrolled off the screen. See `TblRowUsable` and `TblSetRowUsable`. The function `TblGetLastUsableRow` returns the row that appears at the bottom of the screen.
`masked`	If `true` and the item's column also has a `masked` attribute of `true`, the table cell is drawn on the screen but is shaded to obscure the information that it contains. See `TblSetRowMasked` and `TblRowMasked`.
`reserved2`	Reserved for future use.
`invalid`	If `true`, the row needs to be redrawn. See `TblRowInvalid`, `TblMarkRowInvalid`, and `TblMarkTableInvalid`.
`staticHeight`	`true` if the row height cannot be changed, `false` otherwise. If `false`, text fields in this table row will dynamically resize to multiple lines as necessary. See `TblSetRowStaticHeight`.
`selectable`	If `true`, the user can select individual cells in this row. See `TblSetRowSelectable` and `TblRowSelectable`.
`reserved3`	Reserved for future use.

Compatibility

The `masked` field is defined only if 3.5 New Feature Set is present.

TableType

The `TableType` structure represents a table.

```
typedef struct TableType {
  UInt16                  id;
  RectangleType           bounds;
  TableAttrType           attr;
  Int16                   numColumns;
  Int16                   numRows;
  Int16                   currentRow;
  Int16                   currentColumn;
  Int16                   topRow;
  TableColumnAttrType *   columnAttrs;
  TableRowAttrType *      rowAttrs;
  TableItemPtr            items;
  FieldType               currentField;
} TableType;
```

Your code should treat the `TableType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Field Descriptions

`id`	ID value you specified when you created the table resource. This ID is included as part of the event data of `tblEnterEvent`.
`bounds`	Position and size of the table object. The functions `TblGetBounds`, `FrmGetObjectBounds`, `TblSetBounds`, and `FrmSetObjectBounds` retrieve and set this value.
`attr`	The table's attributes. See `TableAttrType`.
`numColumns`	Number of columns displayed by the table object. You specify the number of columns when you create the table resource, and this value cannot be changed.

`numRows`	Maximum number of visible rows in the table object. You specify this value when you create the table resource, and it does not change; however, the total number of rows in a table can change if you insert new rows in a table, and even the number of currently visible rows can change if a text field within a table cell is dynamically resized. The function `TblGetNumberOfRows` returns the value of this field.
`currentRow`	Row index of the currently selected table cell. Rows are numbered from top to bottom starting with 0.
`currentColumn`	Column index of the currently selected table cell. Columns are numbered from left to right starting with 0. If the `TableAttrType` `selected` is `true`, then this table cell is highlighted. If `selected` is `false`, the table still considers this the "current" item, but it is not highlighted. The functions `TblGetSelection` and `TblSelectItem` retrieve and set the values of `currentRow` and `currentColumn`.
`topRow`	First visible row of the table object.
`columnAttrs`	An array of each table column's attributes. See `TableColumnAttrType`.
`rowAttrs`	An array of each row's attributes, such as its ID, height, and whether or not it is usable, selectable, or invalid. See `TableRowAttrType`.

`items`	An array of each item's (table cell's) attributes, such as the item type, font ID, an integer value, and a character pointer. See `TableItemType`.
`currentField`	Field object the user is currently editing. The function `TblGetCurrentField` retrieves the value of this item.

Table Constants

Constant	Value	Description
`tableDefaultColumnSpacing`	1	Never used.
`tableNoteIndicatorHeight`	11	The height in pixels of the note indicator for tables items of type `textWithNoteTableItem`.
`tableNoteIndicatorWidth`	7	The width in pixels of the note indicator for tables items of type `textWithNoteTableItem`.
`tableMaxTextItemSize`	255	The maximum length of an editable text field within a table cell.
`tblUnusableRow`	0xffff	Value returned by `TblGetLastUsableRow` if none of the table's rows are usable. This value is only available in version 3.5 and higher.

Table Resource

The Table Resource (tTBL) represents a table on screen.

Table Functions

TblDrawTable

Purpose Draw a table.

Prototype

```
void TblDrawTable (TableType *tableP)
```

Parameters

-> `tableP` Pointer to a table object. (See `TableType`.)

Result Returns nothing.

Comments This function is called as part of `FrmDrawForm` when the form contains a table object.

This function draws the entire table, marking all rows valid before drawing. See the `TableItemType` struct description for more information about how each type of table cell is drawn.

When drawing cells with editable text fields (`textTableItem`, `textWithNoteTableItem`, or `narrowTextTableItem`), this function uses the `TableLoadDataFuncType` callback function to load the text into the table cells. The text field does not retain the text handle that your `TableLoadDataFunc` returns, meaning that you are responsible for freeing the memory that you load into the table.

When drawing `narrowTextTableItem` cells or `customTableItem` cells, this function uses the `TableDrawItemFuncType` callback function to draw the extra pixels after the text or to draw the entire cell.

On color systems, tables are always drawn using the same color as is used for the field background color.

When the user has set the security setting to mask private records, table cells that contain private database records are drawn as shaded cells to obscure the information they contain. You must explicitly tell the table which cells are masked by making the appropriate calls to `TblSetRowMasked` and `TblSetColumnMasked`.

Compatibility Color support and masked private records are only supported in Palm OS® version 3.5 and higher.

In versions earlier than 3.5, this function did not erase table cells before it drew them. In earlier releases, consider calling `TblEraseTable` before calling this function, particularly if the entire table has changed, as the visual effect of drawing over a white background may be more pleasing.

See Also `TblEraseTable`, `TblRedrawTable`, `TblSetCustomDrawProcedure`

TblEditing

Purpose Check whether a table is in edit mode.

Prototype

```
Boolean TblEditing (const TableType *tableP)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)

Result Returns `true` if the table is in edit mode, `false` otherwise.

Comments The table is in edit mode while the user edits a text item. More specifically, the table is in edit mode when a `tblEnterEvent` is received on an editable table cell (`textTableItem`, `textWithNoteTableItem`, or `narrowTextTableItem`), or when `TblGrabFocus` is called.

The table is taken out of edit mode when a the user places the pen on a note in a `textWithNoteTableItem` or when the table releases the focus (`TblReleaseFocus`).

TblEraseTable

Purpose Erase a table object.

Prototype
```
void TblEraseTable (TableType *tableP)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)

Result Returns nothing.

Comments This function sets the table's `visible` and `selected` attributes to `false`. It does not invalidate table rows.

See Also `TblDrawTable`, `TblSetCustomDrawProcedure`, `TblRedrawTable`

TblFindRowData

Purpose Return the number of the row that contains the specified data value.

Prototype
```
Boolean TblFindRowData (const TableType *tableP,
UInt32 data, Int16 *rowP)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `data`	Row data to find.
<- `rowP`	Pointer to the row number (return value).

Result Returns `true` if a match was found, `false` otherwise.

Comments This function searches for a row whose attributes contain the specified data. The data is any application-specific data that you have set with `TblSetRowData`.

See Also `TblGetRowData`, `TblFindRowID`, `TableRowAttrType`

TblFindRowID

Purpose Return the number of the row with the specified ID.

Prototype

```
Boolean TblFindRowID (const TableType *tableP,
UInt16 id, Int16 *rowP)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `id`	Row ID to find.
<- `rowP`	Pointer to the row number (return value).

Result Returns `true` if a match was found, `false` otherwise.

See Also `TblSetRowID`, `TblFindRowData`, `TableRowAttrType`

TblGetBounds

Purpose Return the bounds of a table.

Prototype

```
void TblGetBounds (const TableType *tableP,
RectangleType *r)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
<- `r`	A `RectangleType` structure in which the bounds are returned.

Result Returns nothing. The `r` parameter contains the bounds.

See Also `TblGetItemBounds`

TblGetColumnSpacing

Purpose Return the spacing after the specified column.

Prototype

```
Coord TblGetColumnSpacing
(const TableType *tableP, Int16 column)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number (zero-based).

Result Returns the spacing after column (in pixels).

This function may display a fatal error message if the `column` parameter is invalid.

See Also `TblGetColumnWidth`, `TblSetColumnSpacing`, `TblSetColumnUsable`

TblGetColumnWidth

Purpose Return the width of the specified column.

Prototype

```
Coord TblGetColumnWidth (const TableType *tableP,
Int16 column)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number (zero-based).

Result Returns the width of a column (in pixels). This function may display a fatal error message if the `column` parameter is invalid.

See Also `TblGetColumnSpacing`, `TblSetColumnWidth`, `TblSetColumnUsable`

TblGetCurrentField

Purpose Return a pointer to the FieldType in which the user is currently editing a text item.

Prototype

```
FieldPtr TblGetCurrentField
(const TableType *tableP)
```

Parameters

-> tableP	Pointer to a table object. (See TableType.)

Result Returns a pointer to the currently selected field, or NULL if the table is not in edit mode.

See Also TblGetSelection

TblGetItemBounds

Purpose Return the bounds of an item in a table.

Prototype

```
void TblGetItemBounds (const TableType *tableP,
Int16 row, Int16 column, RectangleType *r)
```

Parameters

-> tableP	Pointer to a table object. (See TableType.)
-> row	Row number of the item (zero-based).
-> column	Column number of the item (zero-based).
<- r	Pointer to a structure that holds the bounds of the item.

Result Returns nothing. Stores the bounds in r. This function may raise a fatal exception if the row or column parameter specifies a row or column that does not appear on screen.

TblGetItemFont

Purpose Return the font used to display a table item.

Prototype

```
FontID TblGetItemFont (const TableType *tableP,
Int16 row, Int16 column)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the item (zero-based).
-> `column`	Column number of the item (zero-based).

Result Returns the ID of the font used for the table item at the row and column indicated. This function may display a fatal error message if the `row` or `column` parameter specifies a row or column that is not on the screen.

Comments This function returns the value stored in the `fontID` field for this table item. Only certain types of table items use the font specified by the `fontID` field when they are displayed. The `TableItemType` description specifies what font is used to display each type of table item.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `TblSetItemFont`

TblGetItemInt

Purpose Return the integer value stored in a table item.

Prototype

```
Int16 TblGetItemInt (const TableType *tableP,
Int16 row, Int16 column)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the item (zero-based).

-> `column`	Column number of the item (zero-based).

Result Returns the integer value. This function may display a fatal message if the `row` or `column` does not appear on the screen.

Comments This function returns the value stored in the `intValue` field for this table item. Certain types of table items display the value stored in `intValue`, and other types display the value pointed to by the `ptr` field. See the `TableItemType` description for details. If the `intValue` was never set for this table item, this function returns 0.

See Also `TblSetItemInt`, `TblGetItemPtr`

TblGetItemPtr

Purpose Return the pointer value stored in a table item

Prototype

```
void * TblGetItemPtr (const TableType *tableP,
Int16 row, Int16 column)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the item (zero-based).
-> `column`	Column number of the item (zero-based).

Result Returns the item's pointer value or `NULL` if the item does not have a pointer value. This function may display a fatal message if the `row` or `column` parameter is invalid.

Comments This function returns the value stored in the `ptr` field for this table item. Certain types of table items display the value pointed to by the `ptr`, and other types display the value stored in the `intValue` field. See the `TableItemType` description for details. An application may have set the value of the `ptr` field anyway, even for items that use the `intValue`. This function always returns that value.

Compatibility Implemented only if 3.5 New Feature Set is present. In earlier versions, you can implement this function using the following code:

```
return tableP->items[row * tableP->numColumns +
  column].ptr;
```

See Also TblSetItemPtr

TblGetLastUsableRow

Purpose Return the last row in a table that is usable (visible).

Prototype

```
Int16 TblGetLastUsableRow
(const TableType *tableP)
```

Parameters

-> `tableP`	Pointer to a table object. (See TableType.)

Result Returns the row index (zero-based) or `tblUnusableRow` if there are no usable rows.

See Also TblGetRowData, TblGetRowID

TblGetNumberOfRows

Purpose Return the number of rows in a table.

Prototype

```
Int16 TblGetNumberOfRows (const TableType *tableP)
```

Parameters

-> `tableP`	Pointer to a table object. (See TableType.)

Result Returns the maximum number of visible rows in the specified table.

Comments Note that even though you can add and remove rows to and from a table, the value returned by this function does not change. The value returned by this function indicates the maximum number of rows that can be displayed on the screen at one time. It is set when you create the table resource.

TblGetRowData

Purpose Return the data value of the specified row.

Prototype
```
UInt32 TblGetRowData (const TableType *tableP,
Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Number of the row (zero-based).

Result Returns the application-specific data stored for this row, if any. Returns 0 if there is no application-specific data value.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblFindRowData`, `TblSetRowData`, `TableRowAttrType`

TblGetRowHeight

Purpose Return the height of the specified row.

Prototype
```
Coord TblGetRowHeight (const TableType *tableP,
Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Number of the row (zero-based).

Result Returns the height in pixels. This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblGetItemBounds`, `TblSetRowHeight`

TblGetRowID

Purpose Return the ID value of the specified row.

Prototype

```
UInt16 TblGetRowID (const TableType *tableP,
Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Number of the row (zero-based).

Result Returns the ID value of the row in the table.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblGetRowData`, `TblSetRowID`, `TblFindRowID`, `TableRowAttrType`

TblGetSelection

Purpose Return the row and column of the currently selected table item.

Prototype

```
Boolean TblGetSelection (const TableType *tableP,
Int16 *rowP, Int16 *columnP)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
<- `rowP, columnP`	The row and column indexes (zero-based) of the currently selected item.

Result Returns `true` if the item is highlighted, `false` if not.

See Also `TblSetRowSelectable`

TblGrabFocus

Purpose Put a table into edit mode.

Prototype

```
void TblGrabFocus (TableType *tableP, Int16 row,
Int16 column)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Current row to be edited (zero-based).
-> `column`	Current column to be edited (zero-based).

Result Returns nothing. This function may display a fatal error message if the table already has the focus or if the `row` or `column` parameter is invalid.

Comments This function puts the table into edit mode and sets the current item to the one at the row and column passed in. An editable field must exist in the coordinates passed to this function.

You must call `FrmSetFocus` before calling this function. `FrmSetFocus` releases the focus from the object that previously had it and sets the form's internal structures. After calling this function, you must call `FldGrabFocus` to display the insertion point in the field. (You can use `TblGetCurrentField` to obtain a pointer to the field.)

For example, the following function from the Address Book application sets the focus in an editable field within a table:

```
static void EditViewRestoreEditState () {
  Int16     row;
  FormPtr   frm;
  TablePtr  table;
  FieldPtr  fld;

  if (CurrentFieldIndex == noFieldIndex)
    return;

  // Find the row that the current field is in.
  table = GetObjectPtr (EditTable);
```

```
    if ( ! TblFindRowID (table,
      CurrentFieldIndex, &row) )
      return;

    frm = FrmGetActiveForm ();
    FrmSetFocus (frm, FrmGetObjectIndex (frm,
      EditTable));
    TblGrabFocus (table, row, editDataColumn);

    // Restore the insertion point position.
    fld = TblGetCurrentField (table);
    FldSetInsPtPosition (fld, EditFieldPosition);
    FldGrabFocus (fld);
  }
```

See Also TblReleaseFocus

TblHandleEvent

Purpose Handle an event for the table.

Prototype

```
Boolean TblHandleEvent (TableType *tableP,
EventType *event)
```

Parameters

-> `tableP`	Pointer to a table object. (See TableType.)
-> `event`	The event to be handled.

Result Returns `true` if the event was handled, `false` if it was not.

Comments Returns `false` if the table is not an editable table.

If the table is editable, this function passes along any keyDownEvent, fldEnterEvent, or menuCmdBarOpenEvent to the currently selected field.

When a fldHeightChangedEvent occurs, this function changes the height of the specified field as indicated by the event. If the field being resized is going to scroll off the bottom of the screen, then instead the table scrolls the rows above it up off the top. Otherwise,

the table is scrolled downward and rows below the current row are scrolled off the bottom as necessary.

Note that the `fldHeightChangedEvent` is only handled for dynamically sized fields. See the descriptions of `FieldAttrType` and `FldMakeFullyVisible` for more information.

When a `penDownEvent` occurs, the table checks to see if the focus is being changed. If it is and the user was previously editing a text field within the table, it saves the data in the table cell using the `TableSaveDataFuncType` callback function, then it enqueues a `tblEnterEvent` with the new row and column that are selected.

When a `tblEnterEvent` occurs, this function tracks the pen until it is lifted. If the pen is lifted within the bounds of the same item it went down in, a `tblSelectEvent` is added to the event queue; if not, a `tblExitEvent` is added to the event queue.

TblHasScrollBar

Purpose Set the `hasScrollBar` attribute in the table. (See `TableAttrType`.)

Prototype

```
void TblHasScrollBar (TableType *tableP,
Boolean hasScrollBar)
```

Parameters

-> `tableP` Pointer to a table object. (See `TableType`.)

-> `hasScrollBar` `true` to set the attribute, `false` to unset it.

Result Returns nothing.

Comments Your application must scroll the table itself and keep track of the scroll values. See the `ListViewUpdateScrollers` function in the Memo example application (`MemoMain.c`) for an example of setting scroll bar values for a table.

Compatibility Implemented only if 2.0 New Feature Set is present.

TblInsertRow

Purpose Insert a row into the table before the specified row.

Prototype
```
void TblInsertRow (TableType *tableP, Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row to insert (zero-based).

Result Returns nothing.

Comments The number of rows in a table is the maximum number of rows displayed on the screen. Unlike a multi-line text field, there is no notion of a table that is bigger than the available screen. For this reason, this function does not increase the number of table rows.

Instead of keeping track of a total number of rows in the table and a number of rows displayed on the screen, tables mark any row that isn't currently displayed with a usable value of `false`. (See `TableRowAttrType`.)

The newly inserted row is marked as invalid, unusable, and not masked. If you want to display the newly inserted row, call `TblSetRowUsable` after making sure that the row displays a value and then call `TblRedrawTable` when you are ready to draw the table.

See Also `TblRemoveRow`, `TblSetRowUsable`, `TblSetRowSelectable`

TblMarkRowInvalid

Purpose Mark the row invalid.

Prototype
```
void TblMarkRowInvalid (TableType *tableP,
Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)

-> `row` Row number (zero-based).

Result Returns nothing.

Comments After calling this function, call `TblRedrawTable` to redraw all rows marked invalid.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblRemoveRow`, `TblSetRowUsable`, `TblSetRowSelectable`, `TblMarkTableInvalid`, `TblRowInvalid`, `TableRowAttrType`

TblMarkTableInvalid

Purpose Mark all the rows in a table invalid.

Prototype

```
void TblMarkTableInvalid (TableType *tableP)
```

Parameters -> `tableP` Pointer to a table object. (See `TableType`.)

Result Returns nothing.

Comments After calling this function, you must call `TblRedrawTable` to redraw all rows.

See Also `TblEraseTable`, `TblRedrawTable`, `TableRowAttrType`

TblRedrawTable

Purpose Redraw the rows of the table that are marked invalid.

Prototype

```
void TblRedrawTable (TableType *tableP)
```

Parameters -> `tableP` Pointer to a table object. (See `TableType`.)

Result Returns nothing.

Comments This function draws the invalid rows in the table. See the TableItemType struct description for more information about how each type of table cell is drawn.

When drawing cells with editable text fields (textTableItem, textWithNoteTableItem, or narrowTextTableItem), this function uses the TableLoadDataFuncType callback function to load the text into the table cells. The text field does not retain the text handle that your TableLoadDataFunc returns, meaning that you are responsible for freeing the memory that you load into the table.

When drawing narrowTextTableItem cells or customTableItem cells, this function uses the TableDrawItemFuncType callback function to draw the extra pixels after the text or to draw the entire cell.

On color systems, tables are always drawn using the same color as is used for the field background color.

When the user has set the security setting to mask private records, table cells that contain private database records are drawn as shaded cells to obscure the information they contain. You must explicitly tell the table which cells are masked by making the appropriate calls to TblSetRowMasked and TblSetColumnMasked.

Compatibility Color support and masked private records are only supported in Palm OS version 3.5 and higher.

See Also TblMarkTableInvalid, TblMarkRowInvalid, TblDrawTable

TblReleaseFocus

Purpose Release the focus.

Prototype

```
void TblReleaseFocus (TableType *tableP)
```

Parameters -> tableP Pointer to a table object.

Result Returns nothing.

Comments You typically do not call this function yourself. Instead, call FrmSetFocus with an object index of noFocus to notify the form that the table has lost focus. The form code calls TblReleaseFocus for you.

If the current item is a text item, the TableSaveDataFuncType callback function is called to save the text in the currently selected field, the memory allocated for editing is released, and the insertion point is turned off.

Also note that you might have to call FldReleaseFocus if the focus is in an editable text field and that field uses a custom drawing function (TableDrawItemFuncType).

See Also TblGrabFocus

TblRemoveRow

Purpose Remove the specified row from the table.

Prototype

```
void TblRemoveRow (TableType *tableP, Int16 row)
```

Parameters

-> tableP	Pointer to a table object. (See TableType.)
-> row	Row to remove (zero-based).

Result Returns nothing. This function may raise a fatal error message if an invalid row is specified.

Comments The number of rows in the table is not decreased; instead, this row is moved from its current spot to the end of the table and is marked unusable so that it won't be displayed when the table is redrawn.

This function does not visually update the display. To update the display, call TblRedrawTable.

See Also TblInsertRow, TblSetRowUsable, TblSetRowSelectable, TblMarkRowInvalid

TblRowInvalid

Purpose Return whether a row is invalid.

Prototype
```
Boolean TblRowInvalid (const TableType *tableP,
Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).

Result Returns `true` if the row is invalid, `false` if it's valid. This function may raise a fatal error message if the `row` parameter is invalid.

Comments Invalid rows need to be redrawn. Use `TblRedrawTable` to do so.

See Also `TblMarkRowInvalid`, `TblMarkTableInvalid`

TblRowMasked

Purpose Return whether a row is masked.

Prototype
```
Boolean TblRowMasked (const TableType * tableP,
Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).

Result Returns `true` if the row is masked, `false` otherwise.

Comments Your code should set a row to masked if it contains a private database record and the user has set the display preference for private records to masked. Masked cells are displayed as shaded.

Note that a table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `TblSetRowMasked`, `TblSetColumnMasked`, `TableRowAttrType`, `SecSelectViewStatus`

TblRowSelectable

Purpose Return whether the specified row is selectable.

Prototype

```
Boolean TblRowSelectable (const TableType *tableP,
Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).

Result Returns `true` if the row is selectable, `false` if it's not.

Comments Rows that are not selectable don't highlight when touched.

See Also `TableRowAttrType`

TblRowUsable

Purpose Determine whether the specified row is usable.

Prototype

```
Boolean TblRowUsable (const TableType *tableP,
Int16 row)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).

Result Returns `true` if the row is usable, `false` if it's not.

This function may display a fatal error message if the `column` parameter is invalid.

Comments Rows that are not usable do not display.

See Also TblRowSelectable, TblGetLastUsableRow, TblSetRowUsable

TblSelectItem

Purpose Select (highlight) the specified item. If there is already a selected item, it is unhighlighted.

Prototype

```
void TblSelectItem (TableType *tableP, Int16 row,
Int16 column)
```

Parameters

-> tableP	Pointer to a table object. (See TableType.)
-> row	Row of the item to select (zero-based).
-> column	Column of the item to select (zero-based).

Result Returns nothing.

This function may display a fatal error message if the `column` or `row` parameter point to an item that is not on the screen.

Comments If `row` contains a masked private database record, then the item remains unselected.

This function cannot highlight an entire row; it can only highlight one cell in a row, and it always unhighlights the previously selected table cell. If you want to select an entire row, either create a table that has a single column, or write your own selection code.

If the selected item is a multi-line text field or a text field with a nonstandard height, this function only highlights the top eleven pixels. If you want a larger area highlighted, you must write your own selection code.

See Also TblRowSelectable, TblGetItemBounds, TblGetItemInt

TblSetBounds

Purpose Sets the bounds of a table.

Prototype

```
void TblSetBounds (TableType *tableP,
const RectangleType *rP)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `rP`	Pointer to a `RectangleType` structure that specifies the bounds for the table.

Result Returns nothing.

Compatibility Implemented only if 2.0 New Feature Set is present.

TblSetColumnEditIndicator

Purpose Set the column attribute that controls whether a column highlights when the table is in edit mode.

Prototype

```
void TblSetColumnEditIndicator (TableType *tableP,
Int16 column, Boolean editIndicator)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number (zero based).
-> `editIndicator`	`true` to highlight, `false` to turn off highlight.

Result Returns nothing.

Comments The edit indicator controls whether the item in the column is highlighted when it is selected. By default, text field items have the `editIndicator` value of `true`, and all other table item types have an edit indicator of `false`.

When the table is drawn, only the leftmost contiguous set of items with the edit indicator set are drawn as highlighted. That is, if

columns 1, 2, and 4 have an edit indicator of `true` and column 3 has an edit indicator of `false`, only the items in column 1 and 2 are drawn as highlighted when selected. Column 4 items are not drawn as highlighted.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `TableColumnAttrType`

TblSetColumnMasked

Purpose Set whether the column is masked.

Prototype

```
void TblSetColumnMasked (TableType *tableP,
Int16 column, Boolean masked)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number (zero-based).
-> `masked`	`true` to have the column be masked, `false` otherwise.

Result Returns nothing.

Comments Masked cells are displayed as shaded. You should set a column to masked if its contents should be hidden when it contains information from a private database record and the user has set the display preference for private records to masked.

A table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

Because the number of columns is static, you only need to call this function once per column when you first set up the table. The masked attribute on the row will determine if the contents of the table cell are actually displayed as shaded.

Compatibility Implemented only if 3.5 New Feature Set if present.

See Also `TblRowMasked`, `TblSetRowMasked`, `TableColumnAttrType`, `SecSelectViewStatus`

TblSetColumnSpacing

Purpose Set the spacing after the specified `column`.

Prototype

```
void TblSetColumnSpacing (TableType *tableP,
Int16 column, Coord spacing)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number (zero-based).
-> `spacing`	Spacing after the column in pixels.

Result Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

See Also `TblSetColumnUsable`, `TableColumnAttrType`

TblSetColumnUsable

Purpose Set a column in a table to usable or unusable.

Prototype

```
void TblSetColumnUsable (TableType *tableP,
Int16 column, Boolean usable)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number (zero-based).
-> `usable`	`true` for usable or `false` for not usable.

Result Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

Comments Columns that are not usable do not display.

See Also `TblMarkRowInvalid`, `TableColumnAttrType`

TblSetColumnWidth

Purpose Set the width of the specified column.

Prototype

```
void TblSetColumnWidth (TableType *tableP,
Int16 column, Coord width)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number (zero-based).
-> `width`	Width of the column (in pixels).

Result Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

See Also `TblGetColumnWidth`, `TableColumnAttrType`

TblSetCustomDrawProcedure

Purpose Set the custom draw callback procedure for the specified column.

Prototype

```
void TblSetCustomDrawProcedure (TableType *tableP,
Int16 column, TableDrawItemFuncPtr drawCallback)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number.
-> `drawCallback`	Callback function.

Result Returns nothing.

Comments The custom draw callback function is used to draw table items with a `TableItemStyleType` of `customTableItem`. See the `TableItemType` description for more information.

This function may display a fatal error message if the `column` parameter is invalid.

See Also `TableDrawItemFuncType`, `TblDrawTable`, `TableColumnAttrType`

TblSetItemFont

Purpose Set the font used to display a table item.

Prototype

```
void TblSetItemFont (TableType *tableP, Int16 row,
Int16 column, FontID fontID)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the item (zero-based).
-> `column`	Column number of the item (zero-based).
-> `fontID`	ID of the font to be used.

Result Returns nothing.

Comments This function sets the value stored in the `fontID` field for this table item. Only certain types of table items use the font specified by the `fontID` field when they are displayed. The `TableItemType` description specifies what font is used to display each type of table item. It is not an error to set the `fontID` for a table item that does not use it.

This function may display a fatal error message if the `row` or `column` parameter specifies a row or column that is not on the screen.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `TblGetItemFont`

TblSetItemInt

Purpose Set the integer value of the specified item.

Prototype

```
void TblSetItemInt (TableType *tableP, Int16 row,
Int16 column, Int16 value)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the item (zero-based).
-> `column`	Column number of the item (zero-based).
-> `value`	Any byte value (an integer).

Result Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

Comments You typically use this function when setting up and initializing a table for the first time to set the value of each table cell.

This function sets the value stored in the `intValue` field for this table item. Certain types of table items display the value stored in `intValue`, and other types display the value pointed to by the `ptr` field. See the `TableItemType` description for details. If you set the `intValue` of an item that displays its `ptr` value, it is not an error. An application can store whatever value it wants in the `intValue` field; however, be aware that this has nothing to do with the value displayed by such a table cell.

See Also `TblGetItemInt`, `TblSetItemPtr`

TblSetItemPtr

Purpose Set the item to the specified pointer value.

Prototype
```
void TblSetItemPtr (TableType * tableP, Int16 row,
Int16 column, void *value)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the item (zero-based).
-> `column`	Column number of the item (zero-based).
-> `value`	Pointer to data to display in the table item.

Result Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

Comments This function sets the value stored in the `ptr` field for this table item. Certain types of table items display the value pointed to by `ptr`, and other types display the value stored in the `intValue` field. See the `TableItemType` description for details. If you set the `ptr` of an item that displays its `intValue`, it is not an error. An application can store whatever value it wants in the `ptr` field; however, be aware that this has nothing to do with the value displayed by such a table cell.

See Also `TblGetItemPtr`, `TblSetItemInt`

TblSetItemStyle

Purpose Set the type of item to display; for example, text, numbers, dates, and so on.

Prototype
```
void TblSetItemStyle (TableType *tableP,
Int16 row, Int16 column, TableItemStyleType type)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)

-> `row`	Row number of the item (zero-based).
-> `column`	Column number of the item (zero-based).
-> `type`	The type of item, such as an editable text field or a check box. See `TableItemType` for a list of possible values.

Result Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

Comments You typically use this function when first setting up and initializing a table; you do not dynamically change item styles.

Follow this function with a call to either `TblSetItemInt` or `TblSetItemPtr` to set the value displayed by the table item. You should call one or the other of these functions depending on the type of table item you specified. See the table in the `TableItemType` description for details.

Note also that a table column always contains items of the same type. For example, you might initialize a table using this code:

```
for (row = 0; row < rowsInTable; row++) {
  TblSetItemStyle (table, row, completedColumn,
    checkboxTableItem);
  TblSetItemStyle (table, row, priorityColumn,
    numericTableItem);
  TblSetItemStyle (table, row, descColumn,
    textTableItem);
  TblSetItemStyle (table, row, dueDateColumn,
    customTableItem);
  TblSetItemStyle (table, row, categoryColumn,
    customTableItem);
}
```

See Also `TblSetCustomDrawProcedure`

TblSetLoadDataProcedure

Purpose Set the load-data callback procedure for the specified column.

Prototype

```
void TblSetLoadDataProcedure (TableType *tableP,
Int16 column,
TableLoadDataFuncPtr loadDataCallback)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `column`	Column number (zero-based).
-> `loadDataCallback`	Callback procedure. See `TableLoadDataFuncType`.

Result Returns nothing.

Comments The callback procedure is used to load the data values of a table item. See the `TableLoadDataFuncType` for more information on writing the callback function.

You typically use this function when first setting up and initializing a table.

See Also `TblSetCustomDrawProcedure`

TblSetRowData

Purpose Set the data value of the specified row. The data value is a placeholder for application-specific values.

Prototype

```
void TblSetRowData (TableType *tableP, Int16 row,
UInt32 data)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).

-> `data`	Application-specific data value to store for this row. For example, the Datebook and ToDo applications use this field to store the unique ID of the database record displayed by this row.

Result Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblGetRowData`, `TblFindRowData`

TblSetRowHeight

Purpose Set the height of the specified row.

Prototype

```
void TblSetRowHeight (TableType *tableP,
Int16 row, Coord height)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).
-> `height`	New height in pixels.

Result Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblGetRowHeight`, `TblSetRowStaticHeight`

TblSetRowID

Purpose Set the ID value of the specified row.

Prototype

```
void TblSetRowID (TableType *tableP, Int16 row,
UInt16 id)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)

-> `row`	Row number (zero-based).
-> `id`	ID to identify a row.

Result Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblGetRowID`, `TblFindRowID`, `TableRowAttrType`

TblSetRowMasked

Purpose Set a row in a table to masked or unmasked.

Prototype

```
void TblSetRowMasked (TableType *tableP,
Int16 row, Boolean masked)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).
-> `masked`	`true` to have the row be masked, `false` otherwise.

Result Returns nothing.

Comments Masked cells are displayed as shaded. You should call this function before drawing or redrawing the table. If a table row contains a private database record and the user has set the display preference for private records to masked, you must call this function on that row. For example:

```
UInt16 attr;
privateRecordViewEnum privateRecordStatus;
Boolean masked;

privateRecordStatus = (privateRecordViewEnum)
  PrefGetPreference(prefShowPrivateRecords);
....
DmRecordInfo (ToDoDB, recordNum, &attr, NULL,
  NULL);
```

```
masked = ((attr & dmRecAttrSecret) &&
  (privateRecordStatus == maskPrivateRecords));
TblSetRowMasked(tableP, row, masked);
```

Note that a table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `TblRowMasked`, `TblSetColumnMasked`, `TableRowAttrType`, `SecSelectViewStatus`

TblSetRowSelectable

Purpose Set a row in a table to selectable or nonselectable.

Prototype

```
void TblSetRowSelectable (TableType *tableP,
Int16 row, Boolean selectable)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).
-> `selectable`	`true` or `false`.

Result Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

Comments Rows that are not selectable don't highlight when touched.

See Also `TblRowSelectable`, `TblSetRowUsable`, `TableRowAttrType`

TblSetRowStaticHeight

Purpose Set the static height attribute of a row.

Prototype

```
void TblSetRowStaticHeight (TableType *tableP,
Int16 row, Boolean staticHeight)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).
-> `staticHeight`	`true` to set the static height, `false` to unset it.

Result Nothing.

This function may display a fatal error message if the `row` parameter is invalid.

Comments A row that has its static height attribute set will not expand or contract the height of the row as text is added or removed from a text item.

Compatibility Implemented only if 2.0 New Feature Set is present.

TblSetRowUsable

Purpose Set a row in a table to usable or unusable. Rows that are not usable do not display.

Prototype

```
void TblSetRowUsable (TableType *tableP,
Int16 row, Boolean usable)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number (zero-based).
-> `usable`	`true` or `false`.

Result Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblRowUsable`, `TblSetRowSelectable`

TblSetSaveDataProcedure

Purpose Set the save-data callback procedure for the specified column.

Prototype

```
void TblSetSaveDataProcedure (TableType *tableP,
Int16 column,
TableSaveDataFuncPtr saveDataCallback)
```

Parameters

-> `tableP` Pointer to a table object. (See `TableType`.)

-> `column` Column number (zero-based).

-> `saveDataCallback` Callback function. See `TableSaveDataFuncType`.

Result Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

Comments The callback procedure is called when the table object determines the data of a text object needs to be saved.

See Also `TblSetCustomDrawProcedure`

TblUnhighlightSelection

Purpose Unhighlight the currently selected item in a table.

Prototype

```
void TblUnhighlightSelection (TableType *tableP)
```

Parameters

-> `tableP` Pointer to a table object. (See `TableType`.)

Result Returns nothing.

Application-Defined Functions

TableDrawItemFuncType

Purpose Draw a custom table item.

Prototype

```
void TableDrawItemFuncType (void *tableP,
Int16 row, Int16 column, RectangleType *bounds)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the item to be drawn (zero-based).
-> `column`	Column number of the item to be drawn (zero-based).
-> `bounds`	The area of the screen in which the item is to be drawn.

Result Returns nothing.

Comments This function is called during `TblDrawTable` and `TblRedrawTable`.

You implement a custom drawing function when your table contains items of type `customTableItem` (to draw the entire item) or `narrowTextTableItem` (to draw whatever is required in the space between the text and the right edge of the table cell).

You may implement a custom drawing function to include any style of information in the table. If you don't like the way a predefined item is drawn, you may prefer to use a `customTableItem` instead. For example, if you want to include a date in your table but you want it to show the year as well as the month and day, you should implement a custom drawing function.

See Also `TblSetCustomDrawProcedure`, `TableItemType`

TableLoadDataFuncType

Purpose Load data into a column.

Prototype

```
Err TableLoadDataFuncType (void *tableP,
Int16 row, Int16 column, Boolean editable,
MemHandle *dataH, Int16 *dataOffset,
Int16 *dataSize, FieldPtr fld)
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the table item to load.
-> `column`	Column number of the table item to load.
-> `editable`	If `true`, the table is currently being edited. If `false`, the table is being drawn but not necessarily being edited.
<- `dataH`	Unlocked handle of a block containing a null-terminated text string.
<- `dataOffset`	Offset from start of block to start of the text string.
<- `dataSize`	Allocated size of text string, **not** the string length.
-> `fld`	Pointer to the text field in this table cell.

Result Returns 0 upon success or an error if unsuccessful.

Comments This function is called in two cases: when a text field item is being drawn (`TblDrawTable` or `TblRedrawTable`) and when a text field item is being selected (part of `TblHandleEvent`'s handling of `tblEnterEvent`). If this function returns an error (any nonzero value) and the item is being selected, then the item is not selected and the table's `editing` attribute is set to `false`.

You return the same values for `dataH`, `dataOffset`, and `dataSize` that you would pass to `FldSetText`. That is, you can use this function to point the table cell's text field to a string in a database record so that you can edit that string directly using text field routines. To do so, return the handle to a database record in

`dataH`, the offset from the start of the record to the start of the string in `dataOffset`, and the allocated size of the string in `dataSize`.

The handle that you return from this function is assumed to contain a null-terminated string starting at `dataOffset` bytes in the memory chunk. The string should be between 0 and `dataSize` - 1 bytes in length.

As with `FldSetText`, you are responsible for freeing the memory associated with the `dataH` parameter. You can do so in the `TableSaveDataFuncType` function, but it is only called for a cell that has been edited. For non-editable text cells or text cells that are editable but were never selected, free the memory when you close the form.

The `fld` pointer passed to your function has already been initialized with default values by the table code. If you want to override a field's attributes (for example, if you want to change the underline mode) you can do so in this function.

See Also `TblDrawTable`, `TblHandleEvent`, `TableLoadDataFuncType`

TableSaveDataFuncType

Purpose Save the data associated with a text field.

Prototype

```
Boolean TableSaveDataFuncType (void *tableP,
Int16 row, Int16 column);
```

Parameters

-> `tableP`	Pointer to a table object. (See `TableType`.)
-> `row`	Row number of the table item to load.
-> `column`	Column number of the table item to load.

Result Return `true` if the table should be redrawn, or `false` if the table does not need to be redrawn.

Comments This is called before the memory associated with the currently selected text field in a table cell is freed. Implement this function if you need to do any special cleanup before this memory is freed.

This function is called only when the currently selected editable text field is releasing the focus. You can use `TblGetCurrentField` to retrieve a pointer to this field. It is called only on the currently selected field, not on any other fields in the table.

Note that the table manager already disassociates the memory handle from the text field for you so that the memory associated with your data is not freed when the field is freed. The table manager also calls `FldCompactText` for you.

If the text handle you returned in your `TableLoadDataFuncType` callback points to a string on the dynamic heap, you should implement this callback function to store or free the handle. You can use `FldGetTextHandle` to obtain the handle.

If you return `true` from this function, `TblRedrawTable` is called. You should mark invalid any table rows that you want redrawn before returning.

See Also `TblSetSaveDataProcedure`

20

UI Color List

This chapter provides information about the UI color list by discussing the following topics:

- UI Color Data Types
- UI Color Functions

The header file `UIColor.h` declares the API that this chapter describes. For more information on the color list, see "Color and Grayscale Support" on page 120 in the *Palm OS Programmer's Companion*.

UI Color Data Types

UIColorTableEntries

The `UIColorTableEntries` enum declares symbolic color constants for the various UI elements.

Do not confuse the UI color list with the system color table. The **system color table** (or **system palette**) defines all available colors for the display or draw window, whether they are in use or not. The **UI color list** defines the colors used to draw the interface objects.

```
typedef enum UIColorTableEntries {
  UIObjectFrame = 0,
  UIObjectFill,
  UIObjectForeground,
  UIObjectSelectedFill,
  UIObjectSelectedForeground,

  UIMenuFrame,
  UIMenuFill,
  UIMenuForeground,
  UIMenuSelectedFill,
  UIMenuSelectedForeground,
```

```
  UIFieldBackground,
  UIFieldText,
  UIFieldTextLines,
  UIFieldCaret,
  UIFieldTextHighlightBackground,
  UIFieldTextHighlightForeground,
  UIFieldFepRawText,
  UIFieldFepRawBackground,
  UIFieldFepConvertedText,
  UIFieldFepConvertedBackground,
  UIFieldFepUnderline,

  UIFormFrame,
  UIFormFill,

  UIDialogFrame,
  UIDialogFill,

  UIAlertFrame,
  UIAlertFill,

  UIOK,
  UICaution,
  UIWarning,

  UILastColorTableEntry
} UIColorTableEntries;
```

Field Descriptions

`UIObjectFrame`	Color for the border of user interface objects (such as command buttons, push buttons, selector triggers, menus, arrows checkboxes, and other controls).
`UIObjectFill`	The background color for a solid or "filled" user interface object. Note that UI objects in tables use the `UIField`... colors instead of the `UIObject`... colors.
`UIObjectForeground`	The color for foreground items (such as labels or graphics) in a user interface object.
`UIObjectSelectedFill`	The background color of the currently selected user interface object, whether that object is solid or not.
`UIObjectSelectedForeground`	The color of foreground items in a selected user interface object.
`UIMenuFrame`	The color of the border around the menu.
`UIMenuFill`	The background color of a menu item.
`UIMenuForeground`	The color of the menu's text.
`UIMenuSelectedFill`	The background color of a selected menu item.
`UIMenuSelectedForeground`	The color of the text of a selected menu item.
`UIFieldBackground`	The background color of an editable text field.
`UIFieldText`	The color of the text in the editable field.
`UIFieldTextLines`	The color of underlines in an editable field.
`UIFieldCaret`	The color of the cursor in an editable text field.
`UIFieldTextHighlightBackground`	The background color for selected text in an editable text field.

`UIFieldTextHighlightForeground`	The color of the selected text in an editable text field.
`UIFieldFepRawText`	Color used for unconverted text in the inline conversion area when a FEP is used as a text input method (for example, on Japanese devices). If the FEP colors are identical to field colors, unconverted text has a solid underline.
`UIFieldFepRawBackground`	The background color for unconverted text in the inline conversion area when a FEP is used as a text input method. If the FEP colors are identical to field colors, unconverted text has a solid underline.
`UIFieldFepConvertedText`	Color used for converted text in the inline conversion area when a FEP is used as a text input method (for example, on Japanese devices). If the FEP colors are identical to field colors, converted text has a double-thick underline.
`UIFieldFepConvertedBackground`	The background color used for converted text in the inline conversion area. If the FEP colors are identical to field colors, converted text has a double-thick underline.
`UIFieldFepUnderline`	The color used for underlines in the inline conversion area.
`UIFormFrame`	The color of the border and titlebar on a form.
`UIFormFill`	The background color of a form. White is recommended for this value.
`UIDialogFrame`	The color of a border and titlebar on a modal form.
`UIDialogFill`	The background color of a modal form.

`UIAlertFrame`	The color of the border and titlebar on an alert panel.
`UIAlertFill`	The background color of an alert panel.
`UIOK`	The color for an informational icon.
`UICaution`	The color for a caution icon.
`UIWarning`	The color for a warning icon.
	Palm OS® does not currently use the `UIOK`, `UICaution`, and `UIWarning` constants.
`UILastColorTableEntry`	Placeholder to indicate end of enum.

Compatibility

Implemented only if 3.5 New Feature Set is present.

UI Color Functions

UIColorGetTableEntryIndex

Purpose Return the index value for a UI color for the current system palette.

Prototype

```
IndexedColorType UIColorGetTableEntryIndex
(UIColorTableEntries which)
```

Parameters

-> `which`	One of the symbolic color constants. See `UIColorTableEntries`.

Result Returns the system color table index of the color used for the specified symbolic color.

Comments One way to find out the indexes of all the colors that the OS is using is to loop through the UI color list, calling `UIColorGetTableEntryIndex` for each slot, and keep a list (excluding duplicates).

```
IndexedColorType
  colorsUsed[UILastColorTableEntry];
UInt16 numColors = 0;
...
for (i = 0; i < UILastColorTableEntry; i++) {
  IndexedColorType currentColor;
  Boolean isNew = true;

  currentColor = UIColorGetTableEntryIndex(i);

  for (j = 0; ((j < numColors) && isNew); j++)
    if (colorsUsed[j] == currentColor)
      isNew = false; /* exit loop */
  if (isNew) {
    numColors++;
    colorsUsed[j] = currentColor;
  }
```

To get the RGB values of the colors, do the same thing but call `UIColorGetTableEntryRGB`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `IndexedColorType`, `WinIndexToRGB`

UIColorGetTableEntryRGB

Purpose Return the RGB value for the UI color.

Prototype

```
void UIColorGetTableEntryRGB
(UIColorTableEntries which, RGBColorType *rgbP)
```

Parameters

-> `which`	One of the symbolic color constants. See `UIColorTableEntries`.
<- `rgbP`	Pointer to an RGB color value corresponding to the current color used for the symbolic color. (See `RGBColorType`.)

Result Returns nothing.

Comments In general, it is more efficient to work with indexed color entries instead of RGB color entries.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `UIColorGetTableEntryIndex`, `WinRGBToIndex`

UIColorSetTableEntry

Purpose Change a value in the UI color list.

Prototype

```
Err UIColorSetTableEntry
(UIColorTableEntries which,
const RGBColorType *rgbP)
```

Parameters

-> `which`	One of the symbolic color constants. See `UIColorTableEntries`.
-> `rgbP`	The RGB value of the color that should be used for the specified UI object. (See `RGBColorType`.)

Result Returns 0 upon success.

Comments Sets the value of a UI color entry to the passed RGB value. Updates the index for that UI color entry to the current best fit for that RGB value according to the palette used by the current draw window.

It is best to use this function only if the draw window is currently onscreen. Otherwise, the best-fit algorithm may choose a color that is not available on the current screen.

See Also `WinIndexToRGB`, `UIColorGetTableEntryIndex`, `UIColorGetTableEntryRGB`

21

UI Controls

This chapter describes the UI controls API as declared in `UIControls.h`.

UI Control Functions

UIBrightnessAdjust

Purpose Displays the brightness adjust dialog.

Prototype `void UIBrightnessAdjust()`

Parameters None

Result Returns nothing.

Comments On hardware that supports a brightness setting, this function displays a dialog that allows the user to change the brightness level. On hardware that has a backlight, this function toggles the backlight.

Compatibility Implemented only if 3.5 New Feature Set is present.

UIContrastAdjust

Purpose Displays the contrast adjust dialog (currently only available on the Palm V™ Connected Organizer).

Prototype

```
void UIContrastAdjust()
```

Parameters None.

Result Returns nothing.

Compatibility This function was renamed from `ContrastAdjust` to `UIContrastAdjust` in Palm OS® release 3.5. The `ContrastAdjust` function is available if 3.1 New Feature Set is present.

UIPickColor

Purpose Displays a dialog that allows the user to choose a color.

Prototype

```
Boolean UIPickColor (IndexedColorType *indexP,
RGBColorType *rgbP, UIPickColorStartType start,
const Char *titleP, const Char *tipP)
```

Parameters

<-> `indexP`	Index value of the selected color. (See `IndexedColorType`.) Upon entry, this points to the index value of the color initially selected. Upon return, this points to the index value of the color the user selected. Pass `NULL` to not set or return this value.
<-> `rgbP`	RGB value of the selected color. (See `RGBColorType`.) Upon entry, this points to the RGB value of the color initially selected when the dialog is displayed. Upon return, this points to the RGB value that the user selected. Pass `NULL` to not set or return this value.

-> `start`	Either `UIPickColorStartPalette` to display the system palette as a series of color squares or `UIPickColorStartRGB` to display individual sliders for the red, green, and blue values. This parameter is only used if both `indexP` and `rgbP` are not `NULL`.
-> `titleP`	String to display as the title of the dialog. Specify `NULL` to use the default title, which is "Pick Color."
-> `tipP`	Not used.

Result Returns `true` if a new color was selected, `false` otherwise.

Comments Use this function to allow users to choose a color used in your user interface. (The system never calls `UIPickColor`.)

This function can display two versions of the dialog: palette or RGB. The palette version of the dialog displays a series of squares, each containing a different color defined on the system palette. The `indexP` value contains the index of the square that is initially selected.

The RGB version of the dialog displays three sliders that allow the user to select the level of red, green, and blue in the color. The `rgbP` parameter contains the red, green, and blue values initially shown in the dialog. The sliders only allow values that are defined in the current system color table.

If `indexP` is initially `NULL`, only the RGB dialog is displayed. Similarly, if `rgbP` is `NULL`, only the palette version is displayed. If both parameters are non-`NULL`, the system adds a pull-down list that allows the user to switch between the palette dialog and the RGB dialog, and the `start` parameter controls which version of the dialog is initially shown. In this case, both `indexP` and `rgbP` contain the value of the user-selected color upon return.

Palm OS 3.5 supports a maximum of 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, the chosen RGB may not have an exact match. If this is the case, the `indexP` parameter (if not `NULL`) contains the closest match using a luminance best-fit if the color lookup table is entirely grayscale (red,

green, and blue values for each entry are identical), or a shortest-distance fit in the RGB space is used if the palette contains colors.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinSetBackColor`, `WinSetForeColor`, `WinSetTextColor`, `UIColorSetTableEntry`

22

Miscellaneous User Interface Functions

This chapter provides descriptions of miscellaneous user interface functions. You can find declarations for the functions described in this chapter in the header files `PhoneLookup.h`, and `UIResources.h`.

Miscellaneous User Interface Functions

PhoneNumberLookup

Purpose This routine calls the Address Book application to lookup a phone number.

Prototype

```
void PhoneNumberLookup (FieldType *fldP)
```

Parameters

`fldP`	Field object in which the text to match is found.

Comments When trying to match a field, this function first tries to match selected text.

- If there is some selected text, the function replaces it with the phone number if there is a match.
- If there is no selected text, the function replaces the text in which the insertion point is with the phone number if there is a match.
- If there is no match, the function displays the Address Book short list.

Result Nothing returned; it's locked.

Compatibility Implemented only if 2.0 New Feature Set is present.

ResLoadConstant

Purpose Load a constant from a `'tint'` resource and return its value.

Prototype

```
UInt32 ResLoadConstant (UInt16 rscID)
```

Parameters

-> `rscID` The ID of the `'tint'` resource (symbolically named `constantRscType`) to load.

Result The four-byte value of the constant in the resource, or 0 if the resource could not be found. The return value may be cast as necessary.

Comments Use this function to load constant values that are stored as `'tint'` resources. (All open resource databases are searched for the resource ID you specify.) You should store a constant value as a resource when its value changes depending on the locale.

As an example, consider the maximum length of the Alarm Sound trigger label in the Datebook application's preferences panel. The list displayed by this trigger uses the localized name for each sound stored in the system. Because localized names are used, the maximum length that the Datebook application allows for the label differs depending on the current locale. The maximum length is stored as a resource constant so that each overlay database can specify a different value for the constant.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `DmGetResource`, `DmGet1Resource`

ResLoadForm

Purpose Copy and initialize a form resource. The structures are complete except pointers updating. Pointers are stored as offsets from the beginning of the form.

Prototype

```
void* ResLoadForm (UInt16 rscID)
```

Parameters

`rscID`	The resource ID of the form.

Result The handle of the memory block that the form is in, since the form structure begins with the `WindowType`, this is also a `WinHandle`.

ResLoadMenu

Purpose Copy and initialize a menu resource. The structures are complete except pointers updating. Pointers are stored as offsets from the beginning of the menu.

Prototype

```
void* ResLoadMenu (UInt16 rscID)
```

Parameters

`rscID`	The resource ID of the menu.

Result The handle of the memory block that the form is in, since the form structure begins with the `WindowType` this is also a `WinHandle`.

Part II: System Management

23

Alarm Manager

This chapter provides reference material for the alarm manager:

- Alarm Manager Functions
- Application-Defined Functions

The alarm manager API is declared in the header file `AlarmMgr.h`.

For more information on the alarm manager, see the section "Alarms" in the *Palm OS Programmer's Companion*.

Alarm Manager Functions

AlmGetAlarm

Purpose Return the date and time for the application's alarm, if one is set.

Prototype

```
UInt32 AlmGetAlarm (UInt16 cardNo, LocalID dbID,
UInt32* refP)
```

Parameters

-> `cardNo`	Number of the storage card on which the application resides.
-> `dbID`	Local ID of the application.
<- `refP`	The alarm's reference value is returned here. This value is passed to the application when the alarm is triggered.

Result The date and time the alarm will trigger, given in seconds since 1/1/1904; if no alarm is active for the application, 0 is returned for the alarm seconds and the reference value is undefined.

See Also `AlmSetAlarm`

AlmGetProcAlarm

Purpose Macro that returns the date and time that a procedure alarm will trigger. Also returns the caller-defined alarm reference value.

Prototype

```
AlmGetProcAlarm (procP, refP)
```

Parameters

-> `procP`	Pointer to a function that will be called when alarm is triggered. See `AlmAlarmProcPtr`.
<- `refP`	A `UInt32` pointer to a location where the alarm's reference value is returned. This value is passed to the procedure when the alarm is triggered.

Result The date and time the alarm will trigger, given in seconds since 1/1/1904; if no alarm is active for the procedure, 0 is returned for the alarm seconds and the reference value is undefined.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `AlmSetProcAlarm`

AlmSetAlarm

Purpose Set or cancel an alarm for the given application.

Prototype

```
Err AlmSetAlarm (UInt16 cardNo, LocalID dbID,
UInt32 ref, UInt32 alarmSeconds, Boolean quiet)
```

Parameters

-> `cardNo`	Number of the storage card on which the application resides.
-> `dbID`	Local ID of the application.
-> `ref`	Caller-defined value. This value is passed with the launch code that notifies the application that the alarm has been triggered.
-> `alarmSeconds`	Alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).

	-> `quiet`	Reserved for future upgrade. This value is not currently used.

Result

`0`	No error.
`almErrMemory`	Insufficient memory.
`almErrFull`	Alarm table is full.

Comments

This function sets an alarm for the specified application. An application can have only one alarm set at a time. If an alarm for this application has already been set, it is replaced with the new alarm.

The `alarmSeconds` parameter specifies the time at which the alarm will be triggered. As soon as possible after this time, the alarm manager sends the `sysAppLaunchCmdAlarmTriggered` launch code to the specified application. If there is another alarm that should be set for this application, you can set it in response to this launch code. Following the `sysAppLaunchCmdAlarmTriggered` launch code, the alarm manager sends a `sysAppLaunchCmdDisplayAlarm` launch code. This is where your application should do things such as display a modal dialog indicating that the event has occurred. Read more about these launch codes in Chapter 1, "Application Launch Codes."

If your application needs access to any particular value to respond to the alarm, pass a pointer to that value in the `ref` parameter. The system will pass this pointer back to the application in the launch codes' parameter blocks.

See Also `AlmGetAlarm`

AlmSetProcAlarm

Purpose Macro that sets or cancels a procedure alarm.

Prototype

```
AlmSetProcAlarm (procP, ref, alarmSeconds)
```

Parameters

-> `procP`	Pointer to a function that should be called when alarm is triggered. See `AlmAlarmProcPtr`.

-> `ref`	A caller-defined `UInt32` value. This value is passed with the launch code that notifies the application that the alarm has been triggered.
-> `alarmSeconds`	A `UInt32` indicating the alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).

Result Returns one of the following error codes:

`0`	No error.
`almErrMemory`	Insufficient memory.
`almErrFull`	Alarm table is full.

Comments This macro is similar to the `AlmSetAlarm` function, but it specifies a procedure to be called at the specified date and time rather than an application to be launched. With this macro, you can set alarms that are independent of any application. For example, a shared library can set procedure alarms that call a procedure implemented in the library.

Procedure alarms also differ from regular system alarms in that if they trigger when the device is in sleep mode, the LCD does not light up. Thus, you can use procedure alarms to perform a scheduled task in a manner that is entirely hidden from the user.

IMPORTANT: Because the `procP` pointer is used to directly call the procedure, the pointer must remain valid from the time `AlmSetProcAlarm` is called to the time the alarm is triggered. If the procedure is in a shared library, you must keep the library open. If the procedure is in a separately loaded code resource, the resource must remain locked until the alarm is triggered. When you close a library or unlock a resource, you must remove any pending alarms. If you don't, the system will crash when the alarm is triggered.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `AlmGetProcAlarm`

Application-Defined Functions

AlmAlarmProcPtr

Purpose A procedure to be executed when an alarm is triggered.

Prototype

```
void (*AlmAlarmProcPtr) (UInt16 almProcCmd,
SysAlarmTriggeredParamType *paramP)
```

Parameters

`-> almProcCmd` One of the `AlmProcCmdEnum` constants. These are commands that your function must handle. Possible values are:

`almProcCmdTriggered`
The alarm's date and time has passed and the alarm has been triggered. The function should perform its main task in response to this command.

`almProcCmdReschedule`
A system time change occurred, so the function must reschedule the alarm.

`-> paramP` Pointer to a `SysAlarmTriggeredParamType` structure. See below.

Result Returns nothing.

Comments `AlmAlarmProcPtr` procedures are called when an alarm set by `AlmSetProcAlarm` is triggered. Your implementation should check the value of `almProcCmd` and respond accordingly.

The `paramP` parameter is a pointer to a `SysAlarmTriggeredParamType` structure. This structure is defined as:

```
typedef struct SysAlarmTriggeredParamType {
  UInt32  ref;
  UInt32  alarmSeconds;
  Boolean purgeAlarm;
} SysAlarmTriggeredParamType;
```

`ref` and `alarmSeconds` are both values specified in `AlmSetProcAlarm` when the alarm is set. The `purgeAlarm` field specifies if the alarm will be removed from the alarm table when the function returns so that the `sysAppLaunchCmdDisplayAlarm` launch code is not triggered. This should be `true` for all procedure alarms; the alarm manager set it to `true` for you after your function returns.

If necessary, you can define new values for the `almProcCmd` parameter to call the procedure for something other than a triggered alarm or a system time change. The value you use must be greater than the constant `almProcCmdCustom` as defined in `AlarmMgr.h`.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `AlmGetProcAlarm`

24

Bitmaps

This chapter provides information about bitmaps by discussing these topics:

- Bitmap Data Structures
- Bitmap Constants
- Bitmap Resources
- Bitmap Functions

The header file `Bitmap.h` declares the API that this chapter describes. For more information on windows, see the section "Bitmaps" on page 109 in the *Palm OS Programmer's Companion*.

Bitmap Data Structures

BitmapCompressionType

The `BitmapCompressionType` enum specifies possible bitmap compression types. These are the possible values for the `compressionType` field of `BitmapType`. You can compress or uncompress a bitmap using a call to `BmpCompress`.

```
typedef enum {
  BitmapCompressionTypeScanLine = 0,
  BitmapCompressionTypeRLE,
  BitmapCompressionTypeNone = 0xFF
} BitmapCompressionType;
```

Value Descriptions

`BitmapCompressionTypeScanLine`	Use scan line compression. Scan line compression is compatible with Palm OS® 2.0 and higher.
`BitmapCompressionTypeRLE`	Use RLE compression. RLE compression is supported in Palm OS 3.5 only.
`BitmapCompressionTypeNone`	No compression is used. This value should only be used as an argument to `BmpCompress`.

Compatibility

This type is only defined if 3.5 New Feature Set is present. Earlier releases do support compressed bitmaps, but in scan line format only.

BitmapFlagsType

The `BitmapFlagsType` bit field defines the `flags` field of `BitmapType`. It specifies the bitmap's attributes.

```
typedef struct BitmapFlagsType {
  UInt16 compressed:1;
  UInt16 hasColorTable:1;
  UInt16 hasTransparency:1;
  UInt16 indirect:1;
  UInt16 forScreen:1;
  UInt16 reserved:11;
} BitmapFlagsType;
```

Field Descriptions

`compressed`	If `true`, the bitmap is compressed and the `compressionType` field specifies the compression used. If `false`, the bitmap is uncompressed. The `BmpCompress` function sets this field.
`hasColorTable`	If `true`, the bitmap has its own color table. If `false`, the bitmap uses the system color table. You specify whether the bitmap has its own color table when you create the bitmap.
`hasTransparency`	If `true`, the OS will not draw pixels that have a value equal to the `transparentIndex`. If `false`, the bitmap has no transparency value. You specify the transparent color when you create the bitmap using Constructor.
`indirect`	If `true`, the address to the bitmap's data is stored where the bitmap itself would normally be stored. The actual bitmap data is stored elsewhere. If `false`, the bitmap data is stored directly following the bitmap header or directly following the bitmap's color table if it has one. Never set this flag. Only the display (screen) bitmap has the indirect bit set.
`forScreen`	If `true`, the bitmap is the bitmap for the display (screen) window. Never set this flag.
`reserved`	Reserved for future use.

Compatibility

All flags other than `compressed` and `hasColorTable` are only defined if 3.5 New Feature Set is present. Note that the size of this structure did not change.

BitmapPtr

The `BitmapPtr` type defines a pointer to a `BitmapType` structure.

```
typedef BitmapType *BitmapPtr;
```

BitmapType

The `BitmapType` structure represents a bitmap. This structure defines both the bitmaps representing the window display and bitmap resources (`'Tbmp'` and `'tAIB'`) that you create using Constructor or some other application and load into your program.

```
typedef struct BitmapType {
  Int16           width;
  Int16           height;
  UInt16          rowBytes;
  BitmapFlagsType flags;
  UInt8           pixelSize;
  UInt8           version;
  UInt16          nextDepthOffset;
  UInt8           transparentIndex;
  UInt8           compressionType;
  UInt16          reserved;
} BitmapType;
```

Field Descriptions

`width`	The width of the bitmap in pixels. You specify this value when you create the bitmap.
`height`	The height of the bitmap in pixels. You specify this value when you create the bitmap.
`rowBytes`	The number of bytes stored for each row of the bitmap where `height` is the number of rows.
`flags`	The bitmap's attributes. See `BitmapFlagsType`.
`pixelSize`	The pixel depth. Currently supported pixel depths are 1, 2, 4, and 8-bit. You specify this value when you create the bitmap.

`version`	The version of bitmap encoding used. See Bitmap Constants.
`nextDepthOffset`	For bitmap families, this field specifies the start of the next bitmap in the family. The value it contains is the number of 4-byte words to the next `BitmapType` from the beginning of this one. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the `nextDepthOffset` is 0.
`transparentIndex`	The color index for the transparent color. Only used for version 2 bitmaps and only when the `transparent` flag is set in `flags`. You specify this value when you create the bitmap using Constructor.
`compressionType`	The compression type used. Only used for version 2 bitmaps and only when the `compressed` flag is set in `flags`. See `BitmapCompressionType` for possible values. The `BmpCompress` function sets this field.
`reserved`	Reserved for future use. Must be set to 0.

Note the following about the `BitmapType` structure:

- None of these fields contains the actual bitmap data. Instead, the bitmap data is stored immediately following this `BitmapType` header structure. If the bitmap has its own color table, the color table is stored in between the header and the data. You can retrieve a bitmap's data by passing its `BitmapType` structure to `BmpGetBits`, and you can retrieve its color table with `BmpGetColortable`.
- Unlike most other user interface structures, the `BitmapType` does not store the bitmap's location on the screen. The `WindowType` or the `FormBitmapType` with which this bitmap is associated contains that information.
- A bitmap may be part of a **bitmap family**. A bitmap family is a group of bitmaps, each containing the same drawing but at a different pixel depth (see Figure 24.1). When requested to

draw a bitmap family, the operating system chooses the version of the bitmap with the pixel depth closest to the display. When `BitmapType` represents a bitmap family, the `nextDepthOffset` field contains the offset from the start of this bitmap to the next bitmap in the family.

Figure 24.1 Bitmap Family

Compatibility

The `transparentIndex` and `compressionType` flags are defined only if 3.5 New Feature Set is present.

ColorTableType

The `ColorTableType` structure defines a color table. Bitmaps can have color tables attached to them; however, doing so is not recommended for performance reasons.

```
typedef struct ColorTableType {
  UInt16          numEntries;
  // RGBColorType entry[];
} ColorTableType;
```

Field Descriptions

`numEntries` The number of entries in table. High bits (`numEntries` > 256) reserved.

The color table entries themselves are of type `RGBColorType`, and there is one per `numEntries`. Use the macro `ColorTableEntries` to retrieve these entries.

Care should be taken not to confuse a full color table (which includes the count) with an array of RGB color values. Some routines operate on entire color tables; others operate on lists of color entries.

Compatibility

This type is defined only if 3.5 New Feature Set is present.

RGBColorType

The `RGBColorType` structure defines a color. It is used as an entry in the color table. `RGBColorTypes` can also be created manually and passed to several user interface functions.

```
typedef struct RGBColorType {
  UInt8 index;
  UInt8 r;
  UInt8 g;
  UInt8 b;
} RGBColorType;
```

Field Descriptions

`index` The index of this color in the color table. Not all functions that use `RGBColorType` use the `index` field.

In Palm OS® 3.5, the maximum supported number of colors is 256. The number of possible RGB colors greatly exceeds this amount. For this reason, some drawing functions use a color look up table (CLUT). If the CLUT is used, the `index` field contains the index of an available color that is the closest match to the color specified by the `r`, `g`, and `b` fields.

`r` Amount of red (0 to 255).

`g` Amount of green (0 to 255).

`b` Amount of blue (0 to 255).

Compatibility

This type is defined only if 3.5 New Feature Set is present.

Bitmap Constants

Constant	Value	Description
`BitmapVersionZero`	0	Uses the version 0 encoding of a bitmap. Version 0 encoding is supported in Palm OS® 1.0 and later.

Constant	Value	Description
`BitmapVersionOne`	1	Uses the version 1 encoding of a bitmap. Version 1 encoding is supported in Palm OS 3.0 and later. PalmRez automatically creates version 1 bitmaps unless you've specified a transparency index or a compressed type when creating the bitmap in Constructor.
`BitmapVersionTwo`	2	Uses the version 2 encoding of a bitmap. Palm OS 3.5 supports version 2 bitmaps. Version 2 bitmaps either use the transparency index or are compressed. If you programmatically create a bitmap using `BmpCreate`, a version 2 bitmap is created.

Bitmap Resources

You can create a bitmap resource and include it as part of your application's PRC file. Use the resource type `'Tbmp'` for most images and the resource type `'tAIB'` for application icons. Symbolically, these two resource types are `bitmapRsc` an `iconType`, respectively.

Note that if you are creating a bitmap or a bitmap family in Constructor, you create a `'tbmf'` resource (or `'taif'` resource for icons) and one or more 'PICT' images, and the PalmRez post linker converts them into a single `'Tbmp'` or `'tAIB'` resource. Note that the PalmRez post linker takes PICT images even on the Microsoft Windows operating system.

Bitmap Functions

BmpBitsSize

Purpose Return the size of the bitmap's data.

Prototype
```
UInt16 BmpBitsSize (BitmapType *bitmapP)
```

Parameters -> `bitmapP` Pointer to the bitmap. (See `BitmapType`.)

Result Returns the size in bytes of the bitmap's data, excluding the header and the color table.

Comments This function returns the bitmap's data size even if the bitmap's `indirect` flag is set. (See `BitmapFlagsType`.)

If the bitmap is compressed, this function returns the compressed size of the bitmap.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpSize`, `BmpColortableSize`, `BmpGetBits`

BmpColortableSize

Purpose Return the size of the bitmap's color table.

Prototype
```
UInt16 BmpColortableSize (BitmapType *bitmapP)
```

Parameters -> `bitmapP` Pointer to the bitmap. (See `BitmapType`.)

Result Returns the size in bytes of the bitmap's color table or 0 if the bitmap does not use its own color table.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpBitsSize`, `BmpSize`, `BmpGetColortable`

BmpCompress

Purpose Compress or uncompress a bitmap.

Prototype

```
Err BmpCompress (BitmapType *bitmapP,
BitmapCompressionType compType)
```

Parameters

-> `bitmapP`	Pointer to the bitmap to compress. (See `BitmapType`.)
-> `compType`	The type of compression to use. (See `BitmapCompressionType`.) If set to `BitmapCompressionTypeNone` and `bitmapP` is compressed, this function uncompresses the bitmap.

Result Returns one of the following values:

`errNone`	Success.
`sysErrParamErr`	Either the `compType` parameter does not specify a compression type or the bitmap is already compressed, is in the storage heap, or represents the screen.
`memErrNotEnoughSpace`	There is not enough memory available to complete the operation.

Comments This function performs the specified compression and resizes the bitmap's allocated memory. The bitmap must be in the dynamic heap.

Compatibility Implemented only if 3.5 New Feature Set is present.

BmpCreate

Purpose Create a bitmap.

Prototype

```
BitmapType *BmpCreate (Coord width, Coord height,
UInt8 depth, ColorTableType *colortableP,
UInt16 *error)
```

Parameters

-> `width`	The width of the bitmap in pixels. Must not be 0.
-> `height`	The height of the bitmap in pixels. Must not be 0.
-> `depth`	The pixel depth of the bitmap. Must be 1, 2, 4 or 8. This value is used as the `pixelSize` field of `BitmapType`.
-> `colortableP`	A pointer to the color table associated with the bitmap, or `NULL` if the bitmap should not include a color table. If specified, the number of colors in the color table must match the `depth` parameter. (2 for 1-bit, 4 for 2-bit, 16 for 4-bit, and 256 for 8-bit).
<- `error`	Contains the error code if an error occurs.

Result Returns a pointer to the new bitmap structure (see `BitmapType`) or `NULL` if an error occurs. The parameter `error` contains one of the following:

`errNone`	Success.
`sysErrParamErr`	The `width`, `height`, `depth`, or `colorTableP` parameter is invalid. See the descriptions above for acceptable values.
`memErrNotEnoughSpace`	There is not enough memory available to allocate the structure.

Comments

This function creates an uncompressed, non-transparent `BitmapVersionTwo` bitmap with the width, height, and depth that you specify.

If you pass a color table, the bitmap's `hasColorTable` flag is set. For performance reasons, attaching a custom color table to a bitmap is strongly discouraged. An alternative is to use the `WinPalette` command to change the color table as needed, draw the bitmap, and then undo your changes after you have finished displaying the bitmap.

The newly created bitmap contains no data. To create data for this bitmap, use the window drawing functions. First, you must use `WinCreateBitmapWindow` to create a offscreen window wrapper around the bitmap, then draw to that window:

```
BitmapType *bmpP;
WinHandle win;
Err error;
RectangleType onScreenRect;

bmpP = BmpCreate(10, 10, 8, NULL, &error);
if (bmpP) {
  win = WinCreateBitmapWindow(bmpP, &error);
  if (win) {
    WinSetDrawWindow(win);
    WinDrawLines(win, ...);
    /* etc */
    WinSetWindowBounds(win, onScreenRect);
  }
}
```

You cannot use this function to create a bitmap written directly to a database; that is, you must create the bitmap on the dynamic heap first, then write it to the storage heap.

It's not necessary to use `BmpCreate` to load a bitmap stored in a resource. Instead, you simply load the resource and lock its handle. The returned pointer is a pointer to a `BitmapType`. For example:

```
MemHandle resH =
  DmGetResource (bitmapRsc, rscID);
BitmapType *bitmap = MemHandleLock (resH);
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpDelete`

BmpDelete

Purpose Delete a bitmap structure.

Prototype

```
Err BmpDelete (BitmapType *bitmapP)
```

Parameters

-> `bitmapP`	Pointer to the structure of the bitmap to be deleted. (See `BitmapType`.)

Result Returns `errNone` upon success, `sysErrParamErr` if the bitmap's `forScreen` flag is set or the bitmap resides in the storage heap. Returns one of the memory errors if the freeing the pointer fails.

Comments Only delete bitmaps that you've created using `BmpCreate`.

You cannot use this function on a bitmap located in a database. To delete a bitmap from a database, use the standard data manager calls.

Compatibility Implemented only if 3.5 New Feature Set is present.

BmpGetBits

Purpose Retrieve the bitmap's data.

Prototype

```
void *BmpGetBits (BitmapType *bitmapP)
```

Parameters

-> `bitmapP`	Pointer to the bitmap's structure. (See `BitmapType`.)

Result Returns a pointer to the bitmap's data.

Comments This function returns the bitmap's data even if the bitmap's `indirect` flag is set. (See `BitmapFlagsType`.)

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpBitsSize`

BmpGetColortable

Purpose Retrieve the bitmap's color table.

Prototype

```
ColorTableType *BmpGetColortable
(BitmapType *bitmapP)
```

Parameters -> `bitmapP` A pointer to the bitmap. See `BitmapType`.

Result Returns a pointer to the color table or `NULL` if the bitmap uses the system color table.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpColortableSize`

BmpSize

Purpose Return the size of the bitmap.

Prototype

```
UInt16 BmpSize (BitmapType *bitmapP)
```

Parameters -> `bitmapP` A pointer to the bitmap. See `BitmapType`.

Result Returns the size in bytes of the bitmap, including its header and color table (if any).

Comments If the bitmap has its `indirect` flag set (see `BitmapFlagsType`), the bitmap data is not included in the size returned by this function.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpBitsSize`, `BmpColortableSize`

ColorTableEntries

Purpose Macro that returns the color table.

Prototype `ColorTableEntries (ctP)`

Parameters -> `ctP` A pointer to a `ColorTableType` structure.

Result Returns an array of `RGBColorType` structures, one for each entry in the color table.

Comments You can use this macro to retrieve the RGB values in use by a bitmap. For example:

```
BitmapType *bmpP;
RGBColorType *tableP =
  ColorTableEntries(BmpGetColorTable(bmpP));
```

If you want to retrieve the RGB values in use by the system color table, you can simply use the `WinPalette` function instead of this macro:

```
RGBColorType table;
Err e;

/* allocate space for table */
e = WinPalette(winPaletteGet, 0, 256, &table);
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpGetColortable`

25

Character Attributes

This chapter provides reference material for character attributes functions defined in `CharAttr.h`.

Character Attribute Functions

ChrHorizEllipsis

Purpose Macro that returns the appropriate character code for the horizontal ellipsis.

Prototype

```
ChrHorizEllipsis (chP)
```

Parameters

<- `chP`	Pointer to a variable in which to return the horizontal ellipsis character code.

Result Returns nothing. Upon return, the variable pointed to by `chP` contains the horizontal ellipsis character.

Comments Version 3.1 of the Palm OS® uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use this macro to return the appropriate code for horizontal ellipsis regardless of which version of Palm OS your application is run on.

ChrIsHardKey

Purpose Macro that returns true if the character is one of the hard keys on the device.

Prototype
```
ChrIsHardKey (ch)
```

Parameters

-> `ch`	The character from the `keyDownEvent`.

Result `true` if the character is one of the four built-in hard keys on the device, `false` otherwise.

Compatibility This macro is obsolete and replaced by `TxtCharIsHardKey` if the International Feature Set is present.

ChrNumericSpace

Purpose Macro that returns the appropriate character code for the numeric space.

Prototype
```
ChrNumericSpace (chP)
```

Parameters

<- `chP`	Pointer to a variable in which to return the numeric space character code.

Result Returns nothing. Upon return, the variable pointed to by `chP` contains the numeric space character.

Comments Version 3.1 of the Palm OS uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use this macro to return the appropriate code for numeric space regardless of which version of Palm OS your application is run on.

GetCharAttr

Purpose Return a pointer to the character attribute array. This array is used by the character classification and character conversion macros (such as `isalpha`).

Prototype

```
UInt16* GetCharAttr (void)
```

Parameters None

Result A pointer to the attributes array. This is an array of 256 `UInt16` values, one for each possible character code. See `CharAttr.h` for an explanation of the attributes.

Compatibility This function is **not** implemented if International Feature Set is present.

NOTE: This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

See Also `TxtCharAttr`, `TxtCharXAttr`

GetCharCaselessValue

Purpose Return a pointer to an array that maps all characters to an assigned caseless and accentless value. Use this function for finding text.

Prototype

```
UInt8* GetCharCaselessValue (void)
```

Parameters None.

Result Returns a pointer to the sort array.

The compiler pads each byte out to a word so each index position contains two characters.

Note: array[x].high = sort value for character 2x+1.

Comment The `GetCharCaselessValue` conversion table converts each character into a numeric value that is caseless and sorted according to Microsoft Windows sorting rules:

- Punctuation characters have the lowest values,
- followed by numbers,
- followed by alpha characters.

  All forms of each alpha character have equivalent values, so that e = E = e-grave = e-circumflex, etc.

This conversion table is used by all the Palm OS sorting and comparison routines to yield caseless searches and caseless sorts in the almost same order as Windows-based programs, except that Palm OS routines produce the same sorting for all locales.

Compatibility This function is **not** implemented if International Feature Set is present.

NOTE: This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

GetCharSortValue

Purpose Return a pointer to an array that maps all characters to an assigned sorting value. Use this function for ordering (sorting) text.

Prototype

```
UInt8* GetCharSortValue (void)
```

Parameters None.

Result Returns a pointer to the attributes array. This is an array of 256 `UInt8` values, one for each possible character code.

The compiler pads each byte out to a word so each index position contains two characters.

NOTE: array[x].low = sort value for character 2x.

Compatibility This function is **not** implemented if International Feature Set is present.

NOTE: This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

26

Data and Resource Manager

This chapter describes the data manager and the resource manager API declared in the header file `DataMgr.h`. It discusses the following topics:

- Data Manager Data Structures
- Data Manager Constants
- Data Manager Functions
- Application-Defined Functions

For more information on the data and resource managers, see the chapter "Files and Databases" in the *Palm OS Programmer's Companion*.

Data Manager Data Structures

DmOpenRef

The `DmOpenRef` type defines a pointer to an open database. The database pointer is created and returned by `DmOpenDatabase`. It is used in any function that requires access to an open database.

```
typedef void* DmOpenRef
```

DmResID

The `DmResID` type defines a resource identifier. You assign each resource an ID at creation time. Note that resource IDs greater than 10000 are reserved for system use.

```
typedef UInt16 DmResID;
```

DmResType

The `DmResType` type defines the type of a resource. The resource type is a four-character code such as `'Tbmp'` for bitmap resources.

```
typedef UInt32 DmResType;
```

SortRecordInfoType

The `SortRecordInfoType` structure specifies information that might be used to sort a record. It is used in the database sorting functions. To create this structure, you can call `DmRecordInfo`, which returns these values for a given record.

```
typedef struct {
  UInt8 attributes;
  UInt8 uniqueID[3];
} SortRecordInfoType;

typedef SortRecordInfoType * SortRecordInfoPtr;
```

Field Descriptions

`attributes`	The record's attributes. See "Record Attribute Constants."
`uniqueID`	The unique identifier for the record.

Data Manager Constants

Category Constants

The following constants are used to specify information about categories:

Constant	Value	Description
dmAllCategories	0xFF	A mask used to represent all categories.
dmCategoryLength	16	The length of a category name. Currently, this is 16 bytes, which includes the null terminator.
dmRecAttrCategoryMask	0x0F	A mask used to retrieve the category information from the record's attributes field.
dmRecNumCategories	16	The number of categories allowed. Currently, this is 16, which includes the "Unfiled" category.
dmUnfiledCategory	0	A mask used to indicate the Unfiled category.

Record Attribute Constants

The following constants specify a database record's attributes.

Constant	Value	Description
dmMaxRecordIndex	0xFFFF	Indicates the highest record index allowed.
dmAllRecAttrs	0xF0	A mask used to specify all record attributes.
dmRecAttrBusy	0x20	Busy. (The application has locked access to this record. A call to DmGetRecord fails on a record that has this bit set.)
dmRecAttrDelete	0x80	Deleted
dmRecAttrDirty	0x40	Dirty (has been modified since last sync)
dmRecAttrSecret	0x10	Private
dmSysOnlyRecAttrs	0x20	A mask used to specify record attributes that only the system can change. (In other words, the busy attribute.)

Database Attribute Constants

The following constants define a database's attributes:

Constant	Description
`dmAllHdrAttrs`	A mask used to specify all header attributes.
`dmDBNameLength`	Maximum length of a database's name. Currently, this is 32 bytes, which include the null terminator. Note that database names must use only 7-bit ASCII characters (0x20 through 0x7E).
`dmHdrAttrAppInfoDirty`	The application info block is dirty (has been modified since the last sync).
`dmHdrAttrBackup`	The database should be backed up to the desktop computer if no application-specific conduit is available.
`dmHdrAttrCopyPrevention`	Prevents the database from being copied by methods such as IR beaming.
`dmHdrAttrHidden`	This database should be hidden from view. For example, this attribute is set to hide some applications in the launcher's main view. This attribute applies to Palm OS® version 3.2 and higher.
`dmHdrAttrLaunchableData`	This database is a data database but it can be "launched" from the launcher. For example, this attribute is set in Palm Query Applications (PQAs) launched by the Web Clipper application.
`dmHdrAttrOpen`	The database is open.
`dmHdrAttrOKToInstallNewer`	The backup conduit can install a newer version of this database with a different name if the current database is open. This mechanism is used to update the Graffiti® Shortcuts database, for example.
`dmHdrAttrReadOnly`	The database is a read-only database.
`dmHdrAttrResDB`	The database is a resource database.

Constant	Description
dmHdrAttrResetAfterInstall	The device must be reset after this database is installed. That is, the HotSync® application forces a reset after installing this database.
dmHdrAttrStream	The database is a file stream.
dmSysOnlyHdrAttrs	A mask specifying the attributes that only the system can change (open and resource database).

Error Codes

The following constants define error codes that are returned by the data manager and resource manager functions. Several functions return a failure value such as NULL or 0 instead of an error code. In many cases, you can call DmGetLastErr upon receiving this value and receive a more descriptive error code.

Also, note that on releases prior to Palm OS release 3.5, many data manager functions display a fatal error message using the ErrFatalDisplayIf macro if certain error conditions are true. Because the Palm OS ROMs are usually shipped with error checking set to partial, you receive the fatal error message. If a ROM is built with error checking set to none, the function returns one of the error codes listed here. (Note that Palm has never released a ROM with error checking set to none and has no plans to do so.)

Constant	Description
dmErrAlreadyExists	Another database with the same name already exists in RAM store.
dmErrAlreadyOpenForWrites	The database is already open with write access.
dmErrCantFind	The specified resource can't be found.
dmErrCantOpen	The database cannot be opened.

Constant	Description
dmErrCorruptDatabase	The database is corrupted.
dmErrDatabaseOpen	The function cannot be performed on an open database, and the database is open.
dmErrDatabaseNotProtected	DmDatabaseProtect failed to protect the specified database.
dmErrIndexOutOfRange	The specified index is out of range.
dmErrInvalidDatabaseName	The name you've specified for the database is invalid.
dmErrInvalidParam	The function received an invalid parameter.
dmErrMemError	A memory error occurred.
dmErrNoOpenDatabase	The function is to search all open databases, but there are none.
dmErrNotRecordDB	You've attempted to perform a record function on a resource database.
dmErrNotResourceDB	You've attempted to perform a resource manager function on a record database.
dmErrNotValidRecord	The record handle is invalid.
dmErrOpenedByAnotherTask	You've attempted to open a database that another task already has open.

Constant	Description
`dmErrReadOnly`	You've attempted to write to or modify a database that is in read-only mode.
`dmErrRecordArchived`	The function requires that the record not be archived, but it is.
`dmErrRecordBusy`	The function requires that the record not be busy, but it is.
`dmErrRecordDeleted`	The record has been deleted.
`dmErrRecordInWrongCard`	You've attempted to attach a record to a database when the record and database reside on different memory cards.
`dmErrResourceNotFound`	The resource can't be found.
`dmErrROMBased`	You've attempted to delete or modify a ROM-based database.
`dmErrSeekFailed`	The operation of seeking the next record in the category failed.
`dmErrUniqueIDNotFound`	A record with the specified unique ID can't be found.
`dmErrWriteOutOfBounds`	A write operation exceeded the bounds of the record.
`memErrCardNotPresent`	The specified card can't be found.

Constant	Description
memErrChunkLocked	The associated memory chunk is locked.
memErrInvalidParam memErrNotEnoughSpace	A memory error occurred.
memErrInvalidStoreHeader memErrRAMOnlyCard	The specified card has no storage RAM.
omErrBaseRequiresOverlay	An attempt was made to open a stripped resource database, but no associated overlay could be found.
omErrUnknownLocale	An attempt was made to open a resource database with overlays using an unknown locale.

Open Mode Constants

The following constants define the mode in which a database can be opened. You pass one or more of these as a parameter to DmOpenDatabase, DmOpenDatabaseByTypeCreator, or DmOpenDBNoOverlay:

Constant	Description
dmModeReadWrite	Read-write access.
dmModeReadOnly	Read-only access.
dmModeWrite	Write-only access.
dmModeLeaveOpen	Leave database open even after application quits.
dmModeExclusive	Don't let anyone else open this database.
dmModeShowSecret	Show records marked private.

Data Manager Functions

DmArchiveRecord

Purpose Mark a record as archived by leaving the record's chunk intact and setting the delete bit for the next sync.

Prototype

```
Err DmArchiveRecord (DmOpenRef dbP, UInt16 index)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Which record to archive.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrRecordArchived`
- `dmErrRecordDeleted`
- `memErrInvalidParam`

Some releases may display a fatal error message instead of returning the error code.

Comments When a record is archived, the deleted bit is set but the chunk is not freed and the local ID is preserved. This way, the next time the user synchronizes with the desktop system, the desktop can save the record data on the PC before it permanently removes the record entry and data from the Palm OS device.

See Also `DmRemoveRecord`, `DmDetachRecord`, `DmNewRecord`, `DmDeleteRecord`

DmAttachRecord

Purpose Attach an existing chunk ID handle to a database as a record.

Prototype

```
Err DmAttachRecord (DmOpenRef dbP, UInt16* atP,
MemHandle newH, MemHandle* oldHP)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
<-> `atP`	Pointer to the index where the new record should be placed. Specify the value `dmMaxRecordIndex` to add the record to the end of the database.
-> `newH`	Handle of the new record.
<-> `oldHP`	If non-`NULL` upon entry, indicates that the record at `*atP` should be replaced. Upon return, contains the handle to the replaced record.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrMemError`
- `memErrChunkLocked`
- `memErrInvalidParam`
- `memErrNotEnoughSpace`
- `dmErrReadOnly`
- `dmErrNotRecordDB`
- `dmErrRecordInWrongCard`
- `dmErrIndexOutOfRange`

Some releases may display a fatal error message instead of returning some of these error codes.

Comments Given the handle of an existing chunk, this routine makes that chunk a new record in a database and sets the dirty bit. The parameter `atP` points to an index variable. If `oldHP` is `NULL`, the new record is inserted at index `*atP` and all record indices that

follow are shifted down. If `*atP` is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in `*atP`. If `oldHP` is not `NULL`, the new record replaces an existing record at index `*atP` and the handle of the old record is returned in `*oldHP` so that the application can free it or attach it to another database.

This function is useful for cutting and pasting between databases.

See Also DmDetachRecord, DmNewRecord, DmNewHandle, DmFindSortPosition

DmAttachResource

Purpose Attach an existing chunk ID to a resource database as a new resource.

Prototype

```
Err DmAttachResource (DmOpenRef dbP,
MemHandle newH, DmResType resType, DmResID resID)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `newH`	Handle of new resource's data.
-> `resType`	Type of the new resource.
-> `resID`	ID of the new resource.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- dmErrMemError
- memErrChunkLocked
- memErrInvalidParam
- memErrNotEnoughSpace
- dmErrReadOnly
- dmErrRecordInWrongCard

Some releases may display a fatal error message instead of returning some of these error codes. All releases may display a fatal error message if the database is not a resource database.

Comments Given the handle of an existing chunk with resource data in it, this routine makes that chunk a new resource in a resource database. The new resource will have the given type and ID.

See Also DmDetachResource, DmRemoveResource, DmNewHandle, DmNewResource

DmCloseDatabase

Purpose Close a database.

Prototype

```
Err DmCloseDatabase (DmOpenRef dbP)
```

Parameters

-> dbP Database access pointer.

Result Returns errNone if no error, or dmErrInvalidParam if an error occurs. Some releases may display a fatal error message instead of returning the error code.

Comments This routine doesn't unlock any records that were left locked. Records and resources should not be left locked. If a record/resource is left locked, you should not use this reference because the record can disappear if the database is deleted by the user or during a HotSync®. To prevent the database from being deleted, you can use DmDatabaseProtect before closing.

If there is an overlay associated with the database passed in, DmCloseDatabase closes the overlay as well.

Compatibility Starting with Palm OS 2.0, DmCloseDatabase updates the database's modification date.

- On Palm OS 2.0, the modification date is updated if the database was opened with write access.
- On Palm OS 3.0 and higher, the modification date is updated only if a change has been made and the database was opened with write access. Changes that trigger an update include adding, deleting, archiving, rearranging, or resizing records, setting a record's dirty bit in DmReleaseRecord,

rearranging or deleting categories, or updating the database header fields using DmSetDatabaseInfo.

Under Palm OS 1.0, the modification date was never updated.

If you need to ensure that the modification date is updated the same way regardless of the operating system version, use DmSetDatabaseInfo to set the modification date explicitly.

See Also DmOpenDatabase, DmDeleteDatabase, DmOpenDatabaseByTypeCreator

DmCreateDatabase

Purpose Create a new database on the specified card with the given name, creator, and type.

Prototype

```
Err DmCreateDatabase (UInt16 cardNo,
const Char * nameP, UInt32 creator, UInt32 type,
Boolean resDB)
```

Parameters

-> cardNo	The card number to create the database on.
-> nameP	Name of new database, up to 32 ASCII bytes long, including the null terminator (as specified by dmDBNameLength). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).
-> creator	Creator of the database.
-> type	Type of the database.
-> resDB	If true, create a resource database.

Result Returns errNone if no error, or one of the following if an error occurs:

- dmErrInvalidDatabaseName
- dmErrAlreadyExists
- memErrCardNotPresent
- dmErrMemError

- memErrChunkLocked
- memErrInvalidParam
- memErrInvalidStoreHeader
- memErrNotEnoughSpace
- memErrRAMOnlyCard

May display a fatal error message if the master database list cannot be found.

Comments Call this routine to create a new database on a specific card. If another database with the same name already exists in RAM store, this routine returns a dmErrAlreadyExists error code. Once created, the database ID can be retrieved by calling DmFindDatabase. The database can be opened using the database ID. To create a resource database instead of a record-based database, set the `resDB` Boolean to `true`.

After you create a database, it's recommended that you call DmSetDatabaseInfo to set the version number. Databases default to version 0 if the version isn't explicitly set.

See Also DmCreateDatabaseFromImage, DmOpenDatabase, DmDeleteDatabase

DmCreateDatabaseFromImage

Purpose Create an entire database from a single resource that contains an image of the database.

Prototype

```
Err DmCreateDatabaseFromImage (MemPtr bufferP)
```

Parameters

-> `bufferP`	Pointer to locked resource containing database image.

Result Returns `errNone` if no error.

Comments An image is the same as a desktop file representation of a prc or pdb file.

This function is intended for applications in the ROM to install default databases after a hard reset. RAM-based applications that want to install a default database should install a pdb file separately to save storage heap space.

See Also DmCreateDatabase, DmOpenDatabase

DmDatabaseInfo

Purpose Retrieve information about a database.

Prototype

```
Err DmDatabaseInfo (UInt16 cardNo, LocalID dbID,
Char* nameP, UInt16* attributesP,
UInt16* versionP, UInt32* crDateP,
UInt32* modDateP, UInt32* bckUpDateP,
UInt32* modNumP, LocalID* appInfoIDP,
LocalID* sortInfoIDP, UInt32* typeP,
UInt32* creatorP)
```

Parameters

-> cardNo	Number of the card the database resides on.
-> dbID	Database ID of the database.
<- nameP	The database's name. Pass a pointer to 32-byte character array for this parameter, or NULL if you don't care about the name.
<- attributesP	The database's attribute flags. The section "Database Attribute Constants" lists constants you can use to query the values returned in this parameter. Pass NULL for this parameter if you don't want to retrieve it.
<- versionP	The application-specific version number. The default version number is 0. Pass NULL for this parameter if you don't want to retrieve it.
<- crDateP	The date the database was created, expressed as the number of seconds since the first instant of Jan 1, 1904. Pass NULL for this parameter if you don't want to retrieve it.

<- `modDateP`	The date the database was last modified, expressed as the number of seconds since the first instant of Jan 1, 1904. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `bckUpDateP`	The date the database was backed up, expressed as the number of seconds since the first instant of Jan 1, 1904. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `modNumP`	The modification number, which is incremented every time a record in the database is added, modified, or deleted. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `appInfoIDP`	The local ID of the application info block, or `NULL`. The application info block is an optional field that the database may use to store application-specific information about the database. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `sortInfoIDP`	The local ID of the database's sort table. This is an optional field in the database header. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `typeP`	The database's type, specified when it is created. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `creatorP`	The database's creator, specified when it is created. Pass `NULL` for this parameter if you don't want to retrieve it.

Result Returns `errNone` if no error, or `dmErrInvalidParam` if an error occurs.

Compatibility Updating of the modification date differs based on which version of the OS your application is running on.

- Under Palm OS 1.0, the modification date is never updated.

- Under Palm OS 2.0, the modification date is updated every time a database opened with write access is closed.
- Beginning with Palm OS 3.0, the modification date is updated only if a change has been made to the database opened with write access. (The update still occurs upon closing the database.) Changes that trigger an update include adding, deleting, archiving, rearranging, or resizing records, setting a record's dirty bit in `DmReleaseRecord`, rearranging or deleting categories, or updating the database header fields using `DmSetDatabaseInfo`.

If you need to ensure that the modification date is updated the same way regardless of the operating system version, use `DmSetDatabaseInfo` to set the modification date explicitly.

See Also `DmSetDatabaseInfo`, `DmDatabaseSize`, `DmOpenDatabaseInfo`, `DmFindDatabase`, `DmGetNextDatabaseByTypeCreator`, `TimSecondsToDateTime`

DmDatabaseProtect

Purpose Increment or decrement the database's protection count.

Prototype

```
Err DmDatabaseProtect (UInt16 cardNo,
LocalID dbID, Boolean protect)
```

Parameters

-> `cardNo`	Card number of database to protect/unprotect.
-> `dbID`	Local ID of database to protect/unprotect.
-> `protect`	If `true`, `protect` count will be incremented. If `false`, `protect` count will be decremented.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `memErrCardNotPresent`
- `dmErrROMBased`
- `dmErrCantFind`
- `memErrNotEnoughSpace`

- `dmErrDatabaseNotProtected`

Comments This routine can be used to prevent a database from being deleted (by passing `true` for the `protect` parameter). It increments the protect count if `protect` is `true` and decrements it if `protect` is `false`. All `true` calls should be balanced by `false` calls before the application terminates.

Use this function if you want to keep a particular record or resource in a database locked down but don't want to keep the database open. This information is kept in the dynamic heap, so all databases are "unprotected" at system reset.

If the database is a resource database that has an overlay associated with it for the current locale, the overlay is also protected or unprotected by this call.

Compatibility Implemented only if 2.0 New Feature Set is present. Overlay support is only available if 3.5 New Feature Set is present.

DmDatabaseSize

Purpose Retrieve size information on a database.

Prototype

```
Err DmDatabaseSize (UInt16 cardNo, LocalID dbID,
UInt32* numRecordsP, UInt32* totalBytesP,
UInt32* dataBytesP)
```

Parameters

-> `cardNo`	Card number the database resides on.
-> `dbID`	Database ID of the database.
<- `numRecordsP`	The total number of records in the database. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `totalBytesP`	The total number of bytes used by the database including the overhead. Pass `NULL` for this parameter if you don't want to retrieve it.

<- `dataBytesP`	The total number of bytes used to store just each record's data, not including overhead. Pass `NULL` for this parameter if you don't want to retrieve it.

Result Returns `errNone` if no error, or `dmErrMemError` if an error occurs.

See Also `DmDatabaseInfo`, `DmOpenDatabaseInfo`, `DmFindDatabase`, `DmGetNextDatabaseByTypeCreator`

DmDeleteCategory

Purpose Delete all records in a category. The category name is not changed.

Prototype

```
Err DmDeleteCategory (DmOpenRef dbR,
UInt16 categoryNum)
```

Parameters

-> `dbR`	Database access pointer.
-> `categoryNum`	Category of records to delete. Category masks such as `dmAllCategories` are invalid.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `memErrInvalidParam`

Some releases may display a fatal error message instead of returning the error code.

Comments This function deletes all records in a category, but does not delete the category itself. For each record in the category, `DmDeleteCategory` marks the `delete` bit in the database header for the record and disposes of the record's data chunk. The record entry in the database header remains, but its `localChunkID` is set to `NULL`.

If the category contains no records, this function does nothing and returns `errNone` to indicate success. The `categoryNum` parameter is assumed to represent a single category. If you pass a category

mask to specify more than one category, this function interprets that value as a single category, finds no records to delete in that category, and returns `errNone`.

You can use the `DmRecordInfo` call to obtain a category index from a given record. For example:

```
DmOpenRef myDB;
UInt16 record, attr, category, total;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
err = DmDeleteCategory(myDB, category);
```

Compatibility Implemented only if 2.0 New Feature Set is present.

DmDeleteDatabase

Purpose Delete a database and all its records.

Prototype

```
Err DmDeleteDatabase (UInt16 cardNo, LocalID dbID)
```

Parameters

-> `cardNo`	Card number the database resides on.
-> `dbID`	Database ID.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrCantFind`
- `dmErrCantOpen`
- `memErrChunkLocked`
- `dmErrDatabaseOpen`
- `dmErrROMBased`
- `memErrInvalidParam`
- `memErrNotEnoughSpace`

Comments Call this routine to delete a database. This routine deletes the database, the application info block, the sort info block, and any other overhead information that is associated with this database.

If the database has an overlay associated with it, this function does **not** delete the overlay. You can delete the overlay with a separate call to `DmDeleteDatabase`.

This routine accepts a database ID as a parameter. To determine the database ID, call either `DmFindDatabase` or `DmGetDatabase` with a database index.

See Also `DmDeleteRecord`, `DmRemoveRecord`, `DmRemoveResource`, `DmCreateDatabase`, `DmGetNextDatabaseByTypeCreator`, `DmFindDatabase`

DmDeleteRecord

Purpose Delete a record's chunk from a database but leave the record entry in the header and set the `delete` bit for the next sync.

Prototype

```
Err DmDeleteRecord (DmOpenRef dbP, UInt16 index)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Which record to delete.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrRecordArchived`
- `dmErrRecordDeleted`
- `memErrInvalidParam`

Some releases may display a fatal error message instead of returning the error code.

Comments Marks the `delete` bit in the database header for the record and disposes of the record's data chunk. Does not remove the record entry from the database header, but simply sets the `localChunkID` of the record entry to `NULL`.

See Also DmDetachRecord, DmRemoveRecord, DmArchiveRecord, DmNewRecord

DmDetachRecord

Purpose Detach and orphan a record from a database but don't delete the record's chunk.

Prototype

```
Err DmDetachRecord (DmOpenRef dbP, UInt16 index,
MemHandle* oldHP)
```

Parameters

-> `dbP`	`DmOpenRef` to open.
-> `index`	Index of the record to detach.
<-> `oldHP`	Pointer to return handle of the detached record.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrIndexOutOfRange
- dmErrNotRecordDB
- memErrChunkLocked
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code.

Comments This routine detaches a record from a database by removing its entry from the database header and returns the handle of the record's data chunk in `*oldHP`. Unlike DmDeleteRecord, this

routine removes its entry in the database header but it does not delete the actual record.

See Also DmAttachRecord, DmRemoveRecord, DmArchiveRecord, DmDeleteRecord

DmDetachResource

Purpose Detach a resource from a database and return the handle of the resource's data.

Prototype

```
Err DmDetachResource (DmOpenRef dbP, UInt16 index,
MemHandle* oldHP)
```

Parameters

-> dbP	DmOpenRef to open database.
-> index	Index of resource to detach.
<-> oldHP	Pointer to return handle of the detached record.

Result Returns errNone if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrIndexOutOfRange
- dmErrCorruptDatabase
- memErrChunkLocked
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code. All releases may display a fatal error message if the database is not a resource database.

Comments This routine detaches a resource from a database by removing its entry from the database header and returns the handle of the resource's data chunk in *oldHP.

See Also DmAttachResource, DmRemoveResource

DmFindDatabase

Purpose Return the database ID of a database by card number and name.

Prototype

```
LocalID DmFindDatabase (UInt16 cardNo,
const Char* nameP)
```

Parameters

-> `cardNo`	Number of card to search.
-> `nameP`	Name of the database to look for.

Result Returns the database ID. If the database can't be found, this function returns 0, and `DmGetLastErr` returns an error code indicating the reason for failure.

See Also `DmGetNextDatabaseByTypeCreator`, `DmDatabaseInfo`, `DmOpenDatabase`

DmFindRecordByID

Purpose Return the index of the record with the given unique ID.

Prototype

```
Err DmFindRecordByID (DmOpenRef dbP,
UInt32 uniqueID, UInt16* indexP)
```

Parameters

-> `dbP`	Database access pointer.
-> `uniqueID`	Unique ID to search for.
<- `indexP`	Return index.

Result Returns 0 if found, otherwise `dmErrUniqueIDNotFound`. May display a fatal error message if the unique ID is invalid.

See Also `DmQueryRecord`, `DmGetRecord`, `DmRecordInfo`

DmFindResource

Purpose Search the given database for a resource by type and ID, or by pointer if it is non-NULL.

Prototype

```
UInt16 DmFindResource (DmOpenRef dbP,
DmResType resType, DmResID resID, MemHandle resH)
```

Parameters

-> dbP	Open resource database access pointer.
-> resType	Type of resource to search for.
-> resID	ID of resource to search for.
->resH	Pointer to locked resource, or NULL.

Result Returns index of resource in resource database, or -1 if not found.

May display a fatal error message if the database is not a resource database.

Comments Use this routine to find a resource in a particular resource database by type and ID or by pointer. It is particularly useful when you want to search only one database for a resource and that database is not the topmost one.

IMPORTANT: This function searches for the resource only in the database you specify. If you pass a pointer to a base resource database, its overlay is **not** searched. To search both a base database and its overlay for a localized resource, use DmGet1Resource instead of this function.

If resH is NULL, the resource is searched for by type and ID.

If resH is not NULL, resType and resID are ignored and the index of the given locked resource is returned.

Once the index of a resource is determined, it can be locked down and accessed by calling DmGetResourceIndex.

See Also DmGetResource, DmSearchResource, DmResourceInfo, DmGetResourceIndex, DmFindResourceType

DmFindResourceType

Purpose Search the given database for a resource by type and type index.

Prototype

```
UInt16 DmFindResourceType (DmOpenRef dbP,
DmResType resType, UInt16 typeIndex)
```

Parameters

-> `dbP`	Open resource database access pointer.
-> `resType`	Type of resource to search for.
-> `typeIndex`	Index of given resource type.

Result Index of resource in resource database, or -1 if not found.

May display a fatal error message if the database is not a resource database.

Comments Use this routine to retrieve all the resources of a given type in a resource database. By starting at `typeIndex` 0 and incrementing until an error is returned, the total number of resources of a given type and the index of each of these resources can be determined. Once the index of a resource is determined, it can be locked down and accessed by calling `DmGetResourceIndex`.

IMPORTANT: This function searches for resources only in the database you specify. If you pass a pointer to a base resource database, its overlay is **not** searched. To search both a base database and its overlay for a localized resource, use `DmGet1Resource` instead of this function.

See Also `DmGetResource`, `DmSearchResource`, `DmResourceInfo`, `DmGetResourceIndex`, `DmFindResource`

DmFindSortPosition

Purpose

Return where a record should be. Useful to find where to insert a record with DmAttachRecord. Uses a binary search.

Prototype

```
UInt16 DmFindSortPosition (DmOpenRef dbP,
void* newRecord, SortRecordInfoPtr newRecordInfo,
DmComparF *compar, Int16 other)
```

Parameters

-> dbP	Database access pointer.
-> newRecord	Pointer to the new record.
-> newRecordInfo	Sort information about the new record. See SortRecordInfoType.
-> compar	Pointer to comparison function. See DmComparF.
-> other	Any value the application wants to pass to the comparison function.

Result

The position where the record should be inserted.

The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

Comments

If newRecord has the same key as another record in the database, DmFindSortPosition assumes that newRecord should be inserted after that record. If there are several records with the same key, newRecord is inserted after all of them. For this reason, if you use DmFindSortPosition to search for the location of a record that you know is already in the database, you must subtract 1 from the result. (Be sure to check that the value is not 0.)

If there are deleted records in the database, DmFindSortPosition only works if those records are at the end of the database. DmFindSortPosition always assumes that a deleted record is greater than or equal to any other record.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `DmFindSortPositionV10`

DmFindSortPositionV10

Purpose Return where a record should be. Useful to find where to insert a record with `DmAttachRecord`. Uses a binary search.

Prototype

```
UInt16 DmFindSortPositionV10 (DmOpenRef dbP,
void* newRecord, DmComparF *compar, Int16 other)
```

Parameters

-> `dbP`	Database access pointer.
-> `newRecord`	Pointer to the new record.
-> `compar`	Pointer to comparison function. See `DmComparF`.
-> `other`	Any value the application wants to pass to the comparison function.

Result Returns the position where the record should be inserted. The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

Comments If there are deleted records in the database, `DmFindSortPositionV10` only works if those records are at the end of the database. `DmFindSortPositionV10` always assumes that a deleted record is greater than or equal to any other record.

Compatibility This function corresponds to the 1.0 version of `DmFindSortPosition`.

See Also `DmFindSortPosition`, `DmQuickSort`, `DmInsertionSort`

DmGetAppInfoID

Purpose Return the local ID of the application info block.

Prototype
```
LocalID DmGetAppInfoID (DmOpenRef dbP).
```

Parameters

-> `dbP`	Database access pointer.

Result Returns local ID of the application info block.

See Also `DmDatabaseInfo`, `DmOpenDatabase`

DmGetDatabase

Purpose Return the database header ID of a database by index and card number.

Prototype
```
LocalID DmGetDatabase (UInt16 cardNo,
UInt16 index)
```

Parameters

-> `cardNo`	Card number of database.
-> `index`	Index of database.

Result Returns the database ID, or 0 if an invalid parameter is passed.

Comments Call this routine to retrieve the database ID of a database by index. The index should range from 0 to `DmNumDatabases`-1.

This routine is useful for getting a directory of all databases on a card. The databases returned may reside in either the ROM or the RAM. The order in which databases are returned is not fixed; therefore, you should not rely on receiving a list of databases in a particular order.

See Also `DmOpenDatabase`, `DmNumDatabases`, `DmDatabaseInfo`, `DmDatabaseSize`

DmGetDatabaseLockState

Purpose Return information about the number of locked and busy records in a database.

Prototype

```
void DmGetDatabaseLockState (DmOpenRef dbR,
UInt8* highest, UInt32* count, UInt32* busy)
```

Parameters

-> `dbR`	Database access pointer.
<- `highest`	The highest lock count found for all of the records in the database. If a database has two records, one has a lock count of 2 and one has a lock count of 1, the highest lock count is 2. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `count`	The number of records that have the lock count that is returned in the `highest` parameter. Pass `NULL` for this parameter if you don't want to retrieve it.
<- `busy`	The number of records that have the busy bit set. Pass `NULL` for this parameter if you don't want to retrieve it.

Result No return value. Returns all information in the parameters you pass.

Comments This function is intended to be used for debugging purposes. You can use it to obtain information about how many records are busy and how much locking occurs.

Compatibility Implemented only if 3.2 New Feature Set is present.

DmGetLastErr

Purpose Return error code from last data manager call.

Prototype

```
Err DmGetLastErr (void)
```

Parameters None.

Result Error code from last unsuccessful data manager call.

Comments Use this routine to determine why a data manager call failed. In particular, calls like DmGetRecord return 0 if unsuccessful, so calling DmGetLastErr is the only way to determine why they failed.

Note that DmGetLastErr does not always reflect the error status of the last data manager call. Rather, it reflects the error status of data manager calls that don't return an error code. For some of those calls, the saved error code value is not set to 0 when the call is successful.

For example, if a call to DmOpenDatabaseByTypeCreator returns NULL for database reference (that is, it fails), DmGetLastErr returns something meaningful; otherwise, it returns the error value of some previous data manager call.

Only the following data manager functions currently affect the value returned by DmGetLastErr:

DmFindDatabase	DmOpenDatabaseByTypeCreator
DmOpenDatabase	DmNewRecord
DmQueryRecord	DmGetRecord
DmQueryNextInCategory	DmPositionInCategory
DmSeekRecordInCategory	DmResizeRecord
DmGetResource	DmGet1Resource
DmNewResource	DmGetResourceIndex

DmNewHandle | DmOpenDBNoOverlay
DmResizeResource

DmGetNextDatabaseByTypeCreator

Purpose

Return a database header ID and card number given the type and/or creator. This routine searches all memory cards for a match.

Prototype

```
Err DmGetNextDatabaseByTypeCreator
(Boolean newSearch, DmSearchStatePtr stateInfoP,
UInt32 type, UInt32 creator,
Boolean onlyLatestVers, UInt16* cardNoP,
LocalID* dbIDP)
```

Parameters

-> `newSearch`	`true` if starting a new search.
<-> `stateInfoP`	If `newSearch` is `false`, this must point to the same data used for the previous invocation.
-> `type`	Type of database to search for, pass 0 as a wildcard.
-> `creator`	Creator of database to search for, pass 0 as a wildcard.
-> `onlyLatestVers`	If `true`, only the latest version of a database with a given type and creator is returned.
<- `cardNoP`	On exit, the card number of the found database.
<- `dbIDP`	Database local ID of the found database.

Result

Returns `errNone` if no error, or `dmErrCantFind` if no matches were found.

Comments

You may need to call this function successively to discover all databases having a specified type/creator pair.

To start the search, pass `true` for `newSearch`. Allocate a `DmSearchStateType` structure and pass it as the `stateInfoP`

parameter. `DmGetNextDatabaseByTypeCreator` stores private information in `stateInfoP` and uses it if the search is continued.

To continue a search where the previous one left off, pass `false` for `newSearch` and pass the same `stateInfoP` that you used during the previous call to this function.

You can pass `NULL` as a wildcard operator for the `type` or `creator` parameter to conduct searches of wider scope. If the `type` parameter is `NULL`, this routine can be called successively to return all databases of the given creator. If the `creator` parameter is `NULL`, this routine can be called successively to return all databases of the given type. You can also pass `NULL` as the value for both of these parameters to return all available databases without regard to type or creator.

Because databases are scattered freely throughout memory space, they are not returned in any particular order—any database matching the specified type/creator criteria can be returned.Thus, if the value of the `onlyLatestVers` parameter is `false`, this function may return a database which is not the most recent version matching the specified type/creator pair. To obtain only the latest version of a database matching the search criteria, set the value of the `onlyLatestVers` parameter to `true`.

When determining which is the latest version of the database, RAM databases are considered newer than ROM databases that have the same version number. Because of this, you can replace any ROM-based application with your own version of it. Also, a RAM database on card 1 is considered newer than a RAM database on card 0 if the version numbers are identical.

Compatibility

In Palm OS version 3.1 and higher, if `onlyLatestVers` is `true`, you only receive one matching database for each type/creator pair. In version 3.0 and earlier, you could receive multiple matching databases if `onlyLatestVers` was `true`.

Note that the behavior is different only when you have specified a value for both `type` and `creator` and `onlyLatestVers` is `true`.

For example, suppose your application maintains three databases that all have the same type, creator, and version number and you write this code to process them in some way:

```
DmSearchStateType state;
Boolean latestVer;
UInt16 card;
LocalID currentDB = 0;

theErr = DmGetNextDatabaseByTypeCreator(true,
  &state, myType, myCreator, latestVer, &card,
  &currentDB);
while (!theErr && currentDB) {
  /* do something with currentDB */
  /* now get the next DB */
  theErr = DmGetNextDatabaseByTypeCreator(
    false, &state, myType, myCreator,
    latestVer, &card, &currentDB);
}
```

If `latestVer` is `false`, then your code will work the same on all versions of Palm OS and will return all three databases whose type and creator match those specified. If `latestVer` is `true`, this code returns all three databases on Palm OS version 3.0 and earlier, but only returns one database on version 3.1 and higher. (Exactly which database it returns is unspecified.)

If you expect multiple databases to match your search criteria, make sure you call `DmGetNextDatabaseByTypeCreator` in one of the following ways to ensure that your code operates the same on all Palm OS versions:

- Set `onlyLatestVers` to `false` if you specify both a `type` and `creator`.
- Specify `NULL` for either the `type` or `creator` parameter (or both).

See Also `DmGetDatabase`, `DmFindDatabase`, `DmDatabaseInfo`, `DmOpenDatabaseByTypeCreator`, `DmDatabaseSize`

DmGetRecord

Purpose Return a handle to a record by index and mark the record busy.

Prototype

```
MemHandle DmGetRecord (DmOpenRef dbP,
UInt16 index)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Which record to retrieve.

Result Returns a handle to record data. If another call to `DmGetRecord` for the same record is attempted before the record is released, `NULL` is returned and `DmGetLastErr` returns an error code indicating the reason for failure.

Comments Returns a handle to given record and sets the `busy` bit for the record.

If the record is ROM-based (pointer accessed), this routine makes a fake handle to it and stores this handle in the `DmAccessType` structure.

`DmReleaseRecord` should be called as soon as the caller finishes viewing or editing the record.

See Also `DmSearchRecord`, `DmFindRecordByID`, `DmRecordInfo`, `DmReleaseRecord`, `DmQueryRecord`

DmGetResource

Purpose Search all open resource databases and return a handle to a resource, given the resource type and ID.

Prototype

```
MemHandle DmGetResource (DmResType type,
DmResID resID)
```

Parameters

-> `type`	The resource type.

`->resID`	The resource ID.

Result Handle to resource data. If the specified resource cannot be found, this function returns `NULL` and `DmGetLastErr` returns an error code indicating the reason for failure.

Comments Searches all open resource databases starting with the most recently opened one for a resource of the given type and ID. If found, the resource handle is returned. The application should call `DmReleaseResource` as soon as it finishes accessing the resource data. The resource handle is not locked by this function.

This function always returns the resource located in the overlay if any open overlay has a resource matching that type and ID. If there is no overlay version of the resource, this function returns the resource from the base database.

See Also `DmGet1Resource`, `DmReleaseResource`, `ResLoadConstant`

DmGetResourceIndex

Purpose Return a handle to a resource by index.

Prototype

```
MemHandle DmGetResourceIndex (DmOpenRef dbP,
UInt16 index)
```

Parameters

-> `dbP`	Access pointer to open database.
-> `index`	Index of resource to lock down.

Result Handle to resource data. If the specified index is out of range, this function returns `NULL` and `DmGetLastErr` returns an error code indicating the reason for failure.

May display a fatal error message if the database is not a resource database.

IMPORTANT: This function accesses the resource only in the database you specify. If you pass a pointer to a base resource database, its overlay is **not** accessed. Therefore, you should use care when using this function to access a potentially localized resource. You can use `DmSearchResource` to obtain a pointer to the overlay database if the resource is localized; however, it's more convenient to use `DmGetResource` or `DmGet1Resource`.

See Also `DmFindResource`, `DmFindResourceType`, `DmSearchResource`

DmGet1Resource

Purpose Search the most recently opened resource database and return a handle to a resource given the resource type and ID.

Prototype

```
MemHandle DmGet1Resource (DmResType type,
DmResID resID)
```

Parameters

-> `type`	The resource type.
-> `resID`	The resource ID.

Result Handle to resource data. If unsuccessful, this function returns `NULL` and `DmGetLastErr` returns an error code indicating the reason for failure.

Comments Searches the most recently opened resource database for a resource of the given type and ID. If the database has an overlay associated with it, the overlay is searched first, and then the base database is searched if the overlay does not contain the resource. If found, the resource handle is returned. The application should call `DmReleaseResource` as soon as it finishes accessing the resource data. The resource handle is not locked by this function.

See Also `DmGetResource`, `DmReleaseResource`, `ResLoadConstant`

DmInsertionSort

Purpose Sort records in a database.

Prototype

```
Err DmInsertionSort (DmOpenRef dbR,
DmComparF *compar, Int16 other)
```

Parameters

-> `dbR`	Database access pointer.
-> `compar`	Comparison function. See `DmComparF`.
-> `other`	Any value the application wants to pass to the comparison function.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrNotRecordDB`

Some releases may display a fatal error message instead of returning the error code.

Comments Deleted records are placed last in any order. All others are sorted according to the passed comparison function. Only records which are out of order move. Moved records are moved to the end of the range of equal records. If a large number of records are being sorted, try to use the quick sort.

The following insertion-sort algorithm is used: Starting with the second record, each record is compared to the preceding record. Each record not greater than the last is inserted into sorted position within those already sorted. A binary insertion is performed. A moved record is inserted after any other equal records.

See Also `DmQuickSort`

DmMoveCategory

Purpose Move all records in a category to another category.

Prototype

```
Err DmMoveCategory (DmOpenRef dbP,
UInt16 toCategory, UInt16 fromCategory,
Boolean dirty)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
->`toCategory`	Category to which the records should be added.
-> `fromCategory`	Category from which to remove records.
-> `dirty`	If `true`, set the dirty bit.

Result Returns 0 if successful, or `dmErrReadOnly` if the database is in read-only mode. Some releases may display a fatal error message instead of returning the error code.

Comments If `dirty` is `true`, the moved records are marked as dirty.

The `toCategory` and `fromCategory` parameters hold category index values. You can learn which category a record is in with the `DmRecordInfo` call and use that value in this function. For example, the following code, ensures that the records `rec1` and `rec2` are in the same category:

```
DmOpenRef myDB;
UInt16 rec1, rec2;
UInt16 rec1Attr, rec2Attr;
UInt16 category1, category2;

DmRecordInfo (myDB, rec1, &rec1Attr, NULL,
  NULL);
category1 = rec1Attr & dmRecAttrCategoryMask;
DmRecordInfo(myDB, rec2, &rec2Attr, NULL,
  NULL);
category2 = rec2Attr & dmRecAttrCategoryMask;
if (category1 != category2)
  DmMoveCategory(myDB, category1, category2,
```

```
        true);
```

DmMoveRecord

Purpose Move a record from one index to another.

Prototype

```
Err DmMoveRecord (DmOpenRef dbP, UInt16 from,
UInt16 to)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `from`	Index of record to move.
-> `to`	Where to move the record.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrNotRecordDB`
- `dmErrMemError`
- `memErrInvalidParam`
- `memErrChunkLocked`

Some releases may display a fatal error message instead of returning the error code.

Comments Insert the record at the `to` index and move other records down. The `to` position should be viewed as an insertion position. This value may be one greater than the index of the last record in the database. In cases where `to` is greater than `from`, the new index of the record becomes `to`–1 after the move is complete.

DmNewHandle

Purpose Attempt to allocate a new chunk in the same data heap or card as the database header of the passed database access pointer. If there is not enough space in that data heap, try other heaps.

Prototype

```
MemHandle DmNewHandle (DmOpenRef dbP, UInt32 size)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `size`	Size of new handle.

Result Returns the `chunkID` of new chunk. If an error occurs, returns 0, and `DmGetLastErr` returns an error code indicating the reason for failure.

Comments Allocates a new handle of the given size. Ensures that the new handle is in the same memory card as the given database. This guarantees that you can attach the handle to the database as a record to obtain and save its `LocalID` in the `appInfoID` or `sortInfoID` fields of the header.

The handle should be attached to a database as soon as possible. If it is not attached to a database and the application crashes, the memory used by the new handle is unavailable until the next soft reset.

DmNewRecord

Purpose Return a handle to a new record in the database and mark the record busy.

Prototype

```
MemHandle DmNewRecord (DmOpenRef dbP, UInt16* atP,
UInt32 size)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
<-> `atP`	Pointer to index where new record should be placed. Specify the value `dmMaxRecordIndex` to add the record to the end of the database.

-> `size`	Size of new record.

Result Handle to record data. If an error occurs, this function returns 0 and `DmGetLastErr` returns an error code indicating the reason for failure.

Some releases may display a fatal error message if the database is opened in read-only mode or it is a resource database.

Comments Allocates a new record of the given size, and returns a handle to the record data. The parameter `atP` points to an index variable. The new record is inserted at index `*atP` and all record indices that follow are shifted down. If `*atP` is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in `*atP`.

Both the `busy` and `dirty` bits are set for the new record and a unique ID is automatically created.

`DmReleaseRecord` should be called as soon as the caller finishes viewing or editing the record.

See Also `DmAttachRecord`, `DmRemoveRecord`, `DmDeleteRecord`

DmNewResource

Purpose Allocate and add a new resource to a resource database.

Prototype

```
MemHandle DmNewResource (DmOpenRef dbP,
DmResType resType, DmResID resID, UInt32 size)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `resType`	Type of the new resource.
-> `resID`	ID of the new resource.
-> `size`	Desired size of the new resource.

Result Returns a handle to the new resource. If an error occurs, this function returns `NULL` and `DmGetLastErr` returns an error code indicating the reason for failure.

May display a fatal error message if the database is not a resource database.

Comments Allocates a memory chunk for a new resource and adds it to the given resource database. The new resource has the given type and ID. If successful, the application should call DmReleaseResource as soon as it finishes initializing the resource.

See Also DmAttachResource, DmRemoveResource

DmNextOpenDatabase

Purpose Return DmOpenRef to next open database for the current task.

Prototype

```
DmOpenRef DmNextOpenDatabase (DmOpenRef currentP)
```

Parameters

-> currentP	Current database access pointer or NULL.

Result DmOpenRef to next open database, or NULL if there are no more.

Comments Call this routine successively to get the DmOpenRefs of all open databases. Pass NULL for currentP to get the first one. Applications don't usually call this function, but is useful for system information.

See Also DmOpenDatabaseInfo, DmDatabaseInfo

DmNextOpenResDatabase

Purpose Return access pointer to next open resource database in the search chain.

Prototype

```
DmOpenRef DmNextOpenResDatabase (DmOpenRef dbP)
```

Parameters

-> dbP	Database reference, or 0 to start search from the top.

Result Pointer to next open resource database.

Comments Returns pointer to next open resource database. To get a pointer to the first one in the search chain, pass `NULL` for `dbP`. This is the database that is searched when `DmGet1Resource` is called.

If you use this function to access a resource database that might have an overlay associated with it, be careful how you use the result. The `DmOpenRef` returned by this function is a pointer to the overlay database, not the base database. If you subsequently pass this pointer to `DmFindResource`, you'll receive a handle to the overlaid resource. If you're searching for a resource that is found only in the base, you won't find it. Instead, always use `DmGetResource` or `DmGet1Resource` to obtain a resource. Both of those functions search both the overlay databases and their associated base databases.

DmNumDatabases

Purpose Determine how many databases reside on a memory card.

Prototype

```
UInt16 DmNumDatabases (UInt16 cardNo)
```

Parameters -> `cardNo` Number of the card to check.

Result The number of databases found.

Comments This routine is helpful for getting a directory of all databases on a card. The routine `DmGetDatabase` accepts an index from 0 to `DmNumDatabases` -1 and returns a database ID by index.

See Also `DmGetDatabase`

DmNumRecords

Purpose Return the number of records in a database.

Prototype
```
UInt16 DmNumRecords (DmOpenRef dbP)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.

Result The number of records in a database.

Comments Records that have that have the `deleted` bit set (that is, records that will be deleted during the next synchronization because the user has marked them deleted) are included in the count. If you want to exclude these records from your count, use `DmNumRecordsInCategory` and pass `dmAllCategories` as the category.

See Also `DmNumRecordsInCategory`, `DmRecordInfo`, `DmSetRecordInfo`

DmNumRecordsInCategory

Purpose Return the number of records of a specified category in a database.

Prototype
```
UInt16 DmNumRecordsInCategory (DmOpenRef dbP,
UInt16 category)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `category`	Category index.

Result The number of records in the category.

Comments Because this function must examine all records in the database, it can be slow to return, especially when called on a large database.

Records that have the `deleted` bit set are not counted, and if the user has specified to hide or mask private records, private records are not counted either.

You can use the DmRecordInfo call to obtain a category index from a given record. For example:

```
DmOpenRef myDB;
UInt16 record, attr, category, total;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
total = DmNumRecordsInCategory(myDB, category);
```

See Also DmNumRecords, DmQueryNextInCategory, DmPositionInCategory, DmSeekRecordInCategory, DmMoveCategory

DmNumResources

Purpose Return the total number of resources in a given resource database.

Prototype

```
UInt16 DmNumResources (DmOpenRef dbP)
```

Parameters

-> `dbP` `DmOpenRef` to open database.

Result The total number of resources in the given database.

May display a fatal error message if the database is not a resource database.

Comments `DmNumResources` only counts the resources in the database indicated by the `DmOpenRef` parameter. If the database is a resource database that has an overlay associated with it, this function only returns the number of resources in the base database, not in the overlay.

DmOpenDatabase

Purpose

Open a database and return a reference to it. If the database is a resource database, also open its overlay for the current locale.

Prototype

```
DmOpenRef DmOpenDatabase (UInt16 cardNo,
LocalID dbID, UInt16 mode)
```

Parameters

-> `cardNo`	Card number database resides on.
-> `dbID`	The database ID of the database.
-> `mode`	Which mode to open database in (see "Open Mode Constants").

Result

Returns `DmOpenRef` to open database. May display a fatal error message if the database parameter is `NULL`. On all other errors, this function returns 0 and `DmGetLastErr` returns an error code indicating the reason for failure.

Comments

Call this routine to open a database for reading or writing.

This routine returns a `DmOpenRef` which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling `DmGetLastErr`.

When you use this routine to open a resource database in read-only mode, it also opens the overlay associated with this database for the current locale, if it exists. (The function `OmGetCurrentLocale` returns the current locale.) Overlays are resource databases typically used to localize applications, shared libraries, and panels. They have the same creator as the base database, a type of `'ovly'` (symbolically named `omOverlayDBType`), and contain resources with the same IDs and types as the resources in the base database. When you request a resource from the database using `DmGetResource` or `DmGet1Resource`, the overlay is searched first. If the overlay contains a resource for the given ID, it is returned. If not, the resource from the base database is returned.

The `DmOpenRef` returned by this function is the pointer to the base database, not to the overlay database, so care should be taken when

passing this pointer to functions such as DmFindResource because this circumvents the overlay.

It's possible to create a "stripped" base resource database, one that does not contain any user interface resources. DmOpenDatabase only opens a stripped database if its corresponding overlay exists. If the overlay does not exist or if the overlay doesn't match the resource database, DmOpenDatabase returns NULL and DmGetLastErr returns the error code omErrBaseRequiresOverlay.

If you open a resource database in a writable mode, the associated overlay is not opened. If you make changes to the resource database, the overlay database is invalidated if those changes affect any resources that are also in the overlay. This means that on future occasions where you open the resource database in read-only mode, the overlay will not be opened because Palm OS considers it to be invalid.

TIP: If you want to prevent your resource database from being overlaid, include an 'xprf' resource (symbolically named sysResTExtPrefs) in the database with the ID 0 (sysResIDExtPrefs) and set its disableOverlays flag. This resource is defined in UIResources.r.

Compatibility

Overlay support is only available if 3.5 New Feature Set is present. On earlier releases, this function opens resource databases without looking for an associated overlay.

See Also

DmCloseDatabase, DmCreateDatabase, DmFindDatabase, DmOpenDatabaseByTypeCreator, DmDeleteDatabase, DmOpenDBNoOverlay

DmOpenDatabaseByTypeCreator

Purpose Open the most recent revision of a database with the given type and creator. If the database is a resource database, also open its overlay for the current locale.

Prototype

```
DmOpenRef DmOpenDatabaseByTypeCreator
(UInt32 type, UInt32 creator, UInt16 mode)
```

Parameters

-> `type`	Type of database.
-> `creator`	Creator of database.
-> `mode`	Which mode to open database in (see "Open Mode Constants").

Result `DmOpenRef` to open database. If the database couldn't be found, this function returns 0 and `DmGetLastErr` returns an error code indicating the reason for failure.

Comments If you use this routine to open a resource database in read-only mode, it also opens the overlay associated with this database for the current locale. See `DmOpenDatabase` for more information on overlays and resource databases.

Compatibility Overlay support is only available if 3.5 New Feature Set is present. On earlier releases, this function opens resource databases without looking for an associated overlay.

See Also `DmCreateDatabase`, `DmOpenDatabase`, `DmOpenDatabaseInfo`, `DmCloseDatabase`, `DmOpenDBNoOverlay`

DmOpenDatabaseInfo

Purpose Retrieve information about an open database.

Prototype

```
Err DmOpenDatabaseInfo (DmOpenRef dbP,
LocalID* dbIDP, UInt16* openCountP, UInt16* modeP,
UInt16* cardNoP, Boolean* resDBP)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
<- `dbIDP`	The ID of the database. Pass `NULL` for this parameter if you don't want to retrieve this information.
<- `openCountP`	The number of applications that have this database open. Pass `NULL` for this parameter if you don't want to retrieve this information.
<- `modeP`	The mode used to open the database (see "Open Mode Constants"). Pass `NULL` for this parameter if you don't want to retrieve this information.
<- `cardNoP`	The number of the card on which this database resides. Pass `NULL` for this parameter if you don't want to retrieve this information.
<- `resDBP`	If `true` upon return, the database is a resource database, `false` otherwise. Pass `NULL` for this parameter if you don't want to retrieve this information.

Result Returns `errNone` if no error.

See Also `DmDatabaseInfo`

DmOpenDBNoOverlay

Purpose Open a database and return a reference to it.

Prototype

```
DmOpenRef DmOpenDBNoOverlay (UInt16 cardNo,
LocalID dbID, UInt16 mode)
```

Parameters

-> `cardNo`	Card number database resides on.
-> `dbID`	The database ID of the database.
-> `mode`	Which mode to open database in (see "Open Mode Constants").

Result `DmOpenRef` to open database. May display a fatal error message if the database parameter is `NULL`. On all other errors, this function returns 0 and `DmGetLastErr` returns an error code indicating the reason for failure.

Comments Call this routine to open a database for reading or writing, while ignoring any overlay databases that might be associated with it.

This routine returns a `DmOpenRef` which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling `DmGetLastErr`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `DmCloseDatabase`, `DmCreateDatabase`, `DmFindDatabase`, `DmOpenDatabaseByTypeCreator`, `DmDeleteDatabase`, `DmOpenDatabase`

DmPositionInCategory

Purpose Return a position of a record within the specified category.

Prototype

```
UInt16 DmPositionInCategory (DmOpenRef dbP,
UInt16 index, UInt16 category)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.

-> `index`	Index of the record.
-> `category`	Index of category to search.

Result Returns the position (zero-based). If the specified index is out of range, this function returns 0 and `DmGetLastErr` returns an error code indicating the reason for failure. Note that this means a 0 return value might indicate either success or failure. If this function returns 0 and `DmGetLastErr` returns `errNone`, the return value indicates that this is the first record in the category.

Comments Because this function must examine all records up to the current record, it can be slow to return, especially when called on a large database.

If the record is ROM-based (pointer accessed) this routine makes a fake handle to it and stores this handle in the `DmAccessType` structure.

To learn which category a record is in, use `DmRecordInfo`.

See Also `DmQueryNextInCategory`, `DmSeekRecordInCategory`, `DmMoveCategory`

DmQueryNextInCategory

Purpose Return a handle to the next record in the specified category for reading only (does not set the `busy` bit).

Prototype

```
MemHandle DmQueryNextInCategory (DmOpenRef dbP,
UInt16* indexP, UInt16 category)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `indexP`	Index of a known record (often retrieved with `DmPositionInCategory`).

-> `category`	Index of category to query, or `dmAllCategories` to find the next record in any category.

Result Returns a handle to the record following a known record. If a record couldn't be found, this function returns `NULL`, and `DmGetLastErr` returns an error code indicating the reason for failure.

Comments This function begins searching the database from the record at `*indexP` for a record that is in the specified category. If the record at `*indexP` belongs to that category, then a handle to it is returned. If not, the function continues searching until it finds a record in the category.

Thus, if you want to find the next record in the category after the one you have an index for, increment the index value before calling this function. For example:

```
DmOpenRef myDB;
UInt16 record, attr, category, pos;
MemHandle newRecH;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
pos = DmPositionInCategory(myDB, record,
  category);
pos++;
newRecH = DmQueryNextInCategory(myDB, &pos,
category);
```

See Also `DmNumRecordsInCategory`, `DmPositionInCategory`, `DmSeekRecordInCategory`

DmQueryRecord

Purpose Return a handle to a record for reading only (does not set the `busy` bit).

Prototype

```
MemHandle DmQueryRecord (DmOpenRef dbP,
UInt16 index)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Which record to retrieve.

Result Returns a record handle. If an error occurs, this function returns `NULL`, and `DmGetLastErr` returns an error code indicating the reason for failure.

Some releases may display a fatal error message if the specified index is out of range.

Comments Returns a handle to the given record. Use this routine only when viewing the record. This routine successfully returns a handle to the record even if the record is busy.

If the record is ROM-based (pointer accessed) this routine returns the fake handle to it.

DmQuickSort

Purpose Sort records in a database.

Prototype

```
Err DmQuickSort (DmOpenRef dbP, DmComparF *compar,
Int16 other)
```

Parameters

-> `dbP`	Database access pointer.
-> `compar`	Comparison function. See `DmComparF`.

-> `other`	Any value the application wants to pass to the comparison function.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrNotRecordDB`

Some releases may display a fatal error message instead of returning the error code.

Comments Deleted records are placed last in any order. All others are sorted according to the passed comparison function.

After `DmQuickSort` returns, equal database records do not have a consistent order. That is, if `DmQuickSort` is passed two equal records, their resulting order is unpredictable. To prevent records that contain the same data from being rearranged in an unpredictable order, pass the record's unique ID to the comparison function (using the `SortRecordInfoType` structure).

See Also `DmFindSortPositionV10`, `DmInsertionSort`

DmRecordInfo

Purpose Retrieve the record information as stored in the database header.

Prototype

```
Err DmRecordInfo (DmOpenRef dbP, UInt16 index,
UInt16* attrP, UInt32* uniqueIDP,
LocalID* chunkIDP)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Index of the record.
<- `attrP`	The record's attributes. See "Record Attribute Constants." Pass `NULL` for this parameter if you don't want to retrieve this value.

<- `uniqueIDP`	The record's unique ID. Pass `NULL` for this parameter if you don't want to retrieve this value.
<- `chunkIDP`	The record's local ID. Pass `NULL` for this parameter if you don't want to retrieve this value.

Result Returns `errNone` if no error or `dmErrIndexOutOfRange` if the specified record can't be found. Some releases may display a fatal error message instead of returning the error code.

See Also `DmNumRecords`, `DmSetRecordInfo`, `DmQueryNextInCategory`

DmReleaseRecord

Purpose Clear the `busy` bit for the given record and set the `dirty` bit if dirty is `true`.

Prototype

```
Err DmReleaseRecord (DmOpenRef dbP, UInt16 index,
Boolean dirty)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	The record to unlock.
-> `dirty`	If `true`, set the `dirty` bit.

Result Returns `errNone` if no error, or `dmErrIndexOutOfRange` if the specified index is out of range. Some releases may display a fatal error message instead of returning the error code.

Comments Call this routine when you finish modifying or reading a record that you've called `DmGetRecord` on or created using `DmNewRecord`.

See Also `DmGetRecord`

DmReleaseResource

Purpose Release a resource acquired with DmGetResource.

Prototype

```
Err DmReleaseResource (MemHandle resourceH)
```

Parameters

-> `resourceH` Handle to resource.

Result Returns `errNone` if no error.

Comments Marks a resource as being no longer needed by the application.

See Also DmGet1Resource, DmGetResource

DmRemoveRecord

Purpose Remove a record from a database and dispose of its data chunk.

Prototype

```
Err DmRemoveRecord (DmOpenRef dbP, UInt16 index)
```

Parameters

-> `dbP` `DmOpenRef` to open database.

-> `index` Index of the record to remove.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrIndexOutOfRange
- dmErrNotRecordDB
- memErrChunkLocked
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code.

Comments Disposes of a the record's data chunk and removes the record's entry from the database header.

See Also DmDetachRecord, DmDeleteRecord, DmArchiveRecord, DmNewRecord

DmRemoveResource

Purpose Delete a resource from a resource database.

Prototype

```
Err DmRemoveResource (DmOpenRef dbP, UInt16 index)
```

Parameters

-> dbP	DmOpenRef to open database.
-> index	Index of resource to delete.

Result Returns errNone if no error, or one of the following if an error occurs:

- dmErrCorruptDatabase
- dmErrIndexOutOfRange
- dmErrReadOnly
- memErrChunkLocked
- memErrInvalidParam
- memErrNotEnoughSpace

May display a fatal error message if the database is not a resource database.

Comments This routine disposes of the memory manager chunk that holds the given resource and removes its entry from the database header.

See Also DmDetachResource, DmRemoveResource, DmAttachResource

DmRemoveSecretRecords

Purpose Remove all secret records.

Prototype

```
Err DmRemoveSecretRecords (DmOpenRef dbP)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.

Result Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrNotRecordDB`

Some releases may display a fatal error message instead of returning the error code.

See Also `DmRemoveRecord`, `DmRecordInfo`, `DmSetRecordInfo`

DmResizeRecord

Purpose Resize a record by index.

Prototype

```
MemHandle DmResizeRecord (DmOpenRef dbP,
UInt16 index, UInt32 newSize)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Which record to retrieve.
-> `newSize`	New size of record.

Result Handle to resized record. Returns `NULL` if there is not enough space to resize the record, and `DmGetLastErr` returns an error code indicating the reason for failure. Some releases may display a fatal error message instead of returning the error code.

Comments This routine reallocates the record in another heap of the same memory card if the current heap is not big enough. If this happens,

the handle changes, so be sure to use the returned handle to access the resized record.

DmResizeResource

Purpose Resize a resource and return the new handle.

Prototype

```
MemHandle DmResizeResource (MemHandle resourceH,
UInt32 newSize)
```

Parameters

-> `resourceH`	Handle to resource.
-> `newSize`	Desired new size of resource.

Result Returns a handle to newly sized resource. Returns `NULL` if there is not enough space to resize the resource, and `DmGetLastErr` returns an error code indicating the reason for failure. Some releases may display a fatal error message instead of returning the error code.

Comments Resizes the resource and returns a new handle. If necessary in order to grow the resource, this routine will reallocate it in another heap on the same memory card that it is currently in.

The handle may change if the resource had to be reallocated in a different data heap because there was not enough space in its present data heap.

DmResourceInfo

Purpose Retrieve information on a given resource.

Prototype

```
Err DmResourceInfo (DmOpenRef dbP, UInt16 index,
DmResType* resTypeP, DmResID* resIDP,
LocalID* chunkLocalIDP)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Index of resource to get info on.

<- `resTypeP`	The resource type. Pass `NULL` if you don't want to retrieve this information.
<- `resIDP`	The resource ID. Pass `NULL` if you don't want to retrieve this information.
<- `chunkLocalIDP`	The memory manager local ID of the resource data. Pass `NULL` if you don't want to retrieve this information.

Result Returns `errNone` if no error or `dmErrIndexOutOfRange` if an error occurred. May display a fatal error message if the database is not a resource database.

Comments If `dbP` is a pointer to a base resource database, the information returned is about the resource from that database alone; this function ignores any associated overlay.

See Also `DmGetResource`, `DmGet1Resource`, `DmSetResourceInfo`, `DmFindResource`, `DmFindResourceType`

DmSearchRecord

Purpose Search all open record databases for a record with the handle passed.

Prototype

```
UInt16 DmSearchRecord (MemHandle recH,
DmOpenRef* dbPP)
```

Parameters

-> `recH`	Record handle.
<- `dbPP`	The database that contains the record `recH`.

Result Returns the index of the record and database access pointer; if not found, returns -1 and `*dbPP` is 0.

See Also `DmGetRecord`, `DmFindRecordByID`, `DmRecordInfo`

DmSearchResource

Purpose Search all open resource databases for a resource by type and ID, or by pointer if it is non-`NULL`.

Prototype

```
UInt16 DmSearchResource (DmResType resType,
DmResID resID, MemHandle resH, DmOpenRef* dbPP)
```

Parameters

-> `resType`	Type of resource to search for.
-> `resID`	ID of resource to search for.
-> `resH`	Pointer to locked resource, or `NULL`.
<- `dbPP`	The resource database that contains the specified resource.

Result Returns the index of the resource, stores `DmOpenRef` in `dbPP`.

Comments This routine can be used to find a resource in all open resource databases by type and ID or by pointer. If `resH` is `NULL`, the resource is searched for by type and ID. If `resH` is not `NULL`, `resType` and `resID` is ignored and the index of the resource handle is returned. On return, `*dbPP` contains the access pointer of the resource database that the resource was eventually found in. Once the index of a resource is determined, it can be locked down and accessed by calling `DmGetResourceIndex`.

If any of the open databases are overlaid, this function finds and returns the localized version of the resource when searching by type and creator. In this case, the `dbPP` return value is a pointer to the overlay database, not the base resource database.

See Also `DmGetResource`, `DmFindResourceType`, `DmResourceInfo`, `DmFindResource`

DmSeekRecordInCategory

Purpose Return the index of the record nearest the offset from the passed record index whose category matches the passed category. (The

offset parameter indicates the number of records to move forward or backward.)

Prototype

```
Err DmSeekRecordInCategory (DmOpenRef dbP,
UInt16* indexP, Int16 offset, Int16 direction,
UInt16 category)
```

Parameters

-> dbP	DmOpenRef to open database.
<-> index	The index to start the search at. Upon return, contains the index of the record at offset from the index that you passed in.
-> offset	Offset of the passed record index. This must be a positive number; use dmSeekBackward for the direction parameter to search backwards.
-> direction	Must be either dmSeekForward or dmSeekBackward.
-> category	Category index.

Result

Returns errNone if no error; returns dmErrIndexOutOfRange or dmErrSeekFailed if an error occurred.

Comments

DmSeekRecordInCategory searches for a record in the specified category. The search begins with the record at index. When it finds a record in the specified category, it decrements the offset parameter and continues searching until a match is found and offset is 0.

Because of this, if you use DmSeekRecordInCategory to find the nearest matching record in a particular category, you must pass different offset parameters if the starting record is in the category than if it isn't. If the record at index is in the category, then you must pass an offset of 1 to find the next record in the category because the comparison is performed before the index value changes. If the record at index isn't in the category, you must pass

an `offset` of 0 to find the next record in the category. In this case, an `offset` of 1 skips the first matching record.

See Also `DmNumRecordsInCategory`, `DmQueryNextInCategory`, `DmPositionInCategory`, `DmMoveCategory`

DmSet

Purpose Write a specified value into a section of a record. This function also checks the validity of the pointer for the record and makes sure the writing of the record information doesn't exceed the bounds of the record.

Prototype

```
Err DmSet (void* recordP, UInt32 offset,
UInt32 bytes, UInt8 value)
```

Parameters

-> `recordP`	Pointer to locked data record (chunk pointer).
-> `offset`	Offset within record to start writing.
-> `bytes`	Number of bytes to write.
-> `value`	Byte value to write.

Result Returns `errNone` if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

Comments Must be used to write to data manager records because the data storage area is write-protected.

See Also `DmWrite`

DmSetDatabaseInfo

Purpose Set information about a database.

Prototype

```
Err DmSetDatabaseInfo (UInt16 cardNo,
LocalID dbID, const Char* nameP,
UInt16* attributesP, UInt16* versionP,
UInt32* crDateP, UInt32* modDateP,
UInt32* bckUpDateP, UInt32* modNumP,
LocalID* appInfoIDP, LocalID* sortInfoIDP,
UInt32* typeP, UInt32* creatorP)
```

Parameters

-> `cardNo`	Card number the database resides on.
-> `dbID`	Database ID of the database.
-> `nameP`	Pointer to 32-byte character array for new name, or `NULL`.
-> `attributesP`	Pointer to new attributes variable, or `NULL`. See "Database Attribute Constants" for a list of possible values.
-> `versionP`	Pointer to new version, or `NULL`.
-> `crDateP`	Pointer to new creation date variable, or `NULL`. Specify the value as a number of seconds since Jan 1, 1904.
-> `modDateP`	Pointer to new modification date variable, or `NULL`. Specify the value as a number of seconds since Jan 1, 1904.
-> `bckUpDateP`	Pointer to new backup date variable, or `NULL`. Specify the value as a number of seconds since Jan 1, 1904.
-> `modNumP`	Pointer to new modification number variable, or `NULL`.
-> `appInfoIDP`	Pointer to new `appInfoID` variable, or `NULL`.
-> `sortInfoIDP`	Pointer to new `sortInfoID` variable, or `NULL`.
-> `typeP`	Pointer to new `type` variable, or `NULL`.

-> `creatorP`	Pointer to new `creator` variable, or `NULL`.

Result Returns `errNone` if no error or one of the following if an error occurred:

- `dmErrInvalidDatabaseName`
- `dmErrAlreadyExists`
- `dmErrInvalidParam`

Comments When this call changes `appInfoID` or `sortInfoID`, the old chunk ID (if any) is marked as an orphan chunk and the new chunk ID is unorphaned. Consequently, you shouldn't replace an existing `appInfoID` or `sortInfoID` if that chunk has already been attached to another database.

Call this routine to set any or all information about a database except for the card number and database ID. This routine sets the new value for any non-`NULL` parameter.

See Also `DmDatabaseInfo`, `DmOpenDatabaseInfo`, `DmFindDatabase`, `DmGetNextDatabaseByTypeCreator`, `TimDateTimeToSeconds`

DmSetRecordInfo

Purpose Set record information stored in the database header.

Prototype

```
Err DmSetRecordInfo (DmOpenRef dbP, UInt16 index,
UInt16* attrP, UInt32* uniqueIDP)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Index of record.
-> `attrP`	Pointer to new attribute variable, or `NULL`. See "Record Attribute Constants" for a list of possible values.

-> `uniqueIDP`	Pointer to new unique ID variable, or `NULL`.

Result Returns `errNone` if no error, or one of the following if an error occurred:

- `dmErrReadOnly`
- `dmErrNotRecordDB`
- `dmErrIndexOutOfRange`

Some releases may display a fatal error message instead of returning the error code.

Comments Sets information about a record.

Normally, the unique ID for a record is automatically created by the data manager when a record is created using `DmNewRecord`, so an application would not typically change the unique ID.

See Also `DmNumRecords`, `DmRecordInfo`

DmSetResourceInfo

Purpose Set information on a given resource.

Prototype

```
Err DmSetResourceInfo (DmOpenRef dbP,
UInt16 index, DmResType* resTypeP,
DmResID* resIDP)
```

Parameters

-> `dbP`	`DmOpenRef` to open database.
-> `index`	Index of resource to set info for.
-> `resTypeP`	Pointer to new `resType` (resource type), or `NULL`.
-> `resIDP`	Pointer to new resource ID, or `NULL`.

Result Returns `errNone` if no error, or one of the following if an error occurred:

- `dmErrIndexOutOfRange`
- `dmErrReadOnly`

May display a fatal error message if the database is not a resource database.

Comments Use this routine to set all or a portion of the information on a particular resource. Any or all of the new info pointers can be `NULL`. If not `NULL`, the type and ID of the resource are changed to `*resTypeP` and `*resIDP`.

DmStrCopy

Purpose Check the validity of the chunk pointer for the record and make sure that writing the record will not exceed the chunk bounds.

Prototype

```
Err DmStrCopy (void* recordP, UInt32 offset,
const Char * srcP)
```

Parameters

<-> `recordP`	Pointer to data record (chunk pointer).
-> `offset`	Offset within record to start writing.
-> `srcP`	Pointer to null-terminated string.

Result Returns `errNone` if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

See Also `DmWrite`, `DmSet`

DmWrite

Purpose Must be used to write to data manager records because the data storage area is write-protected. This routine checks the validity of the chunk pointer for the record and makes sure that the write will not exceed the chunk bounds.

Prototype

```
Err DmWrite (void* recordP, UInt32 offset,
const void * srcP, UInt32 bytes)
```

Parameters

<-> `recordP`	Pointer to locked data record (chunk pointer).
-> `offset`	Offset within record to start writing.

-> `srcP`	Pointer to data to copy into record.
-> `bytes`	Number of bytes to write.

Result Returns `errNone` if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

See Also `DmSet`

DmWriteCheck

Purpose Check the parameters of a write operation to a data storage chunk before actually performing the write.

Prototype

```
Err DmWriteCheck (void* recordP, UInt32 offset,
UInt32 bytes)
```

Parameters

-> `recordP`	Locked pointer to `recordH`.
-> `offset`	Offset into record to start writing.
-> `bytes`	Number of bytes to write.

Result Returns `errNone` if no error; returns `dmErrNotValidRecord` or `dmErrWriteOutOfBounds` if an error occurred.

Application-Defined Functions

DmComparF

Purpose Compares two records in a database.

Prototype

```
typedef Int16 DmComparF (void* rec1, void* rec2,
Int16 other, SortRecordInfoPtr rec1SortInfo,
SortRecordInfoPtr rec2SortInfo,
MemHandle appInfoH)
```

Parameters

-> `rec1`, `rec2`	Pointers to the two records to sort.

-> `other`	Any other custom information you want passed to the comparison function.
-> `rec1SortInfo`, `rec2SortInfo`	Pointers to `SortRecordInfoType` structures that specify unique sorting information for the two records.
-> `appInfoH`	A handle to the database's application info block.

Result

Returns:

- 0 if `rec1` = `rec2`.
- < 0 if `rec1` < `rec2`.
- > 0 if `rec1` > `rec2`.

Comments

This function is used to sort the records in a database. It is specifically called by `DmFindSortPosition`, `DmInsertionSort`, and `DmQuickSort`.

Compatibility

The `DmComparF` prototype changed in Palm OS® version 2.0. Previously, the prototype only defined the first three parameters.

As a rule, the change in the number of arguments from three to six doesn't cause problems when a 1.0 application is run on a 2.0 device because the system only pulls the arguments from the stack that are there.

Keep in mind, however, that some optimized applications built with tools other than Metrowerks CodeWarrior for Palm OS may have problems as a result of the change in arguments when running on a 2.0 or later device.

27

Time Manager

This chapter provides reference material for the time manager.

- Time Manager Data Structures
- Time Manager Functions

The header file `DateTime.h` declares the API that this chapter describes. For more information on the time manager, see the section "Time" in the *Palm OS Programmer's Companion*.

Time Manager Data Structures

The time manager uses these structures to store information.

TimeFormatType

```
typedef enum
  {
  tfColon,
     tfColonAMPM,    // 1:00 pm
     tfColon24h,     // 13:00
     tfDot,
     tfDotAMPM,      // 1.00 pm
     tfDot24h,       // 13.00
     tfHoursAMPM,    // 1 pm
     tfHours24h,     // 13
     tfComma24h      // 13,00
  } TimeFormatType;

typedef TimeFormatType* TimeFormatPtr;
```

DaylightSavingsTypes

```
typedef enum {
  dsNone,            //Daylight Savings Time not
                     //observed
```

```
  dsUSA,            //United States Daylight
                    //Savings Time
  dsAustralia,      //Australian Daylight
                    //Savings Time
  dsWesternEuropean,//Western European Daylight
                    //Savings Time
  dsMiddleEuropean, //Middle European Daylight
                    //Savings Time
  dsEasternEuropean,//Eastern European Daylight
                    //Savings Time
  dsGreatBritain,   //Great Britain and Eire
                    //Daylight Savings Time
  dsRumania,        //Rumanian Daylight Savings
                    //Time
  dsTurkey,         //Turkish Daylight Savings
                    //Time
  dsAustraliaShifted//Australian Daylight
                    //Savings Time with shift
                    //in 1986
} DaylightSavingsTypes;
```

DateFormatType

```
typedef enum {
   dfMDYWithSlashes, // 12/31/95
   dfDMYWithSlashes, // 31/12/95
   dfDMYWithDots,    // 31.12.95
   dfDMYWithDashes,  // 31-12-95
   dfYMDWithSlashes, // 95/12/31
   dfYMDWithDots,    // 95.12.31
   dfYMDWithDashes,  // 95-12-31

   dfMDYLongWithComma,// Dec 31, 1995
   dfDMYLong,         // 31 Dec 1995
   dfDMYLongWithDot,  // 31. Dec 1995
   dfDMYLongNoDay,    // Dec 1995
   dfDMYLongWithComma,// 31 Dec, 1995
   dfYMDLongWithDot,  // 1995.12.31
   dfYMDLongWithSpace,// 1995 Dec 31
```

```
    dfMYMed,          // Dec '95
    dfMYMedNoPost     // Dec 95
                      //added for French 2.0 ROM)
   } DateFormatType;
```

DateTimeType

```
typedef struct{
Int16   second;
Int16   minute;
Int16   hour;
Int16   day;
Int16   month;
Int16   year;
Int16   weekDay;  //Days since Sunday (0 to 6)
}DateTimeType;

typedef DateTimeType* DateTimePtr;
```

TimeType

```
typedef struct {
UInt8   hours;
UInt8   minutes;
}TimeType;

typedef TimeType * TimePtr;
```

DateType

```
typedef struct{
UInt16 year :7; //years since 1904 (Mac format)
UInt16 month:4;
UInt16 day  :5;
}DateType;

typedef DateType * DatePtr;
```

Time Manager Constants

Maximum lengths of strings returned by the date and time formatting routines DateToAscii, DateToDOWDMFormat, and TimeToAscii.

Constant	Value	Description
dateStringLength	9	Max length of string returned by DateToAscii for short date formats.
longDateStrLength	15	Max length of string returned by DateToAscii for medium and long date formats.
timeStringLength	9	Max length of string returned by TimeToAscii.
dowDateStringLength	19	Max length of string returned by DateToDOWDMFormat for short date formats.
dowLongDateStrLength	25	Max length of string returned by DateToDOWDMFormat for long date formats

Time Manager Functions

DateAdjust

Purpose Return a new date +/- the days adjustment.

Prototype

```
void DateAdjust (DatePtr dateP, Int32 adjustment)
```

Parameters

`dateP`	A "DateType" structure with the date to be adjusted (see `DateTime.h`).
`adjustment`	The adjustment in number of days.

Result Changes `dateP` to contain the new date.

Comments This function is useful for advancing a day or week and not worrying about month and year wrapping.

If the time is advanced out of bounds, it is cut at the bounds surpassed.

DateDaysToDate

Purpose Return the date, given days.

Prototype

```
void DateDaysToDate (UInt32 days, DatePtr date)
```

Parameters

`days`	Days since 1/1/1904.
`date`	Pointer to "DateType" structure (returned).

Result Returns nothing, stores the date in `date`.

See Also `TimAdjust`, `DateToDays`

DateSecondsToDate

Purpose Return the date given seconds.

Prototype

```
void DateSecondsToDate (UInt32 seconds,
DatePtr date)
```

Parameters

`seconds`	Seconds since 1/1/1904.
`date`	Pointer to "DateType" structure (returned).

Result Returns nothing; stores the date in `date`.

DateTemplateToAscii

Purpose Convert the date passed to a string, using the formatting specified by the `templateP` string.

Prototype

```
UInt16 DateTemplateToAscii(const Char *templateP,
UInt8 months, UInt8 days, UInt16 years, Char
*stringP, Int16 stringLen)
```

Parameters

`templateP`	pointer to template string used to format date. Max length is `maxDateTemplateLen` bytes, excluding the terminating null.
`months`	months (1-12)
`days`	days (1-31)
`years`	years. For example, 1995.
`stringP`	pointer to string which gets the result. Up to `stringLen` bytes (excluding the terminating null byte.) Can be NULL, in which case the required string length is still returned by the function. If not NULL, then the formatted string is written to it,
`stringLen`	size of string buffer, excluding the terminating NULL byte.

Result The length of the formatted string (without the terminating null byte) is always returned, even if the `stringP` parameter is null.

This then lets you allocate a buffer at run time, without having to previously fix it to some max size.

Comments The `stringP` parameter can be NULL, in which case the required string length is still returned.

NOTE: This routine is only available in PalmOS 3.5 or later ROMs.

This routine uses the template text contained in `templateP`, and creates a properly formatted date string in `stringP` for the values passed in `months`, `days`, and `years`.

Template strings:

A template string contains a mixture of regular text and formatting substrings. Each formatting substring has the format

"^<number><modifier>". The possible values for number are:

```
dateTemplateDayNum = '0',        // Day number (1..31)
dateTemplateDOWName,             // Day name (e.g. Tue)
dateTemplateMonthName,           // Month name (e.g. Aug)
dateTemplateMonthNum,            // Month number (1..12)
dateTemplateYearNum              // Year (e.g. 1995)
```

The possible values for modifier are:

```
#define dateTemplateShortModifier        's'
#define dateTemplateRegularModifier      'r'
#define dateTemplateLongModifier         'l'
#define dateTemplateLeadZeroModifier     'z'
```

The meaning of each modifier depends on what type of formatting string it's part of. An example of each is as follows:

Format	Short	Regular	Long	Zero
DayNum	5	5	5	05
DOWName	T	Tue	Tuesday	n/a
MonthName	A	Aug	August	n/a
MonthNum	8	8	8	08
YearNum	00	2000	2000	n/a

So, for example, the formatting string to get "02 February 2000" would be:

```
"^0z ^2l ^4r"
```

See Also `DateToAscii`, `DateToDOWDMFormat`

DateToAscii

Purpose Convert the date passed to a string using the format specified by the `dateFormat` parameter.

Prototype

```
void DateToAscii (UInt8 months, UInt8 days,
UInt16 years, DateFormatType dateFormat,
Char* pString)
```

Parameters

`months`	Months (1-12).
`days`	Days (1-31).
`years`	Years (for example 1995).
`dateFormat`	Any "DateFormatType" format.
`pString`	Pointer to string which gets the result. Must be of length `dateStringLength` for standard formats or `longDateStrLength` for medium or long formats. (See "Time Manager Constants" for allowed lengths.

Result Returns nothing. Stores the result in `pString`.

Comments

NOTE: If you are using a debug ROM, the string buffer is filled with `dateStringLength` or `longStrLength` debugging bytes, depending on the dateFormat parameter.

Common situations where buffers overflow on debug ROMs include stack-based or global variables, form titles, form labels, and control labels. Overflowing a form object is sometimes very hard to catch, because often what happens is that the following form

object's data (such as its position) is overwritten, and so suddenly the form object disappears.

See Also TimeToAscii, DateToDOWDMFormat, DateTemplateToAscii

DateToDays

Purpose Return the date in days since 1/1/1904.

Prototype

```
UInt32 DateToDays (DateType date)
```

Parameters

date	"DateType" structure.

Result Returns the days since 1/1/1904.

See Also TimAdjust, DateDaysToDate

DateToDOWDMFormat

Purpose Convert a date to a formatted string using the format specified by the dateFormat parameter. The string passed must include the name of the day of the week.

Prototype

```
void DateToDOWDMFormat (UInt8 month, UInt8 day,
UInt16 year, DateFormatType dateFormat,
Char * pString)
```

Parameters

month	Month (1-12).
day	Days (1-31).
year	Years (for example 1995).
dateFormat	Any "DateFormatType" format.

`pString`	Pointer to string which gets the result. Must be of length `dateStringLength` for standard formats or `longDateStrLength` for medium or long date formats. (See "Time Manager Constants" for string buffer lengths.)

Result Returns nothing; stores formatted string in `pString`.

Comments For the routines that return the day-of-week name in addition to the date, the size of the buffers has been expanded, so developers need to check the maximum lengths defined in `DateTime.h`.

Common situations where buffers overflow on debug ROMs include stack-based or global variables, form titles, form labels, and control labels. Overflowing a form object is sometimes very hard to catch, because often what happens is that the following form object's data (such as its position) is overwritten, and so suddenly the form object disappears.

See Also `DateToAscii`, `DateTemplateToAscii`

DayOfMonth

Purpose Return the day of a month on which the specified date occurs.

Prototype

```
Int16 DayOfMonth (Int16 month, Int16 day,
Int16 year)
```

Parameters

`month`	Month (1-12).
`day`	Day (1-31).
`year`	Year (for example 1995).

Result Returns the day of the month; see `DateTime.h`.

Comments For example, "first Monday" is returned for 2/7/00.

DayOfWeek

Purpose Return the day of the week.

Prototype

```
Int16 DayOfWeek (Int16 month, Int16 day,
Int16 year)
```

Parameters

`month`	Month (1-12).
`day`	Day (1-31).
`year`	Year (for example 1995).

Result Returns the day of the week (Sunday = 0, Monday = 1, etc.).

DaysInMonth

Purpose Return the number of days in the month.

Prototype

```
Int16 DaysInMonth (Int16 month, Int16 year)
```

Parameters

`month`	Month (1-12).
`year`	Year (for example, 1995).

Result Returns the number of days in the month for that year.

TimAdjust

Purpose Return a new date, +/- the time adjustment.

Prototype

```
void TimAdjust (DateTimePtr dateTimeP,
Int32 adjustment)
```

Parameters

`dateTimeP`	A "DateType" structure (see `DateTime.h`).
`adjustment`	The adjustment in seconds.

Result Returns nothing. Changes `dateTimeP` to the new date and time.

Comments This function is useful for advancing a day or week and not worrying about month and year wrapping.

If the time is advanced out of bounds it is cut at the bounds surpassed.

See Also DateAdjust

TimDateTimeToSeconds

Purpose Return the number of seconds since 1/1/1904 to the passed date and time.

Prototype

```
UInt32 TimDateTimeToSeconds
(DateTimePtr dateTimeP)
```

Parameters

`dateTimeP`	Pointer to a "DateTimeType" structure (see `DateTime.h`).

Result The time in seconds since 1/1/1904.

See Also TimSecondsToDateTime

TimGetSeconds

Purpose Return the current date and time of the device in seconds since 1/1/1904 12AM.

Prototype

```
UInt32 TimGetSeconds (void)
```

Parameters None.

Result Returns the number of seconds.

See Also TimSetSeconds

TimGetTicks

Purpose Return the tick count since the last reset. The tick count does not advance while the device is in sleep mode.

Prototype

```
UInt32 TimGetTicks (void)
```

Parameters None.

Result Returns the tick count.

Comments Use to determine the number of ticks per second.

TimSecondsToDateTime

Purpose Return the date and time, given seconds.

Prototype

```
void TimSecondsToDateTime (UInt32 seconds,
DateTimePtr dateTimeP)
```

Parameters

`seconds`	Seconds to advance from 1/1/1904.
`dateTimeP`	A "DateTimeType" structure that's filled by the function.

Result Returns nothing. Stores the date and time given seconds since 1/1/1904 in `dateTimeP`.

See Also `TimDateTimeToSeconds`

TimSetSeconds

Purpose Set the clock of the device to the date and time passed as the number of seconds since 1/1/1904 12AM.

Prototype

```
void TimSetSeconds (UInt32 seconds)
```

Parameters

`seconds`	The seconds since 1/1/1904.

Result Returns nothing.

Comments On systems where the Notification Feature Set is present, this function broadcasts the `sysNotifyTimeChangeEvent` to all interested parties. See the "Notification Manager" chapter for more information.

See Also `TimGetSeconds`

TimeToAscii

Purpose Convert the time passed to a formatted string.

Prototype

```
void TimeToAscii (UInt8 hours, UInt8 minutes,
TimeFormatType timeFormat, Char* pString)
```

Parameters

`hours`	Hours (0-23).
`minutes`	Minutes (0-59).
`timeFormat`	`FALSE` to use AM and PM.
`pString`	Pointer to string which gets the result. Must be of length `timeStringLength`. See "Time Manager Constants" for information on string buffer lengths.

Result Returns nothing. Stores the formatted string in `pString`.

Comments

NOTE: If you are using a debug ROM in PalmOS 3.5, the string buffer is filled with timeStringLength debugging bytes.

For the routines that return the day-of-week name in addition to the date, the size of the buffers has been expanded, so developers need to check the maximum lengths defined in `DateTime.h`. See "Time Manager Constants".

See Also `DateToAscii`

28

Error Manager

This chapter provides reference material for the error manager. The error manager API is declared in the header files `ErrorMgr.h` and `ErrorBase.h`. This chapter covers:

- ERROR_CHECK_LEVEL Define
- Error Manager Functions

For more information on the error manager, see the chapter "Debugging Strategies" in the *Palm OS Programmer's Companion*.

ERROR_CHECK_LEVEL Define

The error manager uses the compiler define `ERROR_CHECK_LEVEL` to control the level of error messages displayed. You can set the value of the compiler define to control which level of error checking and display is compiled into the application. Three levels of error checking are supported: none, partial, and full.

If you set `ERR_CHECK_LEVEL` **to...**	**The compiler...**
`ERROR_CHECK_NONE (0)`	Doesn't compile in any error calls.
`ERROR_CHECK_PARTIAL (1)`	Compiles in only `ErrDisplay` and `ErrFatalDisplayIf` calls.
`ERROR_CHECK_FULL (2)`	Compiles in all three calls.

Error Manager Functions

ErrAlert

Purpose Macro that displays an alert dialog for runtime errors.

Prototype

```
ErrAlert (err)
```

Parameters

-> `err`	An error code (as type `Err`).

Result Returns 0, which indicates that the OK button has been clicked to dismiss the dialog.

Comments This macro is intended for use by applications that are likely to receive runtime errors when the application itself is not at fault. For example, a networking application might use it to display an alert if the remote server cannot be found.

The error message displayed on the dialog is stored in a 'tSTL' resource. A 'tSTL' resource contains a list of strings that can be looked up by index. The `err` parameter is used as the index into this list.

To use application-defined error codes in `ErrAlert`, make sure that all of your error codes are greater than or equal to `appErrorClass`. This way, the error manager looks up the code in the application's 'tSTL' resource number 0. All other error codes are taken from 'tSTL' resource stored in the system.

Compatibility Implemented only if 3.2 New Feature Set is present.

ErrDisplay

Purpose Macro that displays an error alert if error checking is set to partial or full.

Prototype

```
ErrDisplay (msg)
```

Parameters

-> `msg`	Error message text as a string.

Result No return value.

Comments Call this macro to display an error message, source code filename, and line number. This macro is compiled into the code only if the compiler define `ERROR_CHECK_LEVEL` is set to 1 or 2 (`ERROR_CHECK_PARTIAL` or `ERROR_CHECK_FULL`).

See Also `ErrFatalDisplayIf`, `ErrNonFatalDisplayIf`

ErrDisplayFileLineMsg

Purpose Display a nonexitable dialog with an error message. Do not allow the user to continue.

Prototype

```
void ErrDisplayFileLineMsg
(const Char * const filename, UInt16 lineno,
const Char * const msg)
```

Parameters

-> `filename`	Source code filename.
-> `lineno`	Line number in the source code file.
-> `msg`	Message to display.

Result Never returns.

Comment Called by `ErrFatalDisplayIf` and `ErrNonFatalDisplayIf`. This function is useful when the application is already on the device and being tested by users.

On Japanese systems, the system displays a generic message indicating that an error has occurred instead of displaying the English message.

See Also ErrFatalDisplayIf, ErrNonFatalDisplayIf, ErrDisplay

ErrFatalDisplayIf

Purpose Macro that displays an error alert dialog if `condition` is `true` and error checking is set to partial or full.

Prototype

```
ErrFatalDisplayIf (condition, msg)
```

Parameters

-> `condition`	A boolean value. If `true`, display the error.
-> `msg`	Error message text as a string.

Result No return value.

Comments Call this macro to display a fatal error message, source code filename, and line number. The alert is displayed only if `condition` is `true`. The dialog is cleared only when the user resets the system by responding to the dialog.

This macro is compiled into the code if the compiler define `ERROR_CHECK_LEVEL` is set to 1 or 2 (`ERROR_CHECK_PARTIAL` or `ERROR_CHECK_FULL`).

See Also `ErrNonFatalDisplayIf`, `ErrDisplay`,

ErrNonFatalDisplayIf

Purpose Macro that displays an error alert dialog if condition is `true` and error checking is set to full.

Prototype

```
ErrNonFatalDisplayIf (condition, msg)
```

Parameters

-> `condition`	A boolean value. If `true`, display the error.

-> `msg` Error message text as a string.

Result No return value.

Comments Call this macro to display a nonfatal error message, source code filename, and line number. The alert is displayed only if `condition` is `true`. The alert dialog is cleared when the user selects to continue (or resets the system).

This macro is compiled into the code only if the compiler define `ERROR_CHECK_LEVEL` is set to 2 (`ERROR_CHECK_FULL`).

See Also `ErrFatalDisplayIf`, `ErrDisplay`,

ErrThrow

Purpose Cause a jump to the nearest Catch block.

Prototype

```
void ErrThrow (Int32 err)
```

Parameters -> `err` Error code.

Result Never returns.

Comments Use the macros ErrTry, ErrCatch, and ErrEndCatch in conjunction with this function.

See Also `ErrFatalDisplayIf`, `ErrNonFatalDisplayIf`, `ErrDisplay`

29

Feature Manager

This chapter provides reference material for the feature manager. The feature manager API is declared in the header file `FeatureMgr.h`.

For more information on the feature manager, see the section "Features" in the *Palm OS Programmer's Companion*.

To learn how to use the predefined Palm OS® features to test for the existence of certain OS features, see the "Compatibility Guide" appendix.

Feature Manager Functions

FtrGet

Purpose Get a feature.

Prototype

```
Err FtrGet (UInt32 creator, UInt16 featureNum,
UInt32 *valueP)
```

Parameters

`-> creator`	Creator ID, which must be registered with Palm Computing®. This is usually the same as the creator ID for the application that owns this feature.
`-> featureNum`	Feature number of the feature.
`<- valueP`	Value of the feature is returned here.

Result Returns 0 if no error, or `ftrErrNoSuchFtr` if the specified feature number doesn't exist for the specified creator.

Comments The value of the feature is application-dependent.

See Also FtrSet

FtrGetByIndex

Purpose Get a feature by index.

Prototype

```
Err FtrGetByIndex (UInt16 index, Boolean romTable,
UInt32 *creatorP, UInt16 *numP, UInt32 *valueP)
```

Parameters

`-> index`	Index of feature.
`-> romTable`	If `true`, index into ROM table; otherwise, index into RAM table.
`<- creatorP`	Feature creator is returned here.
`<- numP`	Feature number is returned here.
`<- valueP`	Feature value is returned here.

Result Returns 0 if no error, or `ftrErrNoSuchFeature` if the index is out of range.

Comments This function is intended for system use only. It is used by shell commands. Most applications don't need it.

Until the caller gets back `ftrErrNoSuchFeature`, it should pass indices for each table (ROM, RAM) starting at 0 and incrementing. Note that in Palm OS 3.1 and higher, the RAM feature table serves the entire system. At system startup, the values in the ROM feature table are copied into the RAM feature table.

FtrPtrFree

Purpose Release memory previous allocated with FtrPtrNew.

Prototype

```
Err FtrPtrFree (UInt32 creator, UInt16 featureNum)
```

Parameters

`-> creator`	The creator ID for the feature.

`-> featureNum`	Feature number of the feature.

Result Returns 0 if no error, or `ftrErrNoSuchFtr` if an error occurs.

Comments This function unregisters the feature before freeing the memory associated with it.

Compatibility Implemented only if 3.1 New Feature Set is present.

FtrPtrNew

Purpose Allocate feature memory.

Prototype

```
Err FtrPtrNew (UInt32 creator, UInt16 featureNum,
UInt32 size, void **newPtrP)
```

Parameters

`-> creator`	Creator ID, which must be registered with Palm Computing. This is usually the same as the creator ID for the application that owns this feature.
`-> featureNum`	Feature number of the feature.
`-> size`	Size in bytes of the temporary memory to allocate. The maximum chunk size is 64K.
`<- newPtrP`	Pointer to the memory chunk is returned here.

Result Returns 0 if no error, `memErrInvalidParam` if the value of `size` is 0, or `memErrNotEnoughSpace` if there is not enough space to allocate a chunk of the specified size.

Comments This function allocates a chunk of memory and stores a pointer to that chunk in the feature table. The same pointer is returned in `newPtrP`. The memory chunk remains allocated and locked until the next system reset or until you free the chunk with `FtrPtrFree`.

`FtrPtrNew` is useful if you want quick, efficient access to data that persists from one invocation of the application to the next. `FtrPtrNew` stores values on the storage heap rather than the dynamic heap, where free space is often extremely limited. The

disadvantage to using feature memory is that writing to storage memory is slower than writing to dynamic memory.

NOTE: Starting with `Palm OS 3.5 FtrPtrNew` allows allocating chunks larger than 64k. Do keep in mind standard issues with allocating large chunks of memory: there might not be enough contiguous space, and it can impact system performance.

You can obtain the pointer to the chunk using `FtrGet`. To write to the chunk, you **must** use `DmWrite` because the chunk is in the storage heap, not the dynamic heap.

For example, if you allocate a memory chunk in this way:

```
FtrPtrNew(appCreator,
    myFtrMemFtr, 32, &ftrMem);
```

You can later access that memory and write to it using the following:

```
void* data;
if (!FtrGet(appCreator,
    myFtrMemFtr, (UInt32*)&data))
  DmWrite(data, 0, &someVal, sizeof(someVal));
```

Compatibility Implemented only if 3.1 New Feature Set is present.

See Also `FtrPtrResize`

FtrPtrResize

Purpose Resize feature memory.

Prototype

```
Err FtrPtrResize (UInt32 creator,
UInt16 featureNum, UInt32 newSize, void **newPtrP)
```

Parameters

`-> creator`	The creator ID for the feature.
`-> featureNum`	Feature number of the feature.
`-> newSize`	New size in bytes for the chunk.

`<- newPtrP`	Pointer to the memory chunk is returned here.

Result Returns 0 if no error, or `ftrErrNoSuchFtr` if the specified feature number doesn't exist for the specified creator, `memErrInvalidParam` if `newSize` is 0, or `memErrNotEnoughSpace` if there's not enough free space available to allocate a chunk of that size.

Comments Use this function to resize a chunk of memory previously allocated by `FtrPtrNew`.

This function may move the chunk to a new location in order to resize it, so it is important to use the pointer returned by this function when accessing the memory chunk. The pointer in the feature table is automatically updated to be the same as the pointer returned by this function.

If this function fails, the old memory pointer still exists and its data is unchanged.

Compatibility Implemented only if 3.1 New Feature Set is present.

See Also `MemHandleResize`

FtrSet

Purpose Set a feature.

Prototype

```
Err FtrSet (UInt32 creator, UInt16 featureNum,
UInt32 newValue)
```

Parameters

`-> creator`	Creator ID, which must be registered with Palm Computing. This is usually the same as the creator ID for the application that owns this feature.
`-> featureNum`	Feature number for this feature.

`-> newValue` New value.

Result Returns 0 if no error, or `memErrNotEnoughSpace` if the feature table must be resized to add a new feature and no space is available.

Comments The value of the feature is application-dependent.

A feature that you define in this manner remains defined until the next system reset or until you explicitly undefine the feature with `FtrUnregister`.

See Also `FtrGet`, `FtrPtrNew`

FtrUnregister

Purpose Unregister a feature.

Prototype

```
Err FtrUnregister (UInt32 creator,
UInt16 featureNum)
```

Parameters

`-> creator` Creator ID for the feature.

`-> featureNum` Feature number of the feature.

Result Returns 0 if no error, or `ftrErrNoSuchFeature` if the specified feature number doesn't exist for the specified creator.

30

File Streaming

This chapter provides reference material for the file streaming API.

- File Streaming Constants
- File Streaming Functions
- File Streaming Error Codes

The header file `FileStream.h` declares the API that this chapter describes. For more information on file streaming, see the chapter "Files and Databases" in the *Palm OS Programmer's Companion*.

File Streaming Constants

Primary Open Mode Constants

This section lists constants passed in the `openMode` parameter to the `FileOpen` function. These constants specify the mode in which a file stream is opened.

For each file stream, you must pass to the `FileOpen` function only one of the primary mode selectors listed.

Constant Values

`fileModeReadOnly`	Open for read-only access
`fileModeReadWrite`	Open/create for read/write access, discarding any previous version of stream
`fileModeUpdate`	Open/create for read/write, preserving previous version of stream if it exists
`fileModeAppend`	Open/create for read/write, always writing to the end of the stream

Secondary Open Mode Constants

You can use the | operator (bitwise inclusive OR) to append to a primary mode selector one or more of the secondary mode selectors listed below.

`fileModeDontOverwrite`	Prevents `fileModeReadWrite` from discarding an existing stream having the same name; may only be specified together with `fileModeReadWrite`
`fileModeLeaveOpen`	Leave stream open when application quits. Most applications should not use this option.
`fileModeExclusive`	No other application can open the stream until the application that opened it in this mode closes it.
`fileModeAnyTypeCreator`	Accept any type/creator when opening or replacing an existing stream. Normally, the `FileOpen` function opens only streams having the specified creator and type. Setting this option enables the `FileOpen` function to open streams having a type or creator other than those specified.
`fileModeTemporary`	Delete the stream automatically when it is closed. For more information, see Comment section of `FileOpen` function description.

File Streaming Functions

FileClearerr

Purpose Clear I/O error status, end of file error status, and last error.

Prototype
```
Err FileClearerr (FileHand stream)
```

Parameters --> `stream` Handle to open stream.

Result 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileGetLastError`, `FileRewind`

FileClose

Purpose Close the file stream and destroy its handle. If the stream was opened with `fileModeTemporary`, it is deleted upon closing.

Prototype
```
Err FileClose (FileHand stream)
```

Parameters --> `stream` Handle to open stream.

Result 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

FileControl

Purpose Perform the operation specified by the `op` parameter on the `stream` file stream.

Prototype

```
Err FileControl (FileOpEnum op, FileHand stream,
void* valueP, Int32* valueLenP)
```

Parameters

`op`	The operation to perform, and its associated formal parameters. See the Comments section for a list of possible values.
--> `stream`	Open stream handle if required for file stream operation.
<--> `valueP`	Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the `op` parameter. For details, see the Comments section.
<--> `valueLenP`	Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the `op` parameter. For details, see the Comments section.

Result Returns either a value defined by the selector passed as the argument to the `op` parameter, or an error code resulting from the requested operation. For details, see the Comments section.

Comments Normally, you do not call the `FileControl` function yourself; it is called for you by most of the other file streaming functions and macros to perform common file streaming operations. You can call `FileControl` yourself to enable specialized read modes.

`fileOpNone`	No-op.
`fileOpDestructiveReadMode`	Enter destructive read mode, and rewind stream to its beginning. Once in this mode, there is no turning back: stream's contents after closing (or crash) are undefined.

Destructive read mode deletes blocks as data are read, thus freeing storage automatically. Once in destructive read mode, you cannot re-use the file stream—the contents of the stream are undefined after it is closed or after a crash.

Writing to files opened without write access or those that are in destructive read state is not allowed; thus, you cannot call the `FileWrite`, `FileSeek`, or `FileTruncate` functions on a stream that is in destructive read mode. One exception to this rule applies to streams that were opened in "write + append" mode and then switched into destructive read state. In this case, the `FileWrite` function can append data to the stream, but it also preserves the current stream position so that subsequent reads pick up where they left off (you can think of this as a pseudo-pipe).

ARGUMENTS:
`stream` = open stream handle
`valueP` = `NULL`
`valueLenP` = `NULL`

RETURNS:
zero on success;
`fileErr`... on error

`fileOpGetEOFStatus`

Get end-of-file status (like C runtime's `feof`) (err = `fileErrEOF`). Indicates end of file condition. Use `FileClearerr` to clear this error status.

ARGUMENTS:
`stream` = open stream handle
`valueP` = `NULL`
`valueLenP` = `NULL`

RETURNS:
zero if not end of file;
non-zero if end of file

`fileOpGetLastError`

Get error code from last operation on stream, and clear the last error code value. Doesn't change status of EOF or I/O errors —use `FileClearerr` to reset all error codes.

ARGUMENTS:
`stream` = open stream handle
`valueP` = `NULL`
`valueLenP` = `NULL`

RETURNS:
Error code from last file stream operation

`fileOpClearError`

Clear I/O and EOF error status and last error.

ARGUMENTS:
`stream` = open stream handle
`valueP` = `NULL`
`valueLenP` = `NULL`

RETURNS:
zero on success; `fileErr`... on error

`fileOpGetIOErrorStatus`

Get I/O error status (like C runtime's `ferror`). Use `FileClearerr` to clear this error status.

ARGUMENTS:
`stream` = open stream handle
`valueP` = `NULL`
`valueLenP` = `NULL`

RETURNS:
zero if not I/O error;
non-zero if I/O error is pending.

`fileOpGetCreatedStatus`

Find out whether file was created by `FileOpen` function

ARGUMENTS:
`stream` = open stream handle
`valueP` = Pointer to Boolean
`valueLenP` = Pointer to `Int32` variable set to sizeof(Boolean)

RETURNS:
zero on success; `fileErr`... on error. The Boolean variable will be set to non-zero if the file was created.

`fileOpGetOpenDbRef`

Get the open database reference (handle) of the underlying database that implements the stream (`NULL` if none); this is needed for performing Palm OS-specific operations on the underlying database, such as changing or getting creator and type, version, backup/reset bits, and so on.

ARGUMENTS:
`stream` = open stream handle
`valueP` = Pointer to `DmOpenRef` variable
`valueLenP` = Pointer to Int32 variable set to sizeof(DmOpenRef)

RETURNS:
zero on success; `fileErr`... on error. The `DmOpenRef` variable will be set to the file's open db reference that may be passed to Data Manager calls;

WARNING: Do not make any changes to the data of the underlying database -- doing so will corrupt the file stream.

`fileOpFlush`

Flush any cached data to storage.

ARGUMENTS:
`stream` = open stream handle
`valueP` = `NULL`
`valueLenP` = `NULL`

RETURNS:
zero on success; `fileErr`... on error;

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileClearerr`, `FileEOF`, `FileError`, `FileFlush`, `FileGetLastError`, `FileRewind`

FileDelete

Purpose Deletes the specified file stream from the specified card. Only a closed stream may be passed to this function.

Prototype

```
Err FileDelete (UInt16 cardNo, Char* nameP)
```

Parameters

`cardNo`	Card on which the file stream to delete resides. Currently, no Palm OS® devices support multiple cards, so this value must be 0.
`nameP`	String that is the name of the stream to delete.

Result 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileOpen`

FileDmRead

Purpose Read data from a file stream into a chunk, record, or resource residing in a database.

Prototype

```
Int32 FileDmRead (FileHand stream,
void* startOfDmChunkP, Int32 destOffset,
Int32 objSize, Int32 numObj, Err* errP)
```

Parameters

--> `stream`	Handle to open stream.
--> `startOfDmChunkP`	Pointer to beginning of chunk, record or resource residing in a database.
`destOffset`	Offset from `startOfDmChunkP` (base pointer) to the destination area (must be >= 0).
`objSize`	Size of each stream object to read.
`numObj`	Number of stream objects to read.

<--> `errP`	Pointer to variable that is to hold the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for more information.

Result The number of whole objects that were read—note that the number of objects actually read may be less than the number requested.

Comments When the number of objects actually read is less than the number requested, you may be able to determine the cause of this result by examining the return value of the `errP` parameter or by calling the `FileGetLastError` function. If the cause is insufficient data in the stream to satisfy the full request, the current stream position is at end-of-file and the "end of file" indicator is set. If a non-`NULL` pointer was passed as the value of the `errP` parameter when the `FileDmRead` function was called and an error was encountered, `*errP` holds a non-zero error code when the function returns. In addition, the `FileError` and `FileEOF` functions may be used to check for I/O errors.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileRead`, `FileError`, `FileEOF`

FileEOF

Purpose Get end-of-file status (err = `fileErrEOF` indicates end of file condition).

Prototype

```
Err FileEOF (FileHand stream)
```

Parameters

--> `stream`	Handle to open stream.

Result 0 if *not* end of file; non-zero if end of file. See the section "File Streaming Error Codes" for more information.

Comments This function's behavior is similar to that of the `feof` function provided by the C programming language runtime library.

Use `FileClearerr` to clear the I/O error status.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileClearerr`, `FileGetLastError`, `FileRewind`

FileError

Purpose Get I/O error status.

Prototype

```
Err FileError (FileHand stream)
```

Parameters --> `stream` Handle to open stream.

Result 0 if no error, and non-zero if an I/O error indicator has been set for this stream. See the section "File Streaming Error Codes" for more information.

Comments This function's behavior is similar to that of the C programming language's `ferror` runtime function.

Use `FileClearerr` to clear the I/O error status.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileClearerr`, `FileGetLastError`, `FileRewind`

FileFlush

Purpose Flush cached data to storage.

Prototype

```
Err FileFlush (FileHand stream)
```

Parameters --> `stream` Handle to open stream.

Result 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Comments It is not always necessary to call this function explicitly—certain operations flush the contents of a stream automatically; for example, streams are flushed when they are closed. Because this function's behavior is similar to that of the `fflush` function provided by the C programming language runtime library, you only need to call it explicitly under circumstances similar to those in which you would call `fflush` explicitly.

Compatibility Implemented only if 3.0 New Feature Set is present.

FileGetLastError

Purpose Get error code from last operation on file stream, and clear the last error code value (will not change end of file or I/O error status -- use `FileClearerr` to reset all error codes)

Prototype

```
Err FileGetLastError (FileHand stream)
```

Parameters

--> `stream` Handle to open stream.

Result Error code returned by the last file stream operation. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileClearerr`, `FileEOF`, `FileError`

FileOpen

Purpose Open existing file stream or create an open file stream for I/O in the mode specified by the `openMode` parameter.

Prototype

```
FileHand FileOpen (UInt16 cardNo, Char* nameP,
UInt32 type, UInt32 creator, UInt32 openMode,
Err* errP)
```

Parameters

`cardNo`	Card on which the file stream to open resides. Currently, no Palm OS devices support multiple cards, so this value must be 0.
--> `nameP`	Pointer to text string that is the name of the file stream to open or create. This value must be a valid name—no wildcards allowed, must not be `NULL`.
`type`	File type of stream to open or create. Pass `0` for wildcard, in which case `sysFileTFileStream` is used if the stream needs to be created and `fileModeTemporary` is not specified. If type is `0` and `fileModeTemporary` is specified, then `sysFileTTemp` is used for the file type of the stream this function creates.
`creator`	Creator of stream to open or create. Pass `0` for wildcard, in which case the current application's creator ID is used for the creator of the stream this function creates.
`openMode`	Mode in which to open the file stream. You must specify only one primary mode selector. Additionally, you can use the `|` operator (bitwise inclusive OR) to append one or more secondary mode selectors to the primary mode selector. See "Primary Open Mode Constants" and "Secondary Open Mode Constants" for the list of possible values.

<--> `errP`	Pointer to variable that is to hold the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for a list of error codes.

Result If successful, returns a handle to an open file stream; otherwise, returns `0`.

Comments The `fileModeReadOnly`, `fileModeReadWrite`, `fileModeUpdate`, and `fileModeAppend` modes are mutually exclusive—pass only one of them to the `FileOpen` function!

When the `fileModeTemporary` open mode is used and the file type passed to `FileOpen` is `0`, the `FileOpen` function uses `sysFileTTemp` (defined in `SystemMgr.rh`) for the file type, as recommended. In future versions of Palm OS, this configuration will enable the automatic cleanup of undeleted temporary files after a system crash. Automatic post-crash cleanup is not implemented in current versions of Palm OS.

To open a file stream even if it has a different type and creator than specified, pass the `fileModeAnyTypeCreator` selector as a flag in the `openMode` parameter to the `FileOpen` function.

The `fileModeLeaveOpen` mode is an esoteric option that most applications should not use. It may be useful for a library that needs to open a stream from the current application's context and keep it open even after the current application quits. By default, Palm OS automatically closes all databases that were opened in a particular application's context when that application quits. The `fileModeLeaveOpen` option overrides this default behavior.

Compatibility Implemented only if 3.0 New Feature Set is present.

FileRead

Purpose

Reads data from a stream into a buffer. Do not use this function to read data into a chunk, record or resource residing in a database—you must use the `FileDmRead` function for such operations.

Prototype

```
Int32 FileRead (FileHand stream, void* bufP,
Int32 objSize, Int32 numObj, Err* errP)
```

Parameters

--> `stream`	Handle to open stream.
--> `bufP`	Pointer to beginning of buffer into which data is read
`objSize`	Size of each stream object to read.
`numObj`	Number of stream objects to read.
<--> `errP`	Pointer to variable that is to hold the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for a list of error codes.

Result

The number of whole objects that were read—note that the number of objects actually read may be less than the number requested.

Comments

Do not use this function to read data into a chunk, record or resource residing in a database—you must use the `FileDmRead` function for such operations.

When the number of objects actually read is fewer than the number requested, you may be able to determine the cause of this result by examining the return value of the `errP` parameter or by calling the `FileGetLastError` function. If the cause is insufficient data in the stream to satisfy the full request, the current stream position is at end-of-file and the "end of file" indicator is set. If a non-`NULL` pointer was passed as the value of the `errP` parameter when the `FileRead` function was called and an error was encountered, `*errP` holds a non-zero error code when the function returns. In addition, the `FileError` and `FileEOF` functions may be used to check for I/O errors.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileDmRead`

FileRewind

Purpose Reset position marker to beginning of stream and clear all error codes.

Prototype

```
Err FileRewind (FileHand stream)
```

Parameters

--> `stream`	Handle to open stream.

Result 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileSeek`, `FileTell`, `FileClearerr`, `FileEOF`, `FileError`, `FileGetLastError`

FileSeek

Purpose Set current position within a file stream, extending the stream as necessary if it was opened with write access.

Prototype

```
Err FileSeek (FileHand stream, Int32 offset,
FileOriginEnum origin)
```

Parameters

--> `stream`	Handle to open stream.
`offset`	Position to set, expressed as the number of bytes from origin. This value may be positive, negative, or `0`.
`origin`	Describes the origin of the position change. Possible values are: `fileOriginBeginning` From the beginning (first data byte of file).

`fileOriginCurrent`
From the current position.

`fileOriginEnd`
From the end of file (one position beyond last data byte).

Result 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Comments Attempting to seek beyond end-of-file in a read-only stream results in an I/O error.

This function's behavior is similar to that of the `fseek` function provided by the C programming language runtime library.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileRewind`, `FileTell`

FileTell

Purpose Get current position and, optionally, file size.

Prototype

```
Int32 FileTell (FileHand stream, Int32* fileSizeP,
Err* errP)
```

Parameters

--> `stream`	Handle to open stream.
<-> `fileSizeP`	Pointer to variable that holds value describing size of stream in bytes when this function returns. Pass `NULL` to ignore.
<--> `errP`	Pointer to variable that is to hold the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for a list of possible error codes.

Result If successful, returns current position, expressed as an offset in bytes from the beginning of the stream. If an error was encountered, returns `-1` as a signed long integer.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileRewind`, `FileSeek`

FileTruncate

Purpose Truncate the file stream to a specified size; not allowed on streams open in destructive read mode or read-only mode.

Prototype

```
Err FileTruncate (FileHand stream, Int32 newSize)
```

Parameters

--> `stream`	Handle of open stream.
`newSize`	New size; must not exceed current stream size.

Result 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for a list of possible error codes.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileTell`

FileWrite

Purpose Write data to a stream.

Prototype

```
Int32 FileWrite (FileHand stream, void* dataP,
Int32 objSize, Int32 numObj, Err* errP)
```

Parameters

--> `stream`	Handle to open stream.
--> `dataP`	Pointer to buffer holding data to write.
`objSize`	Size of each stream object to write; must be ≥ `0`.
`numObj`	Number of stream objects to write.

<--> `errP`	Optional pointer to variable that holds the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for a list of possible error codes.

Result The number of whole objects that were written—note that the number of objects actually written may be less than the number requested. Should available storage be insufficient to satisfy the entire request, as much of the requested data as possible is written to the stream, which may result in the last object in the stream being incomplete.

Comments Writing to files opened without write access or those that are in destructive read state is not allowed; thus, you cannot call the `FileWrite`, `FileSeek`, or `FileTruncate` functions on a stream that is in destructive read mode. One exception to this rule applies to streams that were opened in "write + append" mode and then switched into destructive read state. In this case, the `FileWrite` function can append data to the stream, but it also preserves the current stream position so that subsequent reads pick up where they left off (you can think of this as a pseudo-pipe).

Compatibility Implemented only if 3.0 New Feature Set is present.

File Streaming Error Codes

This section lists all error codes returned by the file streaming functions.

Error Code	Value	Meaning
`fileErrMemErr`	(`fileErrorClass`|1)	Out of memory error
`fileErrInvalidParam`	(`fileErrorClass`|2)	Invalid parameter value passed
`fileErrCorruptFile`	(`fileErrorClass`|3)	Alleged stream is corrupted, invalid, or not a stream

Error Code	Value	Meaning
`fileErrNotFound`	(`fileErrorClass`|4)	Couldn't find the stream
`fileErrTypeCreatorMismatch`	(`fileErrorClass`|5)	Type and/or creator not what was specified
`fileErrReplaceError`	(`fileErrorClass`|6)	Couldn't replace existing stream
`fileErrCreateError`	(`fileErrorClass`|7)	Couldn't create new stream
`fileErrOpenError`	(`fileErrorClass`|8)	Generic open error
`fileErrInUse`	(`fileErrorClass`|9)	Stream couldn't be opened or deleted because it is in use
`fileErrReadOnly`	(`fileErrorClass`|10)	Couldn't open in write mode because existing stream is read-only
`fileErrInvalidDescriptor`	(`fileErrorClass`|11)	Invalid file descriptor (`FileHandle`)
`fileErrCloseError`	(`fileErrorClass`|12)	Error closing the stream
`fileErrOutOfBounds`	(`fileErrorClass`|13)	Attempted operation went out of bounds of the stream
`fileErrPermissionDenied`	(`fileErrorClass`|14)	Couldn't write to a stream open for read-only access
`fileErrIOError`	(`fileErrorClass`|15)	Generic I/O error

Error Code	Value	Meaning
`fileErrEOF`	(`fileErrorClass`|16)	End-of-File error
`fileErrNotStream`	(`fileErrorClass`|17)	Attempted to open an entity that is not a stream

31

Float Manager

This section provides reference material for the float manager. The float manager API is declared in the header file `FloatMgr.h`.

For more information on the float manager, see the section "Floating-Point" in the *Palm OS Programmer's Companion*.

Float Manager Functions

FplAdd

Purpose Add two floating-point numbers (returns a + b).

Prototype

```
FloatType FplAdd (FloatType a, FloatType b)
```

Parameters

`a, b`	The floating-point numbers.

Result Returns the normalized floating-point result of the addition.

Comment Under Palm OS® 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "Floating-Point" section in the *Palm OS Programmer's Companion*.

FplAToF

Purpose Convert a zero-terminated ASCII string to a floating-point number. The string must be in the format: [-]x[.]yyyyyyyy[e[-]zz]

Prototype

```
FloatType FplAToF (char* s)
```

Parameters

`s`	Pointer to the ASCII string.

Result Returns the floating-point number.

Comment The mantissa of the number is limited to 32 bits.

See Also `FplFToA`

FplBase10Info

Purpose Extract detailed information on the base 10 form of a floating-point number: the base 10 mantissa, exponent, and sign.

Prototype

```
Err FplBase10Info (FloatType a, ULong* mantissaP,
Int* exponentP, Int* signP)
```

Parameters

`a`	The floating-point number.
`mantissaP`	The base 10 mantissa (return value).
`exponentP`	The base 10 exponent (return value).
`signP`	The sign, 1 or -1 (return value).

Result Returns an error code, or 0 if no error.

Comments The mantissa is normalized so it contains at least `kMaxSignificantDigits` significant digits when printed as an integer value.

`FlpBase10Info` reports that zero is "negative"; that is, it returns a one for xSign. If this is a problem, a simple workaround is:

```
if (xMantissa == 0) {
    xSign = 0;
```

FplDiv

Purpose Divide two floating-point numbers (result = dividend/divisor).

Prototype

```
FloatType FplDiv (FloatType dividend,
FloatType divisor)
```

Parameters

`dividend`	Floating-point dividend.
`divisor`	Floating-point divisor.

Result Returns the normalized floating-point result of the division.

Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "Floating-Point" section in the *Palm OS Programmer's Companion*.

FplFloatToLong

Purpose Convert a floating-point number to a long integer.

Prototype

```
Long FplFloatToLong (FloatType f)
```

Parameters

`f`	Floating-point number to be converted.

Result Returns the long integer.

See Also `FplLongToFloat`, `FplFloatToULong`

FplFloatToULong

Purpose Convert a floating-point number to an unsigned long integer.

Prototype
```
ULong FplFloatToULong (FloatType f)
```

Parameters

`f`	Floating-point number to be converted.

Result Returns an unsigned long integer.

See Also FplLongToFloat, FplFloatToLong

FplFree

Purpose Release all memory allocated by the floating-point initialization.

Prototype
```
void FplFree()
```

Parameters None.

Result Returns nothing.

Comments Applications must call this routine after they've called other functions that are part of the float manager.

See Also FplInit

FplFToA

Purpose Convert a floating-point number to a zero-terminated ASCII string in exponential format: `[-]x.yyyyyyyye[-]zz`

Prototype
```
Err FplFToA (FloatType a, char* s)
```

Parameters

`a`	Floating-point number.

`s`	Pointer to buffer to contain the ASCII string.

Result Returns an error code, or 0 if no error.

See Also `FplAToF`

FplInit

Purpose Initialize the floating-point conversion routines.

Allocate space in the system heap for floating-point globals.

Initialize the `tenPowers` array in the globals area to the powers of 10 from -99 to +99 in floating-point format.

Prototype

```
Err FplInit()
```

Parameters None.

Result Returns an error code, or 0 if no error.

Comments Applications must call this routine before calling any other `Fpl` function.

See Also `FplFree`

FplLongToFloat

Purpose Convert a long integer to a floating-point number.

Prototype

```
FloatType FplLongToFloat (Long x)
```

`x`	A long integer.

Result Returns the floating-point number.

FplMul

Purpose Multiply two floating-point numbers.

Prototype
```
FloatType FplMul (FloatType a, FloatType b)
```

Parameters

`a, b`	The floating-point numbers.

Result Returns the normalized floating-point result of the multiplication.

Comment Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "Floating-Point" section in the *Palm OS Programmer's Companion*.

FplSub

Purpose Subtract two floating-point numbers (returns a - b).

Prototype
```
FloatType FplSub (FloatType a, FloatType b)
```

Parameters

`a, b`	The floating-point numbers.

Result Returns the normalized floating-point result of the subtraction.

Comment Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "Floating-Point" section in the *Palm OS Programmer's Companion*.

32

Fonts

This chapter provides reference material for font support. The API that this chapter describes is declared in the header files `Font.h` and `FontSelect.h`. For more information on fonts, see the "Text" section in the *Palm OS Programmer's Companion*.

Font Functions

FntAverageCharWidth

Purpose Return the average character width in the current font.

Prototype

```
Int16 FntAverageCharWidth (void)
```

Parameters None.

Result Returns the average character width (in pixels).

FntBaseLine

Purpose Return the distance from the top of character cell to the baseline for the current font.

Prototype

```
Int16 FntBaseLine (void)
```

Parameters None.

Result Returns the baseline of the font (in pixels).

FntCharHeight

Purpose Return the character height, in the current font including accents and descenders.

Prototype

```
Int16 FntCharHeight (void)
```

Parameters None

Result Height of the characters in the current font, expressed in pixels.

FntCharsInWidth

Purpose Find the length in bytes of the characters from a specified string that fit within a passed width.

Prototype

```
void FntCharsInWidth (Char const * string,
Int16 *stringWidthP, Int16 *stringLengthP,
Boolean *fitWithinWidth)
```

Parameters

`string`	Pointer to the character string.
`stringWidthP`	Maximum width to allow (in pixels).
`stringLengthP`	Maximum length of text to allow, in bytes (assumes current font).
`fitWithinWidth`	Set to `true` if string is considered truncated.

Result When the call is completed, the information is updated as follows:

`stringWidthP`	Set to the width of the characters allowed.
`stringLengthP`	Set to the length in bytes of the text that can appear within the width.
`fitWithinWidth`	`true` if the string is considered truncated, `false` if it isn't.

Comments Spaces at the end of a string are ignored and removed. Characters after a carriage return are ignored, the string is considered truncated.

FntCharsWidth

Purpose Return the width of the specified character string. The Missing Character Symbol is substituted for any character which does not exist in the current font.

Prototype

```
Int16 FntCharsWidth (Char const *chars, Int16 len)
```

Parameters

`chars`	Pointer to a string of characters.
`len`	Length in bytes of the string.

Result Returns the width of the string, in pixels.

FntCharWidth

Purpose Return the width of the specified character. If the specified character does not exist within the current font, the Missing Character Symbol is substituted.

Prototype

```
Int16 FntCharWidth (Char ch)
```

Parameters

`ch`	Character whose width is needed.

Result Returns the width of the specified character (in pixels).

Comments `FntCharWidth` works with single-byte characters only. To determine the pixel width of a single-byte character or a multi-byte character, use `TxtCharWidth` instead of this function on systems that support the Text Manager.

FntDefineFont

Purpose Makes a custom font available to your application. The custom font is available only when the application that called this function is

running; when the application quits, the custom font is uninstalled automatically.

Prototype

```
Err FntDefineFont (FontID font, FontPtr fontP)
```

Parameters

`font`	An application-defined value greater than 128 that identifies the custom font to the system. Although this value is local to the application that called the `FntDefineFont` function, it must be greater than 128 because values less than 128 are reserved for system use.
`fontP`	Pointer to the custom font resource to be used by this function. This resource must remain locked until the calling application undefines the custom font or quits.

Result

`0`	no error
`memErrNotEnoughSpace`	Insufficient dynamic heap space

Comments

The font this function specifies is not available at build time; as a result, some UI elements—labels, for example—cannot determine their bounds automatically as they do when using the built-in fonts. This mechanism and its associated tools may be augmented in the near future; for more information, stay in contact with Palm.

Compatibility

Implemented only if 3.0 New Feature Set is present.

See Also

`FontSelect`, `FntSetFont`

FntDescenderHeight

Purpose Return the height of a character's descender in the current font. The height of a descender is the distance between the base line an the bottom of the character cell.

Prototype
```
Int16 FntDescenderHeight (void)
```

Parameters None.

Result Returns the height of a descender, expressed in pixels.

FntGetFont

Purpose Return the Font ID of the current font.

Prototype
```
FontID FntGetFont (void)
```

Parameters None.

Result Returns the Font ID of the current font.

FntGetFontPtr

Purpose Return a pointer to the current font.

Prototype
```
FontPtr FntGetFontPtr (void)
```

Parameters None.

Result Returns the `FontPtr` of the current font.

FntGetScrollValues

Purpose Return the values needed to update a scroll bar based on a specified string and the position within the string.

Prototype

```
void FntGetScrollValues (Char const *chars,
UInt16 width, UInt16 scrollPos, UInt16 *linesP,
UInt16 *topLine)
```

Parameters

`chars`	Null-terminated string.
`width`	Width to word wrap at, in pixels.
`scrollPos`	Character position of the first visible character.
`linesP`	(returned) number of lines of text.
`topLine`	(returned) top visible line.

Result Returns nothing. Stores the number of lines of text in `linesP` and the top visible line in `topLine`.

Compatibility Implemented only if 2.0 New Feature Set is present.

FntLineHeight

Purpose Return the height of a line in the current font. The height of a line is the height of the character cell plus the space between lines (the external leading).

Prototype

```
Int16 FntLineHeight (void)
```

Parameters None.

Result Returns the height of a line in the current font.

FntLineWidth

Purpose Return the width of the specified line of text, taking tab characters in to account. The function assumes that the characters passed are left-

aligned and that the first character in the string is the first character drawn on a line. In other words, this routine doesn't work for characters that don't start at the beginning of a line.

Prototype

```
Int16 FntLineWidth (Char const *pChars,
UInt16 length)
```

Parameters

`pChars`	Pointer to a string of characters.
`length`	Length in bytes of the string.

Result Returns the line width (in pixels).

FntSetFont

Purpose Set the current font.

Prototype

```
FontID FntSetFont (FontID font)
```

Parameters

`font`	ID of the font to make the active font.

Result Returns the ID of the current font before the change.

FntWidthToOffset

Purpose Given a pixel position, return the offset of the character displayed at that location.

Prototype

```
Int16 FntWidthToOffset (Char const *pChars,
UInt16 length, Int16 pixelWidth,
Boolean *leadingEdge, Int16 *truncWidth)
```

Parameters

`-> pChars`	Pointer to the character string. Must not be `NULL`.
`-> length`	Length in bytes of `pChars`.
`-> pixelWidth`	A horizontal location on the screen, given in pixels.

`<- leadingEdge`	Set to `true` if the pixel position `pixelWidth` falls on the left side of the character. Pass `NULL` for this parameter if you don't need this information.
`<- truncWidth`	The width of the text (in pixels) up to the returned offset. Pass `NULL` for this parameter if you don't need this information.

Result Returns the offset into `pChars` of the character displayed at the location `pixelWidth`.

Compatibility Implemented only if 3.1 New Feature Set is present.

FntWordWrap

Purpose Given a string, determine how many bytes of text can be displayed within the specified width.

Prototype

```
UInt16 FntWordWrap (Char const *chars,
UInt16 maxWidth)
```

Parameters

`chars`	Pointer to a null-terminated string.
`maxWidth`	Maximum line width in pixels.

Result Returns the length of the line, in bytes.

Compatibility Implemented only if 2.0 New Feature Set is present.

FntWordWrapReverseNLines

Purpose Word wrap a text string backwards by the number of lines specified. The character position of the start of the first line and the number of lines that are actually word wrapped are returned.

Prototype

```
void FntWordWrapReverseNLines
(Char const *const chars, UInt16 maxWidth,
UInt16 *linesToScrollP, UInt16 *scrollPosP)
```

Parameters

`chars`	Pointer to a null-terminated string.
`maxWidth`	Maximum line width in pixels.
`linesToScrollP`	The number of lines to scroll. Upon return, contains the number lines that were scrolled.
`scrollPosP`	Byte offset of the first character. Upon return, contains the first character after wrapping.

Result Returns nothing. Stores the first character after wrapping and the number of lines scrolled in `scrollPosP` and `linesToScrollP`.

Compatibility Implemented only if 2.0 New Feature Set is present.

FontSelect

Purpose Displays a dialog box in which the user can choose one of three system-supplied fonts, and returns a FontID value representing the user's choice.

Prototype

```
FontID FontSelect (FontID fontID)
```

Parameters

`fontID` A `fontID` value specifying the font to be highlighted as the default choice in the dialog box this function displays. This value must be one of the following system-supplied constants:

`stdFont`
Standard plain text font

`boldFont`
Bold version of `stdFont`

`largeBoldFont`
Larger version of `boldFont`

Result Returns a `fontID` value representing the font that the user chose in the dialog box this function displays.

Comments When your application starts up for the first time, it should use the features `sysFtrDefaultFont` and `sysFtrDefaultBoldFont` to determine the default font for the application. For example:

```
FtrGet(sysFtrCreator, sysFtrDefaultFont,
    &fntID)
```

After this call returns, `fntID` contains an ID compatible with the `FontSelect` function.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FntGetFont`, `FntSetFont`

33

Graffiti Manager

This chapter provides reference material for the Graffiti® manager. The Graffiti manager API is declared in the header file `Graffiti.h`.

For more information on the Graffiti manager, see the section "Receiving User Input" in the *Palm OS Programmer's Companion*.

Graffiti Manager Functions

GrfAddMacro

Purpose Add a macro to the macro list.

Prototype

```
Err GrfAddMacro (Char* nameP, UInt8* macroDataP,
UInt16 dataLen)
```

Parameters

`nameP`	Name of macro.
`macroDataP`	Data of macro.
`dataLen`	Size of macro data in bytes.

Result Returns 0 if no error; returns `grfErrNoMacros`, `grfErrMacroPtrTooSmall`, `dmErrNotValidRecord`, `dmErrWriteOutOfBounds` if an error occurs.

See Also `GrfGetMacro`, `GrfGetMacroName`, `GrfDeleteMacro`

GrfAddPoint

Purpose Add a point to the Graffiti point buffer.

Prototype
```
Err GrfAddPoint (PointType* pt)
```

Parameters

`pt`	Pointer to point buffer.

Result Returns 0 if no error; returns `grfErrPointBufferFull` if an error occurs.

See Also `GrfFlushPoints`

GrfCleanState

Purpose Remove any temporary shifts from the dictionary state.

Prototype
```
Err GrfCleanState (void)
```

Parameters None

Result Returns 0 if no error, or `grfErrNoDictionary` if an error occurs.

See Also `GrfInitState`

GrfDeleteMacro

Purpose Delete a macro from the macro list.

Prototype
```
Err GrfDeleteMacro (UInt16 index)
```

Parameters

`index`	Index of the macro to delete.

Result Returns 0 if no error, or `grfErrNoMacros`, `grfErrMacroNotFound` if an error occurs.

See Also `GrfAddMacro`

GrfFilterPoints

Purpose Filter the points in the Graffiti point buffer.

Prototype
```
Err GrfFilterPoints (void)
```

Parameters None.

Result Always returns 0.

See Also GrfMatch

GrfFindBranch

Purpose Locate a branch in the Graffiti dictionary by flags.

Prototype
```
Err GrfFindBranch (UInt16 flags)
```

Parameters

`flags`	Flags of the branch you're searching for.

Result Returns 0 if no error, or `grfErrNoDictionary` or `grfErrBranchNotFound` if an error occurs.

See Also GrfCleanState, GrfInitState

GrfFlushPoints

Purpose Dispose of all points in the Graffiti point buffer.

Prototype
```
Err GrfFlushPoints (void)
```

Parameters None.

Result Always returns 0.

See Also GrfAddPoint

GrfGetAndExpandMacro

Purpose Look up and expand a macro in the current macros.

Prototype

```
Err GrfGetAndExpandMacro (Char* nameP,
UInt8* macroDataP, UInt16* dataLenP)
```

Parameters

`nameP`	Name of macro to look up.
`macroDataP`	Macro contents returned here.
`dataLenP`	On entry, size of `macroDataP` buffer; on exit, number of bytes in macro data.

Result Returns 0 if no error, or `grfErrNoMacros` or `grfErrMacroNotFound` if an error occurs.

See Also `GrfAddMacro`, `GrfGetMacro`

GrfGetGlyphMapping

Purpose Look up a glyph in the dictionary and return the text.

Prototype

```
Err GrfGetGlyphMapping (UInt16 glyphID,
UInt16* flagsP, void* dataPtrP, UInt16* dataLenP,
UInt16* uncertainLenP)
```

Parameters

`glyphID`	Glyph ID to look up.
`flagsP`	Returned dictionary flags.
`dataPtrP`	Where returned text goes.
`dataLenP`	On entry, size of `dataPtrP`; on exit, number of bytes returned.
`uncertainLenP`	Return number of uncertain characters in text.

Result Returns 0 if no error, or `grfErrNoDictionary` or `grfErrNoMapping` if an error occurs.

See Also `GrfMatch`

GrfGetMacro

Purpose Look up a macro in the current macros.

Prototype

```
Err GrfGetMacro (Char* nameP, UInt8* macroDataP,
UInt16* dataLenP)
```

Parameters

`nameP`	Name of macro to lookup.
`macroDataP`	Macro contents returned here.
`dataLenP`	On entry: size of `macroDataP` buffer. On exit: number of bytes in macro data.

Result Returns 0 if no error or `grfErrNoMacros`, `grfErrMacroNotFound`.

See Also `GrfAddMacro`

GrfGetMacroName

Purpose Look up a macro name by index.

Prototype

```
Err GrfGetMacroName (UInt16 index, Char* nameP)
```

Parameters

`index`	Index of macro.
`nameP`	Name returned here.

Result Returns 0 if no error, or `grfErrNoMacros` or `grfErrMacroNotFound` if an error occurs.

See Also `GrfAddMacro`, `GrfGetMacro`

GrfGetNumPoints

Purpose Return the number of points in the point buffer.

Prototype
```
Err GrfGetNumPoints (UInt16* numPtsP)
```

Parameters

`numPtsP`	Returned number of points.

Result Always returns 0.

See Also `GrfAddPoint`

GrfGetPoint

Purpose Return a point out of the Graffiti point buffer.

Prototype
```
Err GrfGetPoint (UInt16 index, PointType* pointP)
```

Parameters

`index`	Index of the point to get.
`pointP`	Returned point.

Result Returns 0 if no error, or `grfErrBadParam` if an error occurs.

See Also `GrfAddPoint`, `GrfGetNumPoints`

GrfGetState

Purpose Return the current Graffiti shift state.

Prototype
```
Err GrfGetState (Boolean* capsLockP,
Boolean* numLockP, UInt16* tempShiftP,
Boolean* autoShiftedP)
```

Parameters

`capsLockP`	Returns `true` if caps lock on.
`numLockP`	Returns `true` if num lock on.
`tempShiftP`	Current temporary shift.

`autoShiftedP`	Returns `TRUE` if shift not set by the user but by the system, for example, at the beginning of a line.

Result Always returns 0.

Compatibility Note Palm OS® 2.0 and later has more user-friendly auto shifting. It uses an upper case letter under these conditions:

- after an empty field
- after a period or other sentence terminator (such as ? or !).
- after two spaces

See Also `GrfSetState`

GrfInitState

Purpose Reinitialize the Graffiti dictionary state.

Prototype

```
Err GrfInitState (void)
```

Parameters None.

Result Always returns 0.

See Also `GrfGetState`, `GrfSetState`

GrfMatch

Purpose Recognize the current stroke in the Graffiti point buffer and return with the recognized text.

Prototype

```
Err GrfMatch (UInt16* flagsP, void* dataPtrP,
UInt16* dataLenP, UInt16* uncertainLenP,
GrfMatchInfoPtr matchInfoP)
```

Parameters

`flagsP`	Glyph flags are returned here.

`dataPtrP`	Return text is placed here.
`dataLenP`	Size of `dataPtrP` on exit; number of characters returned on exit.
`uncertainLenP`	Return number of uncertain characters.
`matchInfoP`	Array of `grfMaxMatches`, or nil.

Result Returns 0 if no error, or `grfErrNoGlyphTable`, `grfErrNoDictionary`, or `grfErrNoMapping` if an error occurs.

See Also `GrfAddPoint`, `GrfFlushPoints`

GrfMatchGlyph

Purpose Recognize the current stroke as a glyph.

Prototype

```
Err GrfMatchGlyph (GrfMatchInfoPtr matchInfoP,
Int16 maxUnCertainty, UInt16 maxMatches)
```

Parameters

`matchInfoP`	Pointer to array of matches to fill in.
`maxUnCertainty`	Maximum number of errors to tolerate.
`maxMatches`	Size of `matchInfoP` array.

Result Returns 0 if no error, or `grfErrNoGlyphTable` if an error occurs.

See Also `GrfMatch`

GrfProcessStroke

Purpose Translate a stroke to keyboard events using Graffiti.

Prototype

```
Err GrfProcessStroke (PointType* startPtP,
PointType* endPtP, Boolean upShift)
```

Parameters

`startPtP`	Start point of stroke.
`endPtP`	End point of stroke.

`upShift`	Set to `true` to feed an artificial upshift into the engine.

Result Returns 0 if recognized.

Comments Called by `SysHandleEvent` when a `penUpEvent` is detected in the writing area. This routine recognizes the stroke and sends the recognized characters into the key queue. It also flushes the stroke out of the pen queue after recognition.

See Also `SysHandleEvent`

GrfSetState

Purpose Set the current shift state of Graffiti.

Prototype

```
Err GrfSetState (Boolean capsLock,
Boolean numLock, Boolean upperShift)
```

Parameters

`capsLock`	Set to `true` to turn on caps lock.
`numLock`	Set to `true` to turn on num lock.
`upperShift`	Set to `true` to put into upper shift.

Result Always returns 0.

See Also `GrfGetState`

34

Key Manager

This chapter provides reference material for the key manager. The key manager API is declared in the header file `KeyMgr.h`.

For more information on the key manager, see the section "Receiving User Input" in the *Palm OS Programmer's Companion*.

Key Manager Functions

KeyCurrentState

Purpose Return bit field with bits set for each key that is currently depressed.

Prototype

```
UInt32 KeyCurrentState (void)
```

Parameters None.

Result Returns a `UInt32` with bits set for keys that are depressed. See `keyBitPower`, `keyBitPageUp`, `keyBitPageDown`, etc., in `KeyMgr.h`.

Comments Called by applications that need to poll the keys.

See Also `KeyRates`

KeyRates

Purpose Get or set the key repeat rates.

Prototype

```
Err KeyRates (Boolean set, UInt16* initDelayP,
UInt16* periodP, UInt16* doubleTapDelayP,
Boolean* queueAheadP)
```

Parameters

`set`	If `true`, settings are changed; if `false`, current settings are returned.
`initDelayP`	Initial delay in ticks for a auto-repeat event.
`periodP`	Auto-repeat rate specified as period in ticks.
`doubleTapDelayP`	Maximum double-tap delay, in ticks.
`queueAheadP`	If `true`, auto-repeating keeps queueing up key events if the queue has keys in it. If `false`, auto-repeat doesn't enqueue keys unless the queue is already empty.

Result Returns 0 if no error.

See Also `KeyCurrentState`

KeySetMask

Purpose Specify which keys generate `keyDownEvents`.

You can specify this either by using this function or by using the `poweredOnKeyMask` modifier.

Prototype

```
UInt32 KeySetMask (UInt32 keyMask)
```

Parameters

`keyMask`	Mask with bits set for those keys to generate `keyDownEvents` for.

Result Returns the old `key Mask`.

Compatibility Implemented only if 2.0 New Feature Set is present.

35

Memory Manager

This chapter provides reference information for the memory manager. The memory manager API is declared in the header file `MemoryMgr.h`.

For more information on the memory manager, see the chapter "Memory" in the *Palm OS Programmer's Companion*.

Memory Manager Functions

MemCardInfo

Purpose Return information about a memory card.

Prototype

```
Err MemCardInfo (UInt16 cardNo, Char* cardNameP,
Char* manufNameP, UInt16* versionP,
UInt32* crDateP, UInt32* romSizeP,
UInt32* ramSizeP, UInt32* freeBytesP)
```

Parameters

`cardNo`	Card number.
`cardNameP`	Pointer to character array (32 bytes), or 0.
`manufNameP`	Pointer to character array (32 bytes), or 0.
`versionP`	Pointer to version variable, or 0.
`crDateP`	Pointer to creation date variable, or 0.
`romSizeP`	Pointer to ROM size variable, or 0.
`ramSizeP`	Pointer to RAM size variable, or 0.
`freeBytesP`	Pointer to free byte-count variable, or 0.

Result Returns 0 if no error.

Comments Pass 0 for those variables that you don't want returned.

MemCmp

Purpose Compare two blocks of memory.

NOTE: Blocks are compared as unsigned bytes.

Prototype

```
Int16 MemCmp (const void* s1, const void* s2,
Int32 numBytes)
```

Parameters

`s1, s2`	Pointers to block of memory.
`numBytes`	Number of bytes to compare.

Result Zero if they match, non-zero if not:

+ if `s1` > `s2`

- if `s1` < `s2`

Compatibility Implemented only if 2.0 New Feature Set is present.

`MemCmp` can be used to test the equality of blocks in memory on all versions that support `MemCmp`; however, testing the sort ordering of blocks in memory works reliably only on Palm OS® versions 3.5 and higher. On versions earlier than 3.2, `MemCmp` always returns a positive value if the blocks are unequal. On versions 3.2 and 3.3, `MemCmp` reliably returns positive to indicate `s1` > `s2` and negative to indicate `s1` < `s2` **only** if the characters that differ are less than 128 apart. If the difference is greater than that, `MemCmp` may return positive when it should return negative and vice versa.

MemDebugMode

Purpose Return the current debugging mode of the memory manager.

Prototype
```
UInt16 MemDebugMode(void)
```

Parameters No parameters.

Result Returns debug flags as described for MemSetDebugMode.

MemHandleCardNo

Purpose Return the card number a chunk resides in.

Prototype
```
UInt16 MemHandleCardNo (MemHandle h)
```

Parameters

-> `h`	Chunk handle.

Result Returns the card number.

Comments Call this routine to retrieve the card number (0 or 1) a movable chunk resides on.

See Also `MemPtrCardNo`

MemHandleDataStorage

Purpose Return `true` if the given handle is part of a data storage heap. If not, it's a handle in the dynamic heap.

Prototype
```
Boolean MemHandleDataStorage (MemHandle h)
```

Parameters

-> `h`	Chunk handle.

Result Returns `true` if the handle is part of a data storage heap.

Comments Called by Fields package routines to determine if they need to worry about data storage write-protection when editing a text field.

See Also `MemPtrDataStorage`

MemHandleFree

Purpose Dispose of a movable chunk.

Prototype

```
Err MemHandleFree (MemHandle h)
```

Parameters -> `h` Chunk handle.

Result Returns 0 if no error, or memErrInvalidParam if an error occurs.

Comments Call this routine to dispose of a movable chunk.

See Also `MemHandleNew`

MemHandleHeapID

Purpose Return the heap ID of a chunk.

Prototype

```
UInt16 MemHandleHeapID (MemHandle h)
```

Parameters -> `h` Chunk handle.

Result Returns the heap ID of a chunk.

Comments Call this routine to get the heap ID of the heap a chunk resides in.

See Also `MemPtrHeapID`

MemHandleLock

Purpose Lock a chunk and obtain a pointer to the chunk's data.

Prototype
```
MemPtr MemHandleLock (MemHandle h)
```

Parameters

-> `h`	Chunk handle.

Result Returns a pointer to the chunk.

Comments Call this routine to lock a chunk and obtain a pointer to the chunk.

`MemHandleLock` and `MemHandleUnlock` should be used in pairs.

See Also `MemHandleNew`, `MemHandleUnlock`

MemHandleNew

Purpose Allocate a new movable chunk in the dynamic heap and returns a handle to it.

Prototype
```
MemHandle MemHandleNew (UInt32 size)
```

Parameters

-> `size`	The desired size of the chunk.

Result Returns a handle to the new chunk, or 0 if unsuccessful.

Comments Use this call to allocate dynamic memory. Before you can write data to the memory chunk that `MemHandleNew` allocates, you must call `MemHandleLock` to lock the chunk and get a pointer to it.

See Also `MemPtrFree`, `MemPtrNew`, `MemHandleFree`, `MemHandleLock`

MemHandleResize

Purpose Resize a chunk.

Prototype

```
Err MemHandleResize (MemHandle h, UInt32 newSize)
```

Parameters

-> `h`	Chunk handle.
-> `newSize`	The new desired size.

Result

0	No error.
memErrInvalidParam	Invalid parameter passed.
memErrNotEnoughSpace	Not enough free space in heap to grow chunk.
memErrChunkLocked	Can't grow chunk because it's locked.

Comments Call this routine to resize a chunk. This routine is always successful when shrinking the size of a chunk, even if the chunk is locked. When growing a chunk, it first attempts to grab free space immediately following the chunk so that the chunk does not have to move. If the chunk has to move to another free area of the heap to grow, it must be movable and have a lock count of 0.

On devices running version 2.0 or earlier of Palm OS, the `MemHandleResize` function tries to resize the chunk only within the same heap, whereas `DmResizeRecord` will look for space in other data heaps if it can't find enough space in the original heap.

See Also `MemHandleNew`, `MemHandleSize`

MemHandleSetOwner

Purpose Set the owner ID of a chunk.

Prototype

```
Err MemHandleSetOwner (MemHandle h, UInt16 owner)
```

Parameters

-> `h`	Chunk handle.
-> `owner`	New owner ID of the chunk. Specify 0 to set the owner to the operating system.

Result Returns 0 if no error, or memErrInvalidParam if an error occurs.

Comments When you allocate a parameter block to pass to `SysUIAppSwitch` or `SysAppLaunch`, you must call `MemPtrSetOwner` to grant ownership of the parameter block chunk to the OS (your application is originally set as the owner). If the parameter block structure references any chunks by handle, you also must call `MemHandleSetOwner` to grant ownership of those blocks to the OS. If you don't change the ownership of these chunks, they will get freed before the application you're launching has a chance to use them.

MemHandleSize

Purpose Return the requested size of a chunk.

Prototype

```
UInt32 MemHandleSize (MemHandle h)
```

Parameters

-> `h`	Chunk handle.

Result Returns the requested size of the chunk.

Comments Call this routine to get the size originally requested for a chunk.

See Also `MemHandleResize`

MemHandleToLocalID

Purpose Convert a handle into a local chunk ID which is card relative.

Prototype
```
LocalID MemHandleToLocalID (MemHandle h)
```

Parameters

-> `h`	Chunk handle.

Result Returns local ID, or NULL (0) if unsuccessful.

Comments Call this routine to convert a chunk handle to a local ID.

See Also `MemLocalIDToGlobal`, `MemLocalIDToLockedPtr`

MemHandleUnlock

Purpose Unlock a chunk given a chunk handle.

Prototype
```
Err MemHandleUnlock (MemHandle h)
```

Parameters

-> `h`	The chunk handle.

Result

0	No error.
memErrInvalidParam	Invalid parameter passed.

Comments Call this routine to decrement the lock count for a chunk.

`MemHandleLock` and `MemHandleUnlock` should be used in pairs.

See Also `MemHandleLock`

MemHeapCheck

Purpose Check validity of a given heap.

Prototype
```
Err MemHeapCheck (UInt16 heapID)
```

Parameters

`heapID`	ID of heap to check.

Result Returns 0 if no error.

See Also `MemDebugMode`, `MemSetDebugMode`

MemHeapCompact

Purpose Compact a heap.

Prototype
```
Err MemHeapCompact (UInt16 heapID)
```

Parameters

-> `heapID`	ID of the heap to compact.

Result Always returns 0.

Comments Most applications never need to call this function explicitly. The system software calls this function at various times; for example, during memory allocation (if sufficient free space is not available) and during system reboot.

Call this routine to compact a heap and merge all free space. This routine attempts to move all movable chunks to the start of the heap and merge all free space in the center of the heap.

MemHeapDynamic

Purpose Return `true` if the given heap is a dynamic heap.

Prototype

```
Boolean MemHeapDynamic (UInt16 heapID)
```

Parameters

`heapID`	ID of the heap to be tested.

Result Returns `true` if dynamic, `false` if not.

Comments Dynamic heaps are used for volatile storage, application stacks, globals, and dynamically allocated memory.

> **NOTE:** In Palm OS 3.5, the dynamic heap is sized based on the amount of memory available, and is generally larger than before.

See Also `MemNumHeaps`, `MemHeapID`

MemHeapFlags

Purpose Return the heap flags for a heap.

Prototype

```
UInt16 MemHeapFlags (UInt16 heapID)
```

Parameters

-> `heapID`	ID of heap.

Result Returns the heap flags.

Comments Call this routine to retrieve the heap flags for a heap. The flags can be examined to determine if the heap is ROM based or not. ROM-based heaps have the memHeapFlagReadOnly bit set.

See Also `MemNumHeaps`, `MemHeapID`

MemHeapFreeBytes

Purpose Return the total number of free bytes in a heap and the size of the largest free chunk in the heap.

Prototype

```
Err MemHeapFreeBytes (UInt16 heapID,
UInt32* freeP, UInt32* maxP)
```

Parameters

-> `heapID`	ID of heap.
<-> `freeP`	Pointer to a variable of type `UInt32` for free bytes.
<-> `maxP`	Pointer to a variable of type `UInt32` for max free chunk size.

Result Always returns 0.

Comments This routine doesn't compact the heap but may be used to determine in advance whether an allocation request will succeed. Before allocating memory, call this function and test the return value of `maxP` to determine whether enough free space to fulfill your allocation request exists. If not, you may make more space available by calling the `MemHeapCompact` function. An alternative approach is to just call the `MemHeapCompact` function as necessary when an error is returned by the `MemPtrNew` or `MemHandleNew` functions.

See Also `MemHeapSize`, `MemHeapID`, `MemHeapCompact`

MemHeapID

Purpose Return the heap ID for a heap, given its index and the card number.

Prototype

```
UInt16 MemHeapID (UInt16 cardNo, UInt16 heapIndex)
```

Parameters

-> `cardNo`	The card number, either 0 or 1.

-> `heapIndex` The heap index, anywhere from 0 to `MemNumHeaps` - 1.

Result Returns the heap ID.

Comments Call this routine to retrieve the heap ID of a heap, given the heap index and the card number. A heap ID must be used to obtain information on a heap such as its size, free bytes, etc., and is also passed to any routines which manipulate heaps.

See Also `MemNumHeaps`

MemHeapScramble

Purpose Scramble the specified heap.

Prototype

```
Err MemHeapScramble (UInt16 heapID)
```

Parameters `heapID` ID of heap to scramble.

Comments The system attempts to move each movable chunk.

Useful for debugging.

Result Always returns 0.

See Also `MemDebugMode`, `MemSetDebugMode`

MemHeapSize

Purpose Return the total size of a heap including the heap header.

Prototype
```
UInt32 MemHeapSize (UInt16 heapID)
```

Parameters -> `heapID` ID of heap.

Result Returns the total size of the heap.

See Also `MemHeapFreeBytes`, `MemHeapID`

MemLocalIDKind

Purpose Return whether or not a local ID references a handle or a pointer.

Prototype
```
LocalIDKind MemLocalIDKind (LocalID local)
```

Parameters -> `local` Local ID to query

Result Returns LocalIDKind, or a memIDHandle or memIDPtr (see MemoryMgr.h).

Comments This routine determines if the given local ID is to a nonmovable (memIDPtr) or movable (memIDHandle) chunk.

MemLocalIDToGlobal

Purpose Convert a local ID, which is card relative, into a global pointer in the designated card.

Prototype
```
MemPtr MemLocalIDToGlobal (LocalID local,
UInt16 cardNo)
```

Parameters -> `local` The local ID to convert.

-> `cardNo`	Memory card the chunk resides in.

Result Returns pointer or handle to chunk.

See Also `MemLocalIDKind`, `MemLocalIDToLockedPtr`

MemLocalIDToLockedPtr

Purpose Return a pointer to a chunk given its local ID and card number.

If the local ID references a movable chunk handle, this routine automatically locks the chunk before returning.

Prototype

```
MemPtr MemLocalIDToLockedPtr (LocalID local,
UInt16 cardNo)
```

Parameters

`local`	Local chunk ID.
`cardNo`	Card number.

Result Returns pointer to chunk, or 0 if an error occurs.

See Also `MemLocalIDToGlobal`, `MemLocalIDToPtr`, `MemLocalIDKind`, `MemPtrToLocalID`, `MemHandleToLocalID`

MemLocalIDToPtr

Purpose Return pointer to chunk, given the local ID and card number.

Prototype

```
MemPtr MemLocalIDToPtr (LocalID local,
UInt16 cardNo)
```

Parameters

-> `local`	Local ID to query.
-> `cardNo`	Card number the chunk resides in.

Result Returns a pointer to the chunk, or 0 if error.

Comments If the local ID references a movable chunk and that chunk is **not** locked, this function returns 0 to indicate an error.

See Also MemLocalIDToGlobal, MemLocalIDToLockedPtr

MemMove

Purpose Move a range of memory to another rangein the dynamic heap.

Prototype

```
Err MemMove (void* dstP, const void* sP,
Int32 numBytes)
```

Parameters

dstP	Pointer to destination.
sP	Pointer to source.
numBytes	Number of bytes to move.

Result Always returns 0.

Comments Handles overlapping ranges.

For operations where the destination is in a data heap, see DmSet, DmWrite, and related functions.

MemNumCards

Purpose Return the number of memory card slots in the system. Not all slots need to be populated.

Prototype

```
UInt16 MemNumCards (void)
```

Parameters None.

Result Returns number of slots in the system.

MemNumHeaps

Purpose Return the number of heaps available on a particular card.

Prototype
```
UInt16 MemNumHeaps (UInt16 cardNo)
```

Parameters -> `cardNo` The card number; either 0 or 1.

Result Number of heaps available, including ROM- and RAM-based heaps.

Comments Call this routine to retrieve the total number of heaps on a memory card. The information can be obtained by calling MemHeapSize, MemHeapFreeBytes, and MemHeapFlags on each heap using its heap ID. The heap ID is obtained by calling MemHeapID with the card number and the heap index, which can be any value from 0 to `MemNumHeaps`.

MemNumRAMHeaps

Purpose Return the number of RAM heaps in the given card.

Prototype
```
UInt16 MemNumRAMHeaps (UInt16 cardNo)
```

Parameters `cardNo` The card number.

Result Returns the number of RAM heaps.

See Also `MemNumCards`

MemPtrCardNo

Purpose Return the card number (0 or 1) a nonmovable chunk resides on.

Prototype
```
UInt16 MemPtrCardNo (MemPtr p)
```

Parameters

-> `p`	Pointer to the chunk.

Result Returns the card number.

See Also `MemHandleCardNo`

MemPtrDataStorage

Purpose Return `true` if the given pointer is part of a data storage heap; if not, it is a pointer in the dynamic heap.

Prototype
```
Boolean MemPtrDataStorage (MemPtr p)
```

Parameters

`p`	Pointer to a chunk.

Result Returns `true` if the chunk is part of a data storage heap.

Comments Called by Fields package to determine if it needs to worry about data storage write-protection when editing a text field.

See Also `MemHeapDynamic`

MemPtrFree

Purpose Macro to dispose of a chunk.

Prototype
```
Err MemPtrFree (MemPtr p)
```

Parameters

-> `p`	Pointer to a chunk.

Result

0	If no error or `memErrInvalidParam` (invalid parameter).

Comments Call this routine to dispose of a nonmovable chunk.

MemPtrHeapID

Purpose Return the heap ID of a chunk.

Prototype
```
UInt16 MemPtrHeapID (MemPtr p)
```

Parameters

-> `p`	Pointer to the chunk.

Result Returns the heap ID of a chunk.

Comments Call this routine to get the heap ID of the heap a chunk resides in.

MemPtrNew

Purpose Allocate a new nonmovable chunk in the dynamic heap.

Prototype
```
MemPtr MemPtrNew (UInt32 size)
```

Parameters

-> `size`	The desired size of the chunk.

Result Returns pointer to the new chunk, or 0 if unsuccessful.

Comments This routine allocates a nonmovable chunk in the dynamic heap and returns a pointer to the chunk. Applications can use it when allocating dynamic memory.

In Palm OS 3.5, the dynamic heap is sized based on the amount of memory available, and is generally larger than before.

NOTE: You cannot allocate a zero-size reference block.

MemPtrRecoverHandle

Purpose Recover the handle of a movable chunk, given a pointer to its data.

Prototype

```
MemHandle MemPtrRecoverHandle (MemPtr p)
```

Parameters

-> `p`	Pointer to the chunk.

Result Returns the handle of the chunk, or 0 if unsuccessful.

Comments Don't call this function for pointers in ROM or nonmovable data chunks.

MemPtrResize

Purpose Resize a chunk.

Prototype

```
Err MemPtrResize (MemPtr p, UInt32 newSize)
```

Parameters

-> `p`	Pointer to the chunk.
-> `newSize`	The new desired size.

Result Returns 0 if no error, or memErrNotEnoughSpace memErrInvalidParam, or memErrChunkLocked if an error occurs.

Comments Call this routine to resize a locked chunk. This routine is always successful when shrinking the size of a chunk. When growing a

chunk, it attempts to use free space immediately following the chunk.

See Also MemPtrSize, MemHandleResize

MemPtrSetOwner

Purpose Set the owner ID of a chunk.

Prototype

```
Err MemPtrSetOwner (MemPtr p, UInt16 owner)
```

Parameters

-> `p`	Pointer to the chunk.
-> `owner`	New owner ID of the chunk. Specify 0 to set the owner to the operating system.

Result Returns 0 if no error, or memErrInvalidParam if an error occurs.

Comments When you allocate a parameter block to pass to SysUIAppSwitch or SysAppLaunch, you must call `MemPtrSetOwner` or MemHandleSetOwner to grant ownership of the parameter block chunk, and any other chunks referenced in it, to the OS (your application is originally set as the owner). If you don't change the ownership of the parameter block, it will get freed before the application you're launching has a chance to use it.

MemPtrSize

Purpose Return the size of a chunk.

Prototype

```
UInt32 MemPtrSize (MemPtr p)
```

Parameters

-> `p`	Pointer to the chunk.

Result The requested size of the chunk.

Comments Call this routine to get the original requested size of a chunk.

MemPtrToLocalID

Purpose Convert a pointer into a card-relative local chunk ID.

Prototype
```
LocalID MemPtrToLocalID (MemPtr p)
```

Parameters

-> `p`	Pointer to a chunk.

Result Returns the local ID of the chunk.

Comments Call this routine to convert a chunk pointer to a local ID.

See Also `MemLocalIDToPtr`

MemPtrUnlock

Purpose Unlock a chunk, given a pointer to the chunk.

Prototype
```
Err MemPtrUnlock (MemPtr p)
```

Parameters

`p`	Pointer to a chunk.

Result 0 if no error, or memErrInvalidParam if an error occurs.

Comments A chunk must **not** be unlocked more times than it was locked.

See Also `MemHandleLock`

MemSet

Purpose Set a memory range in a dynamic heap to a specific value.

Prototype
```
Err MemSet (void* dstP, Int32 numBytes,
UInt8 value)
```

Parameters

`dstP`	Pointer to the destination.
`numBytes`	Number of bytes to set.

`value`	Value to set.

Result Always returns 0.

Comments For operations where the destination is in a data heap, see DmSet, DmWrite, and related functions.

MemSetDebugMode

Purpose Set the debugging mode of the memory manager.

Prototype

```
Err MemSetDebugMode (UInt16 flags)
```

Parameters

`flags`	Debug flags.

Comments Use the logical OR operator (|) to provide any combination of one, more, or none of the following flags:

```
memDebugModeCheckOnChange
memDebugModeCheckOnAll
memDebugModeScrambleOnChange
memDebugModeScrambleOnAll
memDebugModeFillFree
memDebugModeAllHeaps
memDebugModeRecordMinDynHeapFree
```

Result Returns 0 if no error, or -1 if an error occurs.

MemStoreInfo

Purpose Return information on either the RAM store or the ROM store for a memory card.

Prototype

```
Err MemStoreInfo (UInt16 cardNo,
UInt16 storeNumber, UInt16* versionP,
UInt16* flagsP, Char* nameP, UInt32* crDateP,
UInt32* bckUpDateP, UInt32* heapListOffsetP,
UInt32* initCodeOffset1P,
UInt32* initCodeOffset2P, LocalID* databaseDirIDP)
```

Parameters

-> `cardNo` Card number, either 0 or 1.

-> `storeNumber` Store number; 0 for ROM, 1 for RAM.

<-> `versionP` Pointer to version variable, or 0.

<-> `flagsP` Pointer to flags variable, or 0.

<-> `nameP` Pointer to character array (32 bytes), or 0.

<-> `crDateP` Pointer to creation date variable, or 0.

<-> `bckUpDateP` Pointer to backup date variable, or 0.

<-> `heapListOffsetP`
Pointer to `heapListOffset` variable, or 0.

<-> `initCodeOffset1P`
Pointer to `initCodeOffset1` variable, or 0.

<-> `initCodeOffset2P`
Pointer to initCodeOffset2 variable, or 0.

<-> `databaseDirIDP`
Pointer to database directory chunk ID variable, or 0.

Result Returns 0 if no error, or memErrCardNotPresent, memErrRAMOnlyCard, or memErrInvalidStoreHeader if an error occurs.

Comments Call this routine to retrieve any or all information on either the RAM store or the ROM store for a card. Pass 0 for variables that you don't wish returned.

36

Notification Manager

This chapter provides information about the notification manager by discussing these topics:

- Notification Data Structures
- Notification Constants
- Notification Functions
- Application-Defined Functions

The header file `NotifyMgr.h` declares the API that this chapter describes. For more information on the notification manager, see the section "Notifications" on page 192 in the "Palm System Features" chapter of the *Palm OS Programmer's Companion*.

Notification Data Structures

SleepEventParamType

The `SleepEventParamType` struct is used in notifications of type `sysNotifySleepRequestEvent` to indicate why the system is going to sleep and whether sleep should be deferred.

```
typedef struct {
  UInt16 reason;
  UInt16 deferSleep;
} SleepEventParamType;
```

Field Descriptions

`reason` The reason the system is going to sleep. The possible values are:

`sysSleepAutoOff` The idle time limit has been reached.

`sysSleepPowerButton`
: The user pressed the power off button.

`sysSleepResumed`
: The sleep event was deferred by one of the notification handlers but has been resumed through the use of the `resumeSleepChr`.

`sysSleepUnknown`
: Unknown reason.

`deferSleep` Initially set to 0. If a notification handler wants to defer sleep, then it should increment this value. When `deferSleep` is greater than 0, the system waits before going to sleep.

SysNotifyDisplayChangeDetailsType

The `SysNotifyDisplayChangeDetailsType` struct is used in notifications of type `sysNotifyDisplayChangeEvent` to indicate how the bit depth changed. If the two values in the struct are equal, it means that the color palette has changed instead of the bit depth.

```
typedef struct {
  UInt32 oldDepth;
  UInt32 newDepth;
} SysNotifyDisplayChangeDetailsType;
```

Field Descriptions

`oldDepth` The old bit depth.

`newDepth` The new bit depth.

SysNotifyParamType

The `SysNotifyParamType` struct defines a notification event. This struct is sent along with the `sysAppLaunchCmdNotify` launch

code or is passed as a parameter to the notification callback function.

```
typedef struct SysNotifyParamType {
  UInt32  notifyType;
  UInt32  broadcaster;
  void *  notifyDetailsP;
  void *  userDataP;
  Boolean handled;
  UInt8   reserved2;
} SysNotifyParamType;
```

Field Descriptions

`notifyType`	The type of event that occurred. See Notification Manager Event Constants.
`broadcaster`	The creator ID of the application that broadcast the notification (the application that called `SysNotifyBroadcast` or `SysNotifyBroadcastDeferred`), or `sysNotifyBroadcasterCode` if the system broadcast the event.
`notifyDetailsP`	Pointer to data specific to this notification. Most notifications do not use this parameter. See Notification Manager Event Constants for the specific instances where this parameter is used.
`userDataP`	Pointer to custom data that your notification handler requires. You create this data and pass its pointer to `SysNotifyRegister`.
`handled`	`true` if event is handled yet; `false` otherwise. In some cases, notification handlers can set this field to `true` to indicate that they have handled an event. See Notification Manager Event Constants.
`reserved2`	Reserved for future use.

Notification Constants

Notification Manager Event Constants

The constants in the table below identify events for which the system posts notifications. In general, notifications regarding the creation of information are broadcast after the information has been created. Notifications regarding the deletion of information are broadcast before the information is deleted.

Several notifications are broadcast at various stages when the system goes to sleep and when the system wakes up. These notifications are **not** guaranteed to be broadcast. For example, if the system goes to sleep because the user removes the batteries, sleep notifications are not sent. Thus, these notifications are unsuitable for applications where external hardware must be shut off to conserve power before the system goes to sleep.

Constant	Description
`sysNotifyAntennaRaisedEvent`	Sent during `SysHandleEvent` when the antenna is raised on a Palm VII™ series device. Notification handlers may set the `handled` parameter to `true` to indicate that the antenna key down event has been handled.
`sysNotifyDisplayChangeEvent`	Sent just after the color table has been set to use a specific palette or just after `WinScreenMode` has changed the bit depth. For this notification, the `notifyDetailsP` parameter is a pointer to a `SysNotifyDisplayChangeDetailsType` struct.
`sysNotifyEarlyWakeupEvent`	Sent during `SysHandleEvent` immediately after the system has finished sleeping. The screen may still be turned off, and the system may quickly go back to sleep after handling a procedure alarm or charger event.

Constant	Description
`sysNotifyForgotPasswordEvent`	Sent if the user taps the Lost Password button in the Security application. The notification is sent after the user has confirmed that all private records should be deleted but before the deletion actually occurs.
`sysNotifyLateWakeupEvent`	Sent during `SysHandleEvent` immediately after the device has finished waking up. This is sent at the late stage of wakeup, after the screen has been turned on.
`sysNotifyMenuCmdBarOpenEvent`	Sent during `MenuHandleEvent` when it is about to display the menu shortcut command bar. This notification allows system extensions (such as "hacks" installed with the HackMaster program) to add their own buttons to the menu command bar. The handler should do so by calling `MenuCmdBarAddButton`. It also allows suppression of the command toolbar by setting `handled` to `true`. Applications that need to add their own buttons to the menu command bar should do so in response to a `menuCmdBarOpenEvent`. They should not register for this notification because an application should only add buttons if it is already the active application. The notification is sent after the event, immediately before the command toolbar is displayed.
`sysNotifyResetFinishedEvent`	Sent immediately after the system has finished a reset.

Constant	Description
`sysNotifySleepNotifyEvent`	Sent during `SysHandleEvent` immediately before the system is put to sleep. After the broadcast is complete, the system is put to sleep.
`sysNotifySleepRequestEvent`	Sent during `SysHandleEvent` processing when the system has decided to go to sleep. For this notification, the `notifyDetailsP` parameter is a pointer to a `SleepEventParamType` struct. If the `deferSleep` field is greater than 0, the system does not go to sleep. If you defer sleep, you must post a `keyDownEvent` with a `resumeSleepChr` to the event queue so that the system eventually goes to sleep. You must also set the command key bit in the `keyDownEvent` to signal that `resumeSleepChr` is a virtual character. Note that you may receive this notification several times before the system goes to sleep because notification handlers can delay the system sleep and resume it later.
`sysNotifySyncFinishEvent`	Sent immediately after a HotSync® operation is complete.
`sysNotifySyncStartEvent`	Sent immediately before a HotSync operation is begun.
`sysNotifyTimeChangeEvent`	Sent just after the system time has been changed using `TimSetSeconds`.

Miscellaneous Constants

Constant	Value	Description
`sysNotifyBroadcasterCode`	`'psys'`	Value passed as the creator ID of the broadcaster for notifications broadcast by the system.
`sysNotifyDefaultQueueSize`	15	Maximum number of nested broadcasts allowed.
`sysNotifyNormalPriority`	0	Typical priority value used when registering for notifications.
`sysNotifyVersionNum`	1	Current notification manager version. This number is stored in the system feature `sysFtrNumNotifyMgrVersion`.

Notification Functions

SysNotifyBroadcast

Purpose Synchronously send a notification to all applications registered for the given event type.

Prototype

```
Err SysNotifyBroadcast
(SysNotifyParamType *notify)
```

Parameters

<-> `notify` Information about the notification to broadcast.

Result Returns one of the following error codes:

`errNone` No error.

`sysNotifyErrBroadcastBusy` The broadcast stack limit has already been reached.

`sysErrParamErr`
: The background thread is broadcasting the notification and the `notify` parameter is `NULL`.

`sysNotifyErrNoStackSpace`
: There is not enough space on the stack for the notification.

Comments

When you call this function, the notification you specify is broadcast to all interested parties. The broadcast is performed synchronously, meaning that the system broadcasts the notification immediately and waits for each interested party to perform its notification handler and return before the `SysNotifyBroadcast` call returns. This notification occurs in priority order.

The system allows nested notifications; that is, the recipient of a notification might broadcast a new notification, whose recipient might broadcast another new notification and so on. The constant `sysNotifyDefaultQueueSize` specifies how many levels of nested notification are allowed. If you reach this limit, the error `sysNotifyErrBroadcastBusy` is returned and your notification is not broadcast. To avoid reaching the limit, use `SysNotifyBroadcastDeferred` instead of `SysNotifyBroadcast` in your notification handlers. This ensures that the notification will not be broadcast until the top of the event loop.

WARNING! Do not call `SysNotifyBroadcast` from code that might be called from a background task (such as a trap patch) with the memory semaphore reserved. Use `SysNotifyBroadcastDeferred` instead.

Compatibility

Implemented only if Notification Feature Set is present.

SysNotifyBroadcastDeferred

Purpose Enqueue a notification for later broadcast.

Prototype

```
Err SysNotifyBroadcastDeferred
(SysNotifyParamType *notify, Int16 paramSize)
```

Parameters

<-> `notify`	The notification to enqueue. See `SysNotifyParamType`.
-> `paramSize`	Size of the data pointed to by the field `notify->notifyDetailsP`.

Result Returns one of the following error codes:

`errNone`	No error.
`memErrNotEnoughSpace`	There is not enough memory to allocate a new notification entry in the queue.
`sysErrParamErr`	`paramSize` is a negative number.
`sysNotifyErrQueueFull`	The queue has reached its maximum number of entries.

Comments This function is similar to `SysNotifyBroadcast` except that the broadcast does not take place until the top of the event loop (specifically, the next time `EvtGetEvent` is called). The system copies the `notify` structure to a new memory chunk, which is disposed of upon completion of the broadcast. (The `paramSize` parameter is used when copying the `notifyDetailsP` portion of the `notify` structure.)

Compatibility Implemented only if Notification Feature Set is present.

SysNotifyRegister

Purpose Register to receive a notification.

Prototype

```
Err SysNotifyRegister (UInt16 cardNo,
LocalID dbID, UInt32 notifyType,
SysNotifyProcPtr callbackP, Int8 priority,
void *userDataP)
```

Parameters

-> `cardNo`	Number of the storage card on which the application or code resource resides.
-> `dbID`	Local ID of the application or code resource.
-> `notifyType`	The event that the application wants to receive notifications about. See Notification Manager Event Constants.
-> `callbackP`	`NULL` to receive the notification as an application launch code. If your code does not have a `PilotMain` function (for example, if it is a shared library), pass a pointer to a function that should be called when the notification is broadcast. See `SysNotifyProcPtr`.
-> `priority`	The priority with which the application should receive the event. Most applications and other code resources should always use `sysNotifyNormalPriority`. In rare circumstances, you may need to ensure that your code is notified toward the beginning or toward the end of the notification sequence. If so, specify a value between –15 and +15. The smaller the priority, the earlier your code is notified.
-> `userDataP`	Caller-defined data to pass to the notification handler.

Result Returns one of the following error codes:

`errNone`	No error.

`sysErrParamErr` The database ID is `NULL`.

`sysNotifyErrDuplicateEntry`
This application is already registered to receive this notification.

Comments Call this function when your code should receive a notification that a specific event has occurred or is about to occur. See Notification Manager Event Constants for a list of the events about which you can receive notifications. Once you register for a notification, you remain registered to receive it until a system reset occurs or until you explicitly unregister using `SysNotifyUnregister`.

If you're writing an application, you should pass `NULL` as the `callbackP` parameter. In this case, the system notifies your application by sending it the `sysAppLaunchCmdNotify` launch code. This launch code's parameter block points to a `SysNotifyParamType` structure containing details about the notification.

If your code is not in an application, for example, it is in a shared library or a separately loaded code resource, then receiving a launch code is not possible. In this case, pass a pointer to a callback function in `callbackP`. This callback should follow the prototype shown in `SysNotifyProcPtr`. Note that you should always supply a card number and database ID to `SysNotifyRegister`, even if you specify a callback function.

IMPORTANT: Because the `callbackP` pointer is used to directly call the function, the pointer must remain valid from the time `SysNotifyRegister` is called to the time the notification is broadcast. If the function is in a shared library, you must keep the library open. If the function is in a separately loaded code resource, the resource must remain locked while registered for the notification. When you close a library or unlock a resource, you must first unregister for any notifications. If you don't, the system will crash when the notification is broadcast.

Whether the notification handler is responding to `sysAppLaunchCmdNotify` or uses the callback function, the

notification handler may perform any processing necessary. As with most launch codes, it's not possible to access global variables. If the handler needs access to any particular value to respond to the notification, pass a pointer to that value in the `userDataP` parameter. The system passes this pointer back to your application or callback function in the launch code's parameter block.

The notification handler may unregister for this notification or register for other notifications. It may also broadcast another notifications; however, it's recommended that you use `SysNotifyBroadcastDeferred` to do this so as not to overflow the broadcast stack.

You may display a user interface in your notification handler; however, you should be careful when you do so. Many of the notifications are broadcast during `SysHandleEvent`, which means your application event loop might not have progressed to the point where it is possible for you to display a user interface, or you may overflow the stack by displaying a user interface at this stage. See Notification Manager Event Constants to learn which notifications are broadcast during `SysHandleEvent`.

Compatibility Implemented only if Notification Feature Set is present.

SysNotifyUnregister

Purpose Cancel notification of the given event.

Prototype

```
Err SysNotifyUnregister(UInt16 cardNo,
LocalID dbID, UInt32 notifyType, Int8 priority)
```

Parameters

-> `cardNo`	Number of the storage card on which the application or code resource resides.
-> `dbID`	Local ID of the application or code resource.
-> `notifyType`	The event to unregister for. See Notification Manager Event Constants.

-> `priority` The priority value you passed as part of `SysNotifyRegister`.

Result Returns one of the following error codes:

`errNone` No error.

`sysNotifyErrEntryNotFound` The application wasn't registered to receive these notifications.

Comments Use this function to remove your code from the list of those that receive notifications about a particular event. This function is particularly necessary if you are writing a shared library or a separately loaded code resource that receives notifications. When the resource is unloaded, it must unregister for all of its notifications, or the system will crash when the notification is broadcast.

Compatibility Implemented only if Notification Feature Set is present.

Application-Defined Functions

SysNotifyProcPtr

Purpose Handle a notification.

Prototype

```
Err (*SysNotifyProcPtr)
(SysNotifyParamType *notifyParamsP)
```

Parameters

-> `notifyParamsP` Pointer to a structure that contains the notification event that occurred and any other information about it. See `SysNotifyParamType`.

Result Always return 0.

Comments You pass this function as a parameter to `SysNotifyRegister` when registering code that does not have a `PilotMain` for a notification. See the description of `SysNotifyRegister` for advice on writing a notification handler.

IMPORTANT: Because the `callbackP` pointer is used to directly call the function, the pointer must remain valid from the time `SysNotifyRegister` is called to the time the notification is broadcast. If the function is in a shared library, you must keep the library open. If the function is in a separately loaded code resource, the resource must remain locked while registered for the notification. When you close a library or unlock a resource, you must first unregister for any notifications. If you don't, the system will crash when the notification is broadcast.

37

Overlay Manager

This chapter describes the overlay manager API as declared in the header file `OverlayMgr.h`. It discusses the following topics:

- Overlay Manager Data Structures
- Overlay Manager Constants
- Overlay Manager Functions

For more information on the overlay manager, see the section "Using Overlays to Localize Resources" on page 318 in the "Localized Applications" chapter of the *Palm OS Programmer's Companion*.

Overlay Manager Data Structures

OmLocaleType

The `OmLocaleType` struct specifies a locale.

```
typedef struct {
  UInt16  language;
  UInt16  country;
} OmLocaleType;
```

Field Descriptions

`language` The language spoken in the locale. This value is one of the `LanguageType` constants.

`country` The country or region where the language is spoken. This value is one of the `CountryType` constants.

OmOverlayRscType

The `OmOverlayRscType` struct specifies an overlay of one resource. You create a resource overlay using the tools provided in the Palm OS® SDK.

```
typedef struct {
  OmOverlayKind overlayType;
  UInt32        rscType;
  UInt16        rscID;
  UInt32        rscLength;
  UInt32        rscChecksum;
} OmOverlayRscType;
```

Your code should treat the `OmOverlayRscType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`overlayType` Specifies the action to take with the resource. This can be one of the following:

- `omOverlayKindAdd` Add a resource that doesn't exist in the base database. Supported in overlay manager versions 3 and up.
- `omOverlayKindBase` Description of a base resource. (This appears only in the base database.) Supported in overlay manager versions 3 and up.
- `omOverlayKindReplace` Replace a resource in the base database.

`rscType` The type of the resource to be overlaid.

`rscID` The ID of the resource to be overlaid.

`rscLength` The size in bytes of the resource to be overlaid as it appears in the base database.

`rscChecksum` A checksum of the resource to be overlaid as it appears in the base database.

OmOverlaySpecType

The `OmOverlaySpecType` struct defines a resource of type `'ovly'`. This resource is required in the overlay database and may optionally exist in the corresponding base database as well.

```
typedef struct {
  UInt16            version;
  UInt32            flags;
  UInt32            baseChecksum;
  OmLocaleType      targetLocale;
  UInt32            baseDBType;
  UInt32            baseDBCreator;
  UInt32            baseDBCreateDate;
  UInt32            baseDBModDate;
  UInt16            numOverlays;
  OmOverlayRscType  overlays[0];
} OmOverlaySpecType;
```

Your code should treat the `OmOverlaySpecType` structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

`version` Version number for the overlay manager. The current version is `omOverlayVersion`.

`flags` 0, or one or more of the following flags:

- `omSpecAttrForBase` An `'ovly'` resource in the base database describes the overlay.
- `omSpecAttrStripped` Localized resources in the base database are stripped.

`baseChecksum` Checksum of all of the checksum values for the overlaid resources.

`targetLocale` Locale of this database. See `OmLocaleType`.

`baseDBType` Type of the base database.

`baseDBCreator` Creator of the base database.

`baseDBCreateDate` Creation date of the base database.

`baseDBModDate` Modification date of the base database.

`numOverlays` Number of elements in the `overlays` array.

`overlays` An array of `OmOverlayRscType` structs identifying each change or action the overlay is making to a resource.

Overlay Manager Constants

Constant	Value	Description
`omOverlayVersion`	0x0004	Current version for the overlay manager. This version number controls which types of overlay actions are supported.
`omOverlayDBType`	`'ovly'`	Database type for overlay databases.

Constant	Value	Description
`omOverlayRscType`	`'ovly'`	Symbolic name of an overlay resource that is contained in both the base database and the overlay database. This resource is defined by the `OmOverlaySpecType` struct.
`omOverlayRscID`	1000	Resource ID of the overlay resource in both the base database and the overlay database.
`omFtrCreator`	`'ovly'`	Creator value used for the `omFtrShowErrorsFlag` feature.
`omFtrShowErrorsFlag`	0	Feature that controls the number of error messages displayed by the overlay manager. If this feature is set to `true`, the overlay manager may display several more error messages when validating an overlay against its base database. This feature only takes effect when the error checking level is set to full (common on debug ROMs, not on release ROMs). Use `FtrGet` and `FtrSet` to retrieve and set this value.

Overlay Manager Functions

OmGetCurrentLocale

Purpose Return the current locale.

Prototype

```
void OmGetCurrentLocale
(OmLocaleType *currentLocale)
```

Parameters

<- `currentLocale`
Points to an `OmLocaleType` struct that identifies the current locale.

Result Returns nothing.

Comments This function returns the current locale. The current locale controls which overlays are used for resource databases. For example, suppose you have one application and two associated overlays installed, one for US English and one for British English. In this case, if the country specified in the locale returned by this function is `cUnitedKingdom`, the British English overlay is used for the application. If the country returned is `cUnitedStates`, the US English overlay is used.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `OmGetSystemLocale`

OmGetIndexedLocale

Purpose Return a system locale by index.

Prototype

```
Err OmGetIndexedLocale (UInt16 localeIndex,
OmLocaleType *theLocale)
```

Parameters

-> `localeIndex`	Zero-based index of the locale to return.
<- `theLocale`	Points to an `OmLocaleType` struct that identifies the locale at that index.

Result Returns `errNone` upon success, or `omErrInvalidLocaleIndex` if the index is out of bounds.

Comments You can use this function in a loop to determine how many overlays are installed for system resources. Each system overlay found determines a separate valid system locale. Any locale returned by this function can be sent to `OmSetSystemLocale` to change the system locale.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `OmGetSystemLocale`

OmGetRoutineAddress

Purpose Return the address of an overlay manager function.

Prototype
```
void *OmGetRoutineAddress (OmSelector inSelector)
```

Parameters

-> `inSelector` One of the routine selectors defined in `OverlayMgr.h`.

Result Returns the address of the corresponding function. Returns `NULL` if an invalid routine selector is passed.

Comments You typically use this function to determine whether an overlay manager function exists on the device. As future releases of Palm OS add new functions, older devices with earlier versions of the overlay manager will not implement these newer functions. If `OmGetRoutineAddress` returns `NULL`, the function is unavailable.

Compatibility Implemented only if 3.5 New Feature Set is present.

OmGetSystemLocale

Purpose Return the system locale.

Prototype
```
void OmGetSystemLocale
(OmLocaleType *systemLocale)
```

Parameters

<- `systemLocale` Points to an `OmLocaleType` struct that identifies the system locale.

Result Returns nothing.

Comments You typically don't use this function. Instead, use `OmGetCurrentLocale`, which returns the locale that determines which overlays are used.

The system locale is saved in the storage heap header and persists across soft resets. When the device is reset, the system locale is used to set the current locale.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `OmGetCurrentLocale`

OmLocaleToOverlayDBName

Purpose Return the overlay database's name given the base database name and the locale.

Prototype

```
Err OmLocaleToOverlayDBName
(const Char *baseDBName,
const OmLocaleType *targetLocale,
Char *overlayDBName)
```

Parameters

-> `baseDBName` The name of the base resource database associated with the overlay.

-> `targetLocale` The locale to which this overlay applies. See `OmLocaleType`. Pass `NULL` to use the current locale.

<- `overlayDBName` The overlay database name given the base database name and the target locale. This buffer must be at least `dmDBNameLength` bytes.

Result Returns `errNone` upon success, or `omErrUnknownLocale` if the `targetLocale` parameter is invalid.

Comments The appropriate overlay database name is currently:

baseDBName_llCC

where:

baseDBName The name of the base database as you passed it in.

ll A two-character code identifying the language.

CC A two-character code identifying the country.

The base database name is truncated if necessary to allow for this suffix.

For example, the base database "MemoPad" might have an overlay for US English named "MemoPad_enUS".

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `OmOverlayDBNameToLocale`

OmOverlayDBNameToLocale

Purpose Return an overlay database's locale given its name.

Prototype

```
Err OmOverlayDBNameToLocale
(const Char *overlayDBName,
OmLocaleType *overlayLocale)
```

Parameters

-> `overlayDBName`
The name of the overlay database.

<- `overlayLocale`
Points to an `OmLocaleType` structure identifying the overlay's locale.

Result Returns `errNone` upon success, `omErrBadOverlayDBName` if the string `overlayDBName` is not long enough to have a locale suffix, or `omErrUnknownLocale` if the locale cannot be determined.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `OmLocaleToOverlayDBName`

OmSetSystemLocale

Purpose Set the system locale and reset the device.

Prototype

```
Err OmSetSystemLocale
(const OmLocaleType *systemLocale)
```

Parameters

-> `systemLocale` An `OmLocaleType` structure specifying the locale to switch the system to.

Result Returns `errNone` upon success, or one of the following if an error occurs:

`omErrUnknownLocale`
There is no system overlay for `systemLocale`.

`omErrInvalidLocale`
The system overlay for `systemLocale` has been found but is invalid.

`dmErrInvalidParam`
An error occurred while opening the overlay.

`dmErrMemError` A memory error occurred while opening the overlay.

`dmErrDatabaseOpen`
The system overlay was already open.

Comments This function changes the system locale to the specified locale if it exists. It first determines that the system overlay exists for the requested locale and that it matches the base system database. If so, it updates the system locale information saved in the storage heap header and resets the device. After the device is reset, the current locale is set to the system locale.

A Palm OS device has a default system locale hard-coded into the ROM. This locale is used to set the system locale after a hard reset or any time that the storage heap header is invalid. The storage heap header is typically only invalid when the device is turned on for the first time.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `OmGetSystemLocale`

38

Password

This chapter provides reference material for the password API. The password API is declared in the header file `Password.h`.

Password Functions

PwdExists

Purpose Return `true` if the system password is set.

Prototype `Boolean PwdExists()`

Parameters None

Result Returns `true` if the system password is set.

PwdRemove

Purpose Remove the encrypted password string and recover data hidden in databases.

Prototype `void PwdRemove(void)`

Parameters None

Result Returns nothing.

PwdSet

Purpose Use a passed string as the new password. The password is stored in an encrypted form.

Prototype

```
void PwdSet (Char* oldPassword, Char* newPassword)
```

Parameters

`oldPassword`	The old password must be successfully verified or the new password isn't accepted
`newPassword`	`Char*` to a string to use as the password. `NULL` means no password.

Result Returns nothing.

PwdVerify

Purpose Verify that the string passed matches the system password.

Prototype

```
Boolean PwdVerify (Char* string)
```

Parameters

`string`	String to compare to the system password. `NULL` means no current password.

Result Returns `true` if the string matches the system password.

39

Pen Manager

This chapter provides reference material for the pen manager. The pen manager API is declared in the header file `PenMgr.h`.

For more information on the pen manager, see the section "Receiving User Input" in the *Palm OS Programmer's Companion*.

Pen Manager Functions

PenCalibrate

Purpose Set the calibration of the pen.

Prototype

```
Err PenCalibrate (PointType* digTopLeftP,
PointType* digBotRightP, PointType* scrTopLeftP,
PointType* scrBotRightP)
```

Parameters

`digTopLeftP`	Digitizer output from top-left coordinate.
`digBotRightP`	Digitizer output from bottom-right coordinate.
`scrTopLeftP`	Screen coordinate near top-left corner.
`scrBotRightP`	Screen coordinate near bottom-right corner.

Result Returns 0 if no error.

Comments Called by Preferences application when calibrating pen.

See Also `PenResetCalibration`

PenResetCalibration

Purpose Reset the calibration in preparation for calibrating the pen again.

Prototype

```
Err PenResetCalibration (void)
```

Parameters None.

Result Always returns 0.

Comments Called by Preferences application before capturing points when calibrating the pen.

See Also PenCalibrate

WARNING! The digitizer is off after calling this routine and must be calibrated again!

40

Preferences

This chapter provides reference material for the preferences API. The preferences API is declared in the header file `Preferences.h`.

Preferences Functions

PrefGetAppPreferences

Purpose Return a copy of an application's preferences. Sometimes, for variable length resources, this routine is called twice:

- Once with a `NULL` pointer and size of zero to find out how many bytes need to be read.
- A second time with an allocated buffer allocated of the correct size. Note that the application should always check that the return value is greater than or equal to `prefsSize`.

Prototype

```
Int16 PrefGetAppPreferences (UInt32 creator,
UInt16 id, void* prefs, UInt16* prefsSize,
Boolean saved)
```

Parameters

`creator`	Application creator.
`id`	ID number (lets an application have multiple preferences).
`prefs`	Pointer to a buffer to hold preferences.
`prefsSize`	Pointer to size the buffer passed.

`saved`	If `true`, retrieve the saved preferences. If `false`, retrieve the current preferences.

Result Returns the constant `noPreferenceFound` if the preference resource wasn't found.

If the preference resource was found, the application should check that the value in `prefsSize` is equal or less than the return value. If it's greater than the size passed, then some bytes were not retrieved.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `PrefSetPreferences`, `PrefGetAppPreferencesV10`

PrefGetAppPreferencesV10

Purpose Return a copy of an application's preferences.

Prototype

```
Boolean PrefGetAppPreferencesV10 (UInt32 type,
Int16 version, void* prefs, UInt16 prefsSize)
```

Parameters

`type`	Application creator type.
`version`	Version number of the application.
`prefs`	Pointer to a buffer to hold preferences.
`prefsSize`	Size of the buffer passed.

Result Returns `false` if the preference resource was not found or the preference resource contains the wrong version number.

Comments The content and format of an application preference is application-dependent.

Compatibility This function corresponds to the 1.0 version of `PrefGetAppPreferences`.

See Also `PrefSetPreferences`, `PrefGetAppPreferences`

PrefGetPreference

Purpose Return a system preference. Use this instead of PrefGetPreferences.

Prototype

```
UInt32 PrefGetPreference
(SystemPreferencesChoice choice)
```

Parameters

`choice`	System preference choice; see `Preferences.h` for available options.

Result Returns the system preference.

Comments This function replaces the 1.0 function PrefGetPreferences. While `PrefGetPreferences` only let you retrieve the whole system preferences structure, this function lets you specify which preferences to retrieve. You can also choose among different preferences using an ID, or choose to access the saved or unsaved preferences.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `PrefSetPreferences`, `PrefGetAppPreferences`, `PrefGetAppPreferencesV10`

PrefGetPreferences

Purpose Return a copy of the system preferences.

Prototype

```
void PrefGetPreferences (SystemPreferencesPtr p)
```

Parameters

`p`	Pointer to system preferences.

Result Returns nothing. Stores the system preferences in `p`.

Comments The `p` parameter points to a memory block allocated by the caller that is filled in by this function.

This function is often called in StartApplication to get localized settings.

See Also PrefSetPreferences

PrefOpenPreferenceDBV10

Purpose Return a handle to the system preference database.

Prototype

```
DmOpenRef PrefOpenPreferenceDBV10 (void)
```

Parameters Nothing.

Result Returns the handle, or 0 if an error results.

Comments This function is for system use only in Palm OS® 2.0 and later.

Compatibility This function corresponds to the 1.0 version of `PrefOpenPreferenceDB`.

See Also PrefGetPreferences, PrefSetPreferences

PrefSetAppPreferences

Purpose Set an application's preferences in the preferences database.

Prototype

```
void PrefSetAppPreferences (UInt32 creator,
UInt16 id, Int16 version, void* prefs,
UInt16 prefsSize, Boolean saved)
```

Parameters

`creator`	Application creator type.
`id`	Resource ID (usually 0).
`version`	Version number of the application.
`prefs`	Pointer to a buffer that holds preferences.
`prefsSize`	Size of the buffer passed.

`saved`	If `true`, set the saved preferences. If not, set the current preferences.

Result Returns nothing.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `PrefSetAppPreferencesV10`

PrefSetAppPreferencesV10

Purpose Save an application's preferences in the preferences database.

Prototype

```
void PrefSetAppPreferencesV10 (UInt32 creator,
Int16 version, void* prefs, UInt16 prefsSize)
```

Parameters

`creator`	Application creator type.
`version`	Version number of the application.
`prefs`	Pointer to a buffer holding preferences.
`prefsSize`	Size of the buffer passed.

Result Returns nothing.

Comments The content and format of an application preference is application-dependent.

Compatibility This function corresponds to the 1.0 version of `PrefSetAppPreferences`.

See Also `PrefSetAppPreferences`, `PrefGetPreferences`

PrefSetPreference

Purpose Set a system preference. Using this function instead of `PrefSetPreferences` allows you to set selected preferences without having to access the whole structure.

Prototype

```
void PrefSetPreference
(SystemPreferencesChoice choice, UInt32 value)
```

Parameters

`choice`	A `SystemPreferencesChoice` (see `Preferences.h`)
`value`	Value to assign to the item in `SystemPreferencesChoice`.

Result Returns the system preference.

Compatibility Implemented only if 2.0 New Feature Set is present.

PrefSetPreferences

Purpose Set the system preferences.

Prototype

```
void PrefSetPreferences (SystemPreferencesPtr p)
```

Parameters

`p`	Pointer to system preferences.

Result Returns nothing.

Comments Unless there's a reason for you to access the whole preferences structure, call PrefSetPreference instead.

See Also `PrefGetPreferences`

41

Rectangles

This chapter provides reference material for the rectangles API, declared in the header file `Rect.h`.

Rectangle Functions

RctCopyRectangle

Purpose Copy the source rectangle to the destination rectangle.

Prototype

```
void RctCopyRectangle
(const RectangleType* srcRectP,
RectangleType* dstRectP)
```

Parameters

`srcRectP`	A pointer to the rectangle to be copied.
`dstRectP`	A pointer to the destination rectangle.

See Also `RctSetRectangle`

RctGetIntersection

Purpose Determine the intersection of two rectangles.

Prototype

```
void RctGetIntersection (const RectangleType* r1P,
const RectangleType* r2P, RectangleType* r3P)
```

Parameters

`r1P`	A pointer to a source rectangle.
`r2P`	A pointer to the other source rectangle.
`r3P`	Upon return, points to a rectangle representing the intersection of `r1` and `r2`.

Comments The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function returns in the `r3` parameter a pointer to the rectangle that represents the intersection of the first two rectangles.

If rectangles `r1` and `r2` do not intersect, `r3` contains a rectangle that begins at coordinates (0, 0) and has 0 width and 0 height.

RctInsetRectangle

Purpose Move all of the boundaries of a rectangle by a specified offset.

Prototype

```
void RctInsetRectangle (RectangleType* rP,
Coord insetAmt)
```

Parameters

`rP`	A pointer to the rectangle.
`insetAmt`	Number of pixels to move the boundaries. This can be a negative number.

Comments The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function adds `insetAmt` to the x and y values of the top-left coordinate and then adjusts the width and the height accordingly so that all of the sides of the rectangle are contracted or expanded by the same amount.

A positive `insetAmt` creates a smaller rectangle that is contained inside the old rectangle's boundaries. A negative `insetAmt` creates a larger rectangle that surrounds the old rectangle.

See Also `RctOffsetRectangle`

RctOffsetRectangle

Purpose Move the top and left boundaries of a rectangle by the specified values.

Prototype

```
void RctOffsetRectangle (RectangleType* rP,
Coord deltaX, Coord deltaY)
```

Parameters

`rP`	A pointer to the rectangle.
`deltaX`	Number of pixels to move the left boundary. This can be a negative number.
`deltaY`	Number of pixels to move the top boundary. This can be a negative number.

Comments The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function adds `deltaX` to the x value of the top-left coordinate and `deltaY` to the y value. The width and height are unchanged. Thus, this function shifts the position of the rectangle by the `deltaX` and `deltaY` amounts.

See Also `RctInsetRectangle`

RctPtInRectangle

Purpose Determine if a point lies within a rectangle's boundaries.

Prototype

```
Boolean RctPtInRectangle (Coord x, Coord y,
const RectangleType* rP)
```

Parameters

`x`	The x coordinate of the point.
`y`	The y coordinate of the point.
`rP`	The rectangle.

Result Returns `true` if the point (`x`, `y`) lies within the boundaries of rectangle `r`, `false` otherwise.

RctSetRectangle

Purpose Sets a rectangle's values.

Prototype

```
void RctSetRectangle (RectangleType* rP,
Coord left, Coord top, Coord width, Coord height)
```

Parameters

`rP`	A pointer to the rectangle to be set.
`left`	The x value for the top-left coordinate of the rectangle.
`top`	The y value for the top-left coordinate of the rectangle.
`width`	The rectangle's width.
`height`	The rectangle's height.

See Also `RctCopyRectangle`

42

Sound Manager

This chapter provides reference material for the sound manager.

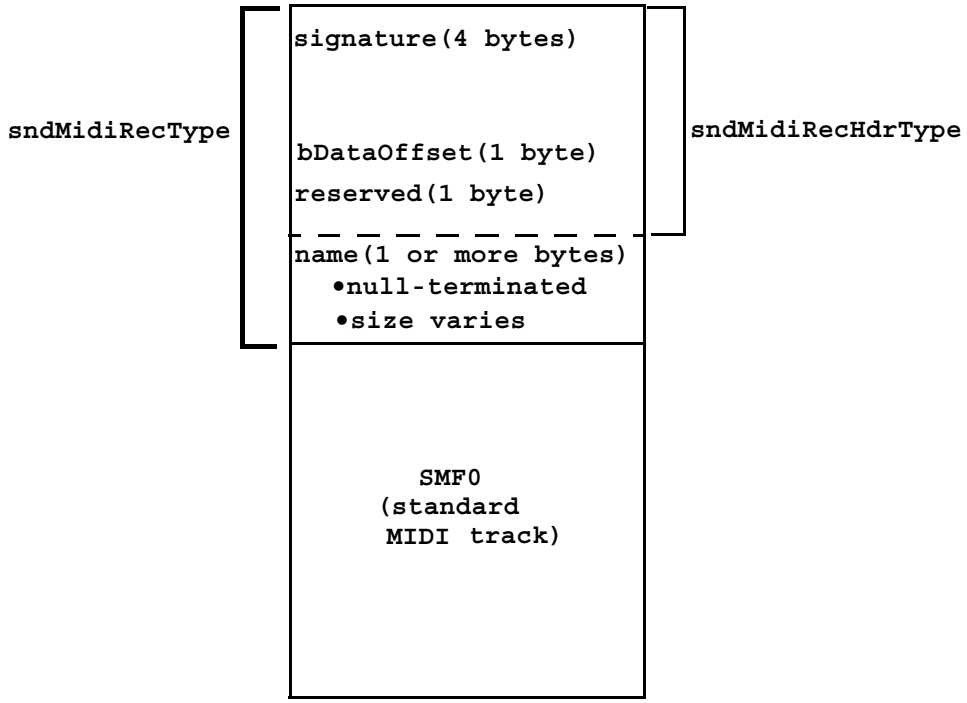
- Sound Manager Data Structures
- Application-Defined Functions
- Sound Manager Functions

The header file `SoundMgr.h` declares the API that this chapter describes. For more information on the sound manager, see the section "Sound" in the *Palm OS Programmer's Companion*.

Sound Manager Data Structures

This section describes the data structures that define the MIDI records and parameter blocks used by sound manager functions. Figure 42.1 depicts a Palm OS® MIDI record graphically.

Figure 42.1 Palm OS Midi Record



SndCallbackInfoType

The `SndCallbackInfoType` structure wraps the sound manager callback functions that you implement. See "Application-Defined Functions" for more information.

```
typedef struct SndCallbackInfoType {
  MemPtr  funcP;
  UInt32  dwUserData;
} SndCallbackInfoType;
```

Field Descriptions

`funcP`	Pointer to the callback function (`NULL` = no function).
`dwUserData`	Value to pass in `dwUserData` parameter of callback function.

SndCmdIDType

The `SndCmdIDType` enumerated type defines the commands that may be specified in the `cmd` field of the `SndCommandType`. Each command defines its own specific use of the `param1`, `param2`, and `param3` fields.

```
typedef enum SndCmdIDType {
  sndCmdFreqDurationAmp = 1,
  sndCmdNoteOn,
  sndCmdFrqOn,
  sndCmdQuiet
} SndCmdIDType;
```

Value Descriptions

`sndCmdFreqDurationAmp`	Play a sound, blocking for the entire duration (except for zero amplitude). `param1` = frequency in Hz `param2` = duration in milliseconds `param3` = amplitude (0 to `sndMaxAmp`) If value of `param3` is 0, returns immediately.
`sndCmdNoteOn`	Play sound at specified MIDI key index with max duration and velocity; return immediately, without waiting for playback to complete. Any other sound play request made before this one completes will interrupt it. `param1` = MIDI key index (0-127) `param2` = maximum duration in milliseconds `param3` = velocity (0 to 127) to be interpolated as amplitude.

`sndCmdFrqOn`	Similar to `sndCmdNoteOn` except note to play is specified as frequency in Hz. `param1` = frequency in Hz `param2` = maximum duration in milliseconds `param3` = amplitude (0 - sndMaxAmp)
`sndCmdQuiet`	Stop playback of current sound. `param1` = 0 `param2` = 0 `param3` = 0

IMPORTANT: `SndDoCmd` in versions of Palm OS before 3.0 will generate a fatal error on anything other than `sndCmdFreqDurationAmp`. For this reason, applications wishing to take advantage of these new commands while staying compatible with the earlier version of the OS, **must** avoid using these commands when running on OS versions less than v3.0. Beginning with v3.0, `SndDoCmd` returns `sndErrBadParam` when an unknown command is passed.

SndCommandType

The `SndCommandType` structure is passed as the value of the `cmdP` parameter to the `SndDoCmd` function. Its parameters are defined by the `SndCmdIDType`.

```
typedef struct SndCommandType {
  SndCmdIDType    cmd;
  UInt8           reserved;
  Int32           param1;
  UInt16          param2;
```

```
    UInt16          param3;
  } SndCommandType;
```

Field Descriptions

`cmd`	Command ID.
`reserved`	Reserved for future use.
`param1, param2, param3`	Use varies according to value of `cmd`.

SndMidiListItemType

When the `SndCreateMidiList` function returns `true`, its `entHP` parameter holds a handle to a memory chunk containing an array of `SndMidiListItemType` structures.

```
typedef struct SndMidiListItemType{
  Char    name[sndMidiNameLength];
  UInt32  uniqueRecID;
  LocalID dbID;
  UInt16  cardNo;
} SndMidiListItemType;
```

Field Descriptions

`name`	MIDI name including `NULL` terminator.
`uniqueRecID`	Unique ID of the record containing the MIDI file.
`dbID`	Database (file) ID.
`cardNo`	Number of the memory card on which the MIDI file resides.

SndMidiRecHdrType

The `SndMidiRecHdrType` structure defines the fixed-size portion of a Palm OS MIDI record. (See `SndCallbackInfoType`.)

```
typedef struct SndMidiRecHdrType {
  UInt32  signature;
  UInt8   bDataOffset;
```

```
    UInt8   reserved;
  } SndMidiRecHdrType;
```

Field Descriptions

`signature`	Set to `sndMidiRecSignature`.
`bDataOffset`	Offset from the beginning of the record to the Standard MIDI File data stream.
`reserved`	Set to zero.

SndMidiRecType

The `SndMidiRecType` structure defines a variable-length header precedes the actual MIDI data in a Palm OS MIDI record. It consists of a fixed-size MIDI record header followed by the name of the MIDI track.

```
typedef struct SndMidiRecType {
  SndMidiRecHdrType hdr;
  Char              name[1];
} SndMidiRecType;
```

Field Descriptions

`hdr`	Fixed-size portion of the Palm OS MIDI record header. See `SndMidiRecHdrType`.
`name`	Track name: 1 or more characters including `NULL` terminator. The length of name, including `NULL` terminator, must not be greater than `sndMidiNameLength`. The `NULL` character must always be provided, even for tracks that have no name.

SndSmfCallbacksType

The `SndSmfCallbacksType` structure is passed as the value of the `callbacksP` parameter to the `SndPlaySmf` function.

```
typedef struct SndSmfCallbacksType {
  SndCallbackInfoType completion;
  SndCallbackInfoType blocking;
  SndCallbackInfoType reserved;
```

```
} SndSmfCallbacksType;
```

Field Descriptions

`completion`	Completion callback function (see `SndComplFuncType`).
`blocking`	Blocking hook callback function (see `SndBlockingFuncType`).
`reserved`	Reserved. Set to 0 before passing.

SndSmfChanRangeType

This `SndSmfChanRangeType` structure is passed as the value of the `chanRangeP` parameter to the `SndPlaySmf` function. It specifies a range of enabled channels. Events for channels outside this range are ignored.

If this structure is not passed, all channels in the track are ignored.

```
typedef struct SndSmfChanRangeType {
  UInt8   bFirstChan;
  UInt8   bLastChan;
} SndSmfChanRangeType;
```

Field Descriptions

`bFirstChan`	First MIDI channel (0-15 decimal).
`bLastChan`	Last MIDI channel (0-15 decimal).

SndSmfOptionsType

The `SndSmfOptionsType` structure is passed as the value of the `selP` parameter to the `SndPlaySmf` function.

```
typedef struct SndSmfOptionsType {
  UInt32      dwStartMilliSec;
  UInt32      dwEndMilliSec;
  UInt16      amplitude;
  Boolean     interruptible;
  UInt8       reserved1;
```

```
    UInt32      reserved;
  } SndSmfOptionsType;
```

Field Descriptions

`dwStartMilliSec` Position at which to begin playback, expressed as number of milliseconds from beginning of the track. 0 means start from the beginning of the track.

This field is used as input for `sndSmfCmdPlay` and output for `sndSmfCmdDuration`.

`dwEndMilliSec` Position at which to stop playback, expressed as number of milliseconds from beginning of track. `sndSmfPlayAllMilliSec` means play the entire track; the default is to play the entire track if this structure is not passed in.

This field is used as input for `sndSmfCmdPlay` and output for `sndSmfCmdDuration`.

`amplitude` Used only for `sndSmfCmdPlay`. Specifies relative volume. Possible values range from 0 to `sndMaxAmp`, inclusively. The default is `sndMaxAmp` if this structure is not passed in. If 0, the sound is not played and the call returns immediately.

`interruptible` Used only for `sndSmfCmdPlay`. If `true`, sound play will be interrupted if user interacts with the controls (digitizer, buttons, etc.) even if the interaction does not generate a sound command. If `false`, playback is not interrupted. The default behavior is `true` if this structure is not passed in.

`reserved1` Reserved for future use.

`reserved` Reserved. Set to 0 before passing.

Sound Manager Functions

SndCreateMidiList

Purpose Generate a list of MIDI records having a specified creator.

Prototype

```
Boolean SndCreateMidiList (UInt32 creator,
Boolean multipleDBs, UInt16* wCountP,
MemHandle *entHP)
```

Parameters

`->creator`	Creator of database in which to find MIDI records. Pass `0` for wildcard.
`->multipleDBs`	Pass `true` to search multiple databases for MIDI records. Pass `false` to search only in the first database found that meets search criteria.
`<->wCountP`	When the function returns, contains the number of MIDI records found.
`<->entHP`	When the function returns, this handle holds a a memory chunk containing an array of `SndMidiListItemType` if MIDI records were found.

Result Returns `false` if no MIDI records were found, `true` if MIDI records were found. When this function returns `true`, it updates the `wCountP` parameter to hold the number of MIDI records found and updates the `entHP` parameter to hold a handle to an array of `SndMidiListItemType` structures. Each record of this type holds the name, record ID, database ID, and card number of a MIDI record.

Comments This function is useful for displaying lists of sounds residing on the Palm device as MIDI records.

Compatibility Implemented only if `3.0 New Feature Set` is present.

See Also `DmFindRecordByID`, `DmOpenDatabase`, `DmQueryRecord`, `DmOpenDatabaseByTypeCreator`,"Rock Music" sample code

SndDoCmd

Purpose Send a sound manager command to a specified sound channel.

Prototype

```
Err SndDoCmd (void* channelP, SndCommandPtr cmdP,
Boolean noWait)
```

Parameters

-> `channelP`	Pointer to sound channel. Present implementation doesn't support multiple channels. Must be `NULL`. Passing `NULL` for the channel pointer causes the command to be sent to the shared sound channel; currently, this is the only option.
-> `cmdP`	Pointer to a `SndCommandType` holding a parameter block that specifies the note to play, its duration, and amplitude.
-> `noWait`	Because asynchronous mode is not yet supported for all commands, you must pass `0` for this value. In the future, pass `0` to await completion (synchronous) and pass a nonzero value to specify immediate return (asynchronous).

Result

0	No error.
`sndErrBadParam`	Invalid parameter.
`sndErrBadChannel`	Invalid channel pointer.
`sndErrQFull`	Sound queue is full.

Comments This function is useful for simple sound playback applications, such as playing a single note to provide user feedback. In addition to providing the same behavior it did in versions 1.0 and 2.0 of Palm OS, (specify the frequency, duration, and amplitude of a single note to be played) new command selectors provided in Palm OS 3.0 and higher allow you to use MIDI values to specify pitch, duration, and

amplitude of the note to play, and to stop the note currently being played.

Compatibility Commands (see SndCmdIDType) other than sndCmdFreqDurationAmp are implemented only if 3.0 New Feature Set is present.

See Also SndPlaySmf

SndGetDefaultVolume

Purpose Return default sound volume levels cached by sound manager.

Prototype

```
void SndGetDefaultVolume (UInt16* alarmAmpP,
UInt16* sysAmpP, UInt16* masterAmpP)
```

Parameters

<-> alarmAmpP	Pointer to storage for alarm amplitude.
<-> sysAmpP	Pointer to storage for system sound amplitude.
<-> masterAmpP	Pointer to storage for master amplitude.

Result Returns nothing.

Comments Any pointer arguments may be passed as NULL. In that case, the corresponding setting is not returned.

SndPlaySmf

Purpose

Performs the operation specified by the `cmd` parameter: play the specified standard MIDI file (SMF) or return the number of milliseconds required to play the entire SMF.

Prototype

```
Err SndPlaySmf (void* chanP, SndSmfCmdEnum cmd,
UInt8* smfP, SndSmfOptionsType* selP,
SndSmfChanRangeType* chanRangeP,
SndSmfCallbacksType* callbacksP, Boolean bNoWait)
```

Parameters

`chanP`
The sound channel used to play the sound. This value must always be `NULL` because current versions of Palm OS provide only one sound channel that all applications share.

`cmd`
The operation to perform, as specified by one of the following selectors:

`sndSmfCmdPlay`
Play the selection synchronously.

`sndSmfCmdDuration`
Return the duration of the entire SMF, expressed in milliseconds.

`-> smfP`
Pointer to the SMF data in memory. This pointer can reference a valid `SndCallbackInfoType` followed by MIDI data, or it can point directly to the beginning of the SMF data.

`-> selP`
`NULL` or a pointer to a `SndSmfOptionsType` specifying options for playback volume, position in the SMF from which to begin playback, and whether playback can be interrupted by user interaction with the display. See the `SndSmfOptionsType` for the default behavior specified by a `NULL` value.

-> `chanRangeP`	`NULL` or a pointer to a `SndSmfChanRangeType` specifying a continuous range of MIDI channels 0 -15 to use for playback. If this value is `NULL`, all tracks are played.
-> `callbacksP`	`NULL` or a pointer to a `SndSmfCallbacksType` that holds your callback functions. Functions of type `SndBlockingFuncType` execute periodically while a note is playing, and functions of type `SndComplFuncType` execute after playback of the SMF completes. For more information, see the "Application-Defined Functions" section.
`bNoWait`	This value is ignored. This function always finishes playing the SMF selection before returning; however, you can execute a callback function while the SMF is playing.

Result

Returns `0` if no error. When an error occurs, this function returns one of the following values; for more information see the `SoundMgr.h` file included with the Palm OS 3.X SDK:

`sndErrBadParam`	Bogus value passed to this function.
`sndErrBadChannel`	Invalid sound channel.
`sndErrMemory`	Insufficient memory.
`sndErrOpen`	Tried to open channel that's already open.
`sndErrQFull`	Can't accept more notes.
`sndErrQEmpty`	Internal use; never returned to applications.
`sndErrFormat`	Unsupported data format.
`sndErrBadStream`	Invalid data stream.
`sndErrInterrupted`	Play was interrupted.

Comments

Although this call is synchronous, a callback function can be called while a note is playing. If the callback does not return before the

number of system ticks required to play the current sound have elapsed, the next note in the SMF will not start on time.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `SndPlaySmfResource`, `SndDoCmd`,`SndCreateMidiList`

SndPlaySmfResource

Purpose Plays a MIDI sound read out of an open resource database.

Prototype

```
Err SndPlaySmfResource (UInt32 resType,
Int16 resID,
SystemPreferencesChoice volumeSelector)
```

Parameters

-> `resType` Card number of the database containing the SMF resource.

-> `resID` Resource ID of the SMF resource.

-> `volumeSelector`
Volume setting to use. The volume is read out of the system preferences. Possible values for this parameter are:

```
prefSysSoundVolume
prefGameSoundVolume
prefAlarmSoundVolume
```

Result Returns 0 upon success, otherwise one of the following:

`sndErrBadParam`	The `volumeSelector` parameter has an invalid value or the SMF resource has invalid data.
`dmErrCantFind`	The specified resource does not exist on the specified card.

or any error code returned by `SndPlaySmf`.

Comments This is a convenience function to be used in place of SndPlaySmf. It plays an SMF sound stored in a resource database. This function plays the entire sound on all MIDI channels at the volume specified and allows the sound to be interrupted by a key down event or a digitizer event. No callbacks are specified.

Compatibility Implemented only if 3.2 New Feature Set is present.

SndPlaySystemSound

Purpose Play a standard system sound.

Prototype

```
void SndPlaySystemSound (SndSysBeepType beepID)
```

Parameters -> `beepID` System sound to play.

Result Returns nothing.

Comments The `SndSysBeepType` enum is defined in `SoundMgr.h` as follows:

```
typedef enum SndSysBeepType {
  sndInfo = 1,
  sndWarning,
  sndError,
  sndStartUp,
  sndAlarm,
  sndConfirmation,
  sndClick
  } SndSysBeepType;
```

Note that in versions of Palm OS prior to 3.0, all of these sounds were synchronous and blocking. In Palm OS 3.0 and higher, `sndAlarm` still blocks, but the rest of these system sounds are implemented asynchronously.

Application-Defined Functions

This section describes callback functions to be executed by the SndPlaySmf function.

SndComplFuncType

Purpose Executed after playback of the SMF completes.

Prototype

```
void SndComplFuncType (void* chanP,
UInt32 dwUserData)
```

Parameters

-> `chanP` The sound channel used to play the sound. This value must always be `NULL` because current versions of Palm OS provide only one sound channel that all applications share.

-> `dwUserData` Application-defined data that this function needs to access, or `NULL`.

Result Returns nothing.

See Also `SndSmfCallbacksType`

SndBlockingFuncType

Purpose A non-blocking callback function that is executed periodically during playback of the SMF.

Prototype

```
Boolean SndBlockingFuncType (void* chanP,
UInt32 dwUserData, Int32 sysTicksAvailable)
```

Parameters

-> `chanP` The sound channel used to play the sound. This value must always be `NULL` because current versions of Palm OS provide only one sound channel that all applications share.

-> `dwUserData` Application-defined data that this function needs to access.

-> `sysTicksAvailable` The maximum amount of time available for completion of this function, or `NULL`.

Result Returns `true` to continue playback, or `false` to cancel playback.

Comments Suggested uses for this function include updating the user interface or checking for user input.

See Also `SndSmfCallbacksType`, `SndPlaySmf`

43

Standard IO

This chapter provides reference material for the standard IO API:

- Standard IO Functions
- Standard IO Provider Functions
- Application-Defined Function

The header files `StdIOPalm.h` and `StdIOProvider.h` declare the standard IO API. For more information on using the standard IO API, see the chapter "Standard IO Applications"in the *Palm OS Programmer's Companion*.

Standard IO Functions

The macros and functions in this section enable standard IO.

fgetc

Purpose Macro that calls `Siofgetc` to return the next character from the input stream.

Prototype `fgetc(fs)`

Parameters

-> `fs`	An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

Result The next character from the input stream. The return value `EOF` indicates an error occurred.

fgets

Purpose Macro that calls `Siofgets` to return a string from the input stream.

Prototype

```
fgets(strP, maxChars, fs)
```

Parameters

<- strP	A pointer to the returned string.
-> maxChars	The number of characters to read from the input stream, plus one for the null terminator.
-> `fs`	An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

Result A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns NULL.

Comments The returned string is always terminated by a null character.

fprintf

Purpose Macro that calls `Siofprintf` to write formatted output to an output stream.

Prototype

```
fprintf(fs, formatP, ...)
```

Parameters

-> `fs`	An output stream to which to write the formatted output. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.
-> `formatP`	A pointer to a format string that controls how subsequent arguments are converted for output.

-> ...	Zero or more parameters to be formatted as specified by the `formatP` string.

Result Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments This function internally calls StrVPrintF to do the formatting. See that function for details on which format specifications are supported.

fputc

Purpose Macro that calls `Siofputc` to write a character to the output stream.

Prototype

```
fputc(c, fs)
```

Parameters

-> `c`	A character to write to the output stream.
-> `fs`	An output stream to which to write the character. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

Result The character that was written. If an error occurs, the value `EOF` is returned.

fputs

Purpose Macro that calls `Siofputs` to write a string to the output stream.

Prototype

```
fputs(strP, fs)
```

Parameters

-> strP	A pointer to the string to write.

-> `fs`	An output stream to which to write the string. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

Result Returns 0 on success and the value `EOF` on error.

getchar

Purpose Macro that calls `Siofgetc` to read the next character from the stdin input stream.

Prototype

```
getchar()
```

Result The next character from the input stream. The return value `EOF` indicates an error occurred.

gets

Purpose Macro that calls `Siogets` to read a string from the stdin input stream.

Prototype

```
gets(strP)
```

Parameters

<- strP	A pointer to the returned string.

Result A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns NULL.

Comments The returned string does not include a null terminator. You must ensure that the input line, if any, is sufficiently short to fit in the string.

printf

Purpose Macro that calls `Sioprintf` to write formatted output to the stdout output stream.

Prototype

```
printf(formatP, ...)
```

Parameters

-> `formatP`	A pointer to a format string that controls how subsequent arguments are converted for output.
-> ...	Zero or more parameters to be formatted as specified by the `formatP` string.

Result Returns the number of characters written out (not including the null terminator used to end output strings).

Comments This function internally calls StrVPrintF to do the formatting. See that function for details on which format specifications are supported. Returns a negative number if there is an error.

putc

Purpose Macro that calls `Siofputc` to write a character to the output stream.

Prototype

```
putc(c, fs)
```

Parameters

-> `c`	A character to write to the output stream.
-> `fs`	An output stream to which to write the character. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

Result The character that was written. If an error occurs, the value `EOF` is returned.

putchar

Purpose Macro that calls `Siofputc` to write a character to the stdout output stream.

Prototype

```
putchar(c)
```

Parameters

-> `c`	A character to write to the stdout output stream.

Result The character that was written. If an error occurs, the value `EOF` is returned.

puts

Purpose Macro that calls `Sioputs` to write a string to the output stream stdout.

Prototype

```
puts(strP)
```

Parameters

-> `strP`	A pointer to the string to write to stdout.

Result Returns a nonnegative value on success and the value `EOF` on error.

SioAddCommand

Purpose Adds a built-in command that is supplied by the standard IO provider application.

Prototype

```
void SioAddCommand (Char* cmdStr,
SioMainProcPtr cmdProcP)
```

Parameters

-> `cmdStr`	Pointer to a string that is the command name.
-> `cmdProcP`	Pointer to the command entry point function (the SioMain function).

Result Returns nothing.

Comments This routine is useful for registering a command that is inside the standard IO provider application instead of in its own database.

This routine must be used to test commands under the Simulator since it can't launch application databases.

sprintf

Purpose Macro that calls StrPrintF to write formatted output to the stdout output stream.

Prototype

```
sprintf (formatP, ...)
```

Parameters

-> `formatP`	A pointer to a format string that controls how subsequent arguments are converted for output.
-> ...	Zero or more parameters to be formatted as specified by the `formatP` string.

Result Returns the number of characters written out (not including the null terminator used to end output strings).

Comments See StrVPrintF for details on which format specifications are supported. Returns a negative number if there is an error.

system

Purpose Macro that calls `Siosystem` to execute another Stdio command.

Prototype

```
system(cmdStrP)
```

Parameters

-> `cmdStrP`	A pointer to a string containing the command line to execute.

Result Returns a value >= 0 on success or < 0 on failure.

Comments This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with

the name "Cmd-*cmdname*" where *cmdname* is the first word in the command string `cmdStrP`.

See Also `SioExecCommand`

vfprintf

Purpose Macro that calls `Siovfprintf` to write formatted output to the stdout output stream.

Prototype

```
vfprintf (fs, formatP, args)
```

Parameters

-> `fs`	An output stream to which to write the formatted output. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.
-> `formatP`	A pointer to a format string that controls how subsequent arguments are converted for output.
-> `args`	A pointer to a list of zero or more parameters to be formatted as specified by the `formatP` string.

Result Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments This function internally calls StrVPrintF to do the formatting. See that function for details on which format specifications are supported.

vsprintf

Purpose Macro that calls StrVPrintF to write formatted output to the stdout output stream.

Prototype

```
vsprintf (fs, formatP, args)
```

Parameters

-> `fs`	An output stream to which to write the formatted output. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.
-> `formatP`	A pointer to a format string that controls how subsequent arguments are converted for output.
-> `args`	A pointer to a list of zero or more parameters to be formatted as specified by the `y` string.

Result Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments See StrVPrintF for details on which format specifications are supported.

Standard IO Provider Functions

These functions are used by a standard IO provider application.

SioClearScreen

Purpose Clears the entire standard IO output field.

Prototype

```
void SioClearScreen(void)
```

SioExecCommand

Purpose Executes a command line.

Prototype

```
Int16 SioExecCommand (const Char* cmd)
```

Parameters

-> `cmd`	A pointer to a string containing the command line to execute.

Result Returns a value >= 0 on success or < 0 on failure.

Comments This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with the name "Cmd-*cmdname*" where *cmdname* is the first word in the command string `cmd`.

If you pass the string “help” or “?” for the `cmd` parameter, `SioExecCommand` causes a help string to be printed for each built-in command. It actually executes each built-in command, passing the string “?” as `argv[1]`. Each command should handle this argument by printing a help line.

The `SioExecCommand` function is faster than calling system to execute a command. However, `SioExecCommand` can be called only by the standard IO provider application, not the standard IO application.

See Also `system`

SioFree

Purpose Closes down the standard IO manager.

Prototype
```
Err SioFree(void)
```

Result Returns 0 on success.

SioHandleEvent

Purpose Handles an event in the form that contains the standard IO output field and scroll arrows if the event belongs to the text field or scroll arrows.

Prototype
```
Boolean SioHandleEvent (SysEventType* event)
```

Parameters

-> `event`	Pointer to an `EventType` structure.

Result Returns `true` if the event was handled and should not be processed by the application's own form event handler; returns `false` otherwise.

Comments This function must be called from the form event handler before it does its own processing with any of the objects unrelated to standard IO in the form.

SioInit

Purpose Initializes the standard IO manager.

Prototype
```
Err SioInit (UInt16 formID, UInt16 fieldID,
UInt16 scrollerID)
```

Parameters

-> `formID`	The ID of the form that contains the input/output field.
-> `fieldID`	The ID of the field to be used for input/output.

-> `scrollerID`	The ID of the scroller associated with the input/output form.

Result Returns 0 on success.

Application-Defined Function

You must supply this function in your stdio application.

SioMain

Purpose The main entry point for the stdio application.

Prototype

```
Int16 SioMain (UInt16 argc, Char* argv[])
```

Parameters

-> `argc`	The number of parameters passed on the command line.
-> `argv`	An array of character pointers, one for each parameter passed on the command line.

Result The return value from this routine is passed back to the system call that invoked it. Return 0 for no error.

44

String Manager

This chapter provides reference material for the string manager. The string manager API is declared in the header file `StringMgr.h`.

For more information, see the "Text" section in the *Palm OS Programmer's Companion*.

String Manager Functions

StrAToI

Purpose Convert a string to an integer.

Prototype

```
Int32 StrAToI (const Char* str)
```

Parameters

`str`	String to convert.

Result Returns the integer.

Comments Use this function instead of the standard `atoi` routine.

StrCaselessCompare

Purpose Compare two strings with case and accent insensitivity.

Prototype

```
Int16 StrCaselessCompare (const Char* s1,
const Char* s2)
```

Parameters

`s1`, `s2`	Two string pointers.

Result Returns 0 if the strings match.

Returns a positive number if `s1` > `s2`.

Returns a negative number if `s1` < `s2`.

Comments Use this function instead of the standard `stricmp` routine. Use it to find strings, or use it with `StrCompare` to sort strings. (See the comments in `StrCompare` for a example code.)

To support systems that use multi-byte character encodings, consider using `TxtCaselessCompare` instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCaselessCompare` can also return the length of the matching text.

See Also `StrNCaselessCompare`, `TxtCaselessCompare`, `StrCompare`, `StrNCompare`, `TxtCompare`

StrCat

Purpose Concatenate one string to another.

Prototype

```
Char* StrCat (Char* dst, const Char* src)
```

Parameters

`dst`	Destination string pointer.
`src`	Source string pointer.

Result Returns a pointer to the destination string.

Comments Use this function instead of the standard `strcat` routine.

StrChr

Purpose Look for a character within a string.

Prototype

```
Char* StrChr (const Char* str, WChar chr)
```

Parameters

`str`	String to search.
`chr`	Character to search for.

Result Returns a pointer to the first occurrence of character in `str`. Returns `NULL` if the character is not found.

Comments Use this function instead of the standard `strchr` routine.

This routine does not correctly find a '\0' character on Palm OS® version 1.0.

This function can handle both single-byte characters and multi-byte characters.

`StrChr` displays a non-fatal error message if `chr` is greater than `0xFF`.

See Also `StrStr`

StrCompare

Purpose Compare two strings.

Prototype

```
Int16 StrCompare (const Char* s1, const Char* s2)
```

Parameters

`s1, s2`	Two string pointers.

Result Returns 0 if the strings match.

Returns a positive number if `s1` sorts after `s2` alphabetically.

Returns a negative number if `s1` sorts before `s2` alphabetically.

Comments Use this function instead of the standard `strcmp` routine.

This function is case sensitive. Use it to sort strings but not to find them.

This function performs a character-by-character comparison of `s1` and `s2` and returns as soon as it finds two unequal characters. For example, if you are comparing the string "celery" with the string "Cauliflower," `StrCompare` returns that "celery" should appear before "Cauliflower" because it sorts the letter "c" before "C."

If you need to perform a true alphabetical sort, use `StrCaselessCompare` before using `StrCompare`, as in the following code:

```
Int16 result = StrCaselessCompare(a, b);
if (result == 0)
  result = StrCompare(a, b);
return(result);
```

To support systems that use multi-byte character encodings, consider using `TxtCompare` instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCompare` can also return the length of the matching text.

See Also `StrNCompare`, `TxtCompare`, `StrCaselessCompare`, `StrNCaselessCompare`, `TxtCaselessCompare`

StrCopy

Purpose Copy one string to another.

Prototype

```
Char* StrCopy (Char* dst, const Char* src)
```

Parameters

`dst, src`	Two string pointers.

Result Returns a pointer to the destination string.

Comments Use this function instead of the standard `strcpy` routine.

This function does not work properly with overlapping strings.

StrDelocalizeNumber

Purpose Delocalize a number passed in as a string. Convert the number from any localized notation to US notation (decimal point and thousandth comma). The current thousand and decimal separators have to be passed in.

Prototype

```
Char* StrDelocalizeNumber (Char* s,
Char thousandSeparator, Char decimalSeparator)
```

Parameters

`s` Pointer to the number as an ASCII string.

`thousandSeparator` Current thousand separator.

`decimalSeparator` Current decimal separator.

Result Returns a pointer to the changed number and modifies the string in `s`.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrLocalizeNumber`, `LocGetNumberSeparators`

StrIToA

Purpose Convert an integer to ASCII.

Prototype

```
Char* StrIToA (Char* s, Int32 i)
```

Parameters

`s` String pointer to store results.

`i` Integer to convert.

Result Returns a pointer to the result string.

See Also `StrAToI`, `StrIToH`

StrIToH

Purpose Convert an integer to hexadecimal ASCII.

Prototype
```
Char* StrIToH (Char* s, UInt32 i)
```

Parameters

`s`	String pointer to store results.
`i`	Integer to convert.

Result Returns the string pointer `s`.

See Also `StrIToA`

StrLen

Purpose Compute the length of a string.

Prototype
```
UInt16 StrLen (const Char* src)
```

Parameters

`src`	String pointer

Result Returns the length of the string in bytes.

Comments Use this function instead of the standard `strlen` routine.

This function returns the length of the string in bytes. On systems that support multi-byte characters, the number returned does not always equal the number of characters.

StrLocalizeNumber

Purpose Convert a number (passed in as a string) to localized format, using a specified thousands separator and decimal separator.

Prototype
```
Char* StrLocalizeNumber (Char* s,
Char thousandSeparator, Char decimalSeparator)
```

Parameters

`s`	Number ASCII string to localize.

`thousandSeparator`
Localized thousand separator.

`decimalSeparator`
Localized decimal separator.

Result Returns a pointer to the changed number. Converts the number string in `s` by replacing all occurrences of "," with `thousandSeparator` and all occurrences of "." with `decimalSeparator`.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrDelocalizeNumber`

StrNCaselessCompare

Purpose Compares two strings out to *n* characters with case and accent insensitivity.

Prototype

```
Int16 StrNCaselessCompare (const Char* s1,
const Char* s2, Int32 n)
```

Parameters

`s1`	Pointer to first string.
`s2`	Pointer to second string.
`n`	Length in bytes of the text to compare.

Result Returns 0 if the strings match.

Returns a positive number if `s1` > `s2`.

Returns a negative number if `s1` < `s2`.

Comments To support systems that use multi-byte character encodings, consider using `TxtCaselessCompare` instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCaselessCompare` can also return the length of the matching text.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also StrNCompare, StrCaselessCompare, TxtCaselessCompare, StrCompare, TxtCompare

StrNCat

Purpose Concatenates one string to another clipping the destination string to a maximum of *n* bytes (including the null character at the end).

Prototype

```
Char* StrNCat (Char* dst, const Char* src,
Int16 n)
```

Parameters

`dst`	Pointer to destination string.
`src`	Pointer to source string.
`n`	Maximum length in bytes for `dst`, including the terminating null character.

Result Returns a pointer to the destination string.

Comment This function differs from the standard C `strncat` function in these ways:

- `StrNCat` treats the parameter `n` as the maximum length in bytes for `dst`. That means it will copy at most `n - StrLen(dst) - 1` bytes from `src`. The standard C function always copies `n` bytes from `src` into `dst`. (It copies the entire `src` into `dst` if the length of `src` is less than `n`).
- If the length of the destination string reaches `n - 1`, `StrNCat` stops copying bytes from `src` and appends the terminating null character to `dst`. If the length of the destination string is already greater than or equal to `n - 1` before the copying begins, `StrNCat` does not copy any data from `src`.
- In the standard C function, if `src` is less than `n`, the entire `src` string is copied into `dst` and then the remaining space is filled with null characters. `StrNCat` does not fill the remaining space with null characters in released ROMs. In debug ROMs, `StrNCat` fills the remaining bytes with the value `0xFE`.

Compatibility Implemented only if 2.0 New Feature Set is present.

StrNCompare

Purpose Compare two strings out to *n* characters. This function is case and accent sensitive.

Prototype

```
Int16 StrNCompare (const Char* s1, const Char* s2,
UInt32 n)
```

Parameters

`s1`	Pointer to first string.
`s2`	Pointer to second string.
`n`	Length in bytes of text to compare.

Result Returns 0 if the strings match.

Returns a positive number if `s1` > `s2`.

Returns a negative number if `s1` < `s2`.

Comments To support systems that use multi-byte character encodings, consider using `TxtCompare` instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCompare` can also return the length of the matching text.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrCompare`, `TxtCompare`, `StrNCaselessCompare`, `StrCaselessCompare`, `TxtCaselessCompare`

StrNCopy

Purpose Copies up to *n* characters from a source string to the destination string. Terminates `dst` string at index *n*-1 if the source string length was *n*-1 or less.

Prototype

```
Char* StrNCopy (Char* dst, const Char* src,
Int16 n)
```

Parameters

`dst`	Destination string.
`src`	Source string.
`n`	Maximum number of bytes to copy from `src` string.

Result Returns nothing.

Comments On systems with multi-byte character encodings, this function makes sure that it does not copy part of a multi-byte character. If the `n`th byte of `src` contains the high-order or middle byte of a multi-byte character, `StrNCopy` backs up in `dst` until the byte after the end of the previous character, and replaces the remaining bytes (up to `n`-1) with nulls.

Compatibility Implemented only if 2.0 New Feature Set is present.

StrPrintF

Purpose Implements a subset of the ANSI C `sprintf` call, which writes formatted output to a string.

Prototype

```
Int16 StrPrintF (Char* s,
const Char* formatStr, ...)
```

Parameters

`s`	Pointer to a string where the results are written.
`formatStr`	Pointer to the format specification string.

...	Zero or more arguments to be formatted as specified by `formatStr`.

Result Number of characters written to destination string. Returns a negative number if there is an error.

Comments This function internally calls `StrVPrintF` to do the formatting. See that function for details on which format specifications are supported.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrVPrintF`

StrStr

Purpose Look for a substring within a string.

Prototype

```
Char* StrStr (const Char* str, const Char* token)
```

Parameters

`str`	String to search.
`token`	String to search for.

Result Returns a pointer to the first occurrence of `token` in `str` or `NULL` if not found.

Comments Use this function instead of the standard `strstr` routine.

On systems with multi-byte character encodings, this function makes sure that it does not match only part of a multi-byte character. If the matching strings begins at an inter-character boundary, then this function returns `NULL`.

See Also `StrChr`

StrToLower

Purpose Convert all the characters in a string to lowercase.

Prototype

```
Char* StrToLower (Char* dst, const Char* src)
```

Parameters

`dst`, `src`	Two string pointers.

Result Returns a pointer to the destination string.

Compatibility Prior to Palm OS version 3.5, this function only converted accented characters on Japanese devices. On Palm OS version 3.5 and higher, all characters are appropriately lowercased, including accented characters on Latin devices.

StrVPrintF

Purpose Implements a subset of the ANSI C `vsprintf` call, which writes formatted output to a string.

Prototype

```
Int16 StrVPrintF (Char* s, const Char* formatStr,
_Palm_va_list argParam)
```

Parameters

`s`	Pointer to a string where the results are written. This string is always terminated by a null terminator.
`formatStr`	Pointer to the format specification string.
`argParam`	Pointer to a list of zero or more parameters to be formatted as specified by the `formatStr` string.

Result Number of characters written to destination string, not including the null terminator. Returns a negative number if there is an error.

Comments Like the C `vsprintf` function, this function is designed to be called by your own function that takes a variable number of arguments and passes them to this function. For details on how to use it, see

"Using the StrVPrintF Function" on page 125 in *Palm OS Programmer's Companion*, or refer to `vsprintf` in a standard C reference book.

Currently, only the conversion specifications `%d`, `%i`, `%u`, `%x`, `%s`, and `%c` are implemented by `StrVPrintF` (and related functions). Optional modifiers that are supported include: `+`, `-`, <space>, `*`, <digits>, `h` and `l` (long). Following is a brief description of how these format specifications work (see a C book for more details).

Each conversion specification begins with the % character. Following the % character, there may be one or more of the characters list in Table 44.1, in sequence.

Table 44.1 StrVPrintF Format Specification

Character	Description

+	Specifies that a sign always be placed before a number produced by a signed conversion. A + overrides a space if both are used. Example:
	`StrPrintF(s,"%+d  %+d",4,-5);`
	Output to `s`:
	`+4  -5`

-	Specifies that the printed value is left justified within the field width allowed for it. Example:
	`StrPrintF(s,"%5d%-5d%d",6,9,8);`
	Output to `s`:
	`    69    8`

<space>	Specifies that a minus sign always be placed before a negative number and a space before a positive number. Example:
	`StrPrintF(s,"% d  % d",4,-5);`
	Output to `s`:
	` 4  -5`

Table 44.1 StrVPrintF Format Specification *(continued)*

Character	Description

*	Indicates that the next argument (must be an integer) in the list specifies the field width. In this case, the argument following that one is used for the value of this field. Example:
	`StrPrintF(s,"%*d%d",4,8,5);`
	Output to `s`:
	`8  5`

<number>	Specifies a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left justified flag is also specified) to fill out the field width. Example:
	`StrPrintF(s,"%d%5d",4,3);`
	Output to `s`:
	`4    3`

`h`	Specifies that the following d, i, u, or x conversion corresponds to a `short` or `unsigned short` argument. Example:
	`StrPrintF(s,"%hd",401);`
	Output to `s`:
	`401`

`l` or `L`	Specifies that the following d, i, u, or x conversion corresponds to a `long` or `unsigned long`
	`StrPrintF(s,"%ld",999999999);`
	Output to `s`:
	`999999999`

<character>	A character that indicates the type of conversion to be performed. The supported conversion characters include:

	`d` or `i` — A signed integer argument is converted to decimal notation. Example:
	`StrPrintF(s,"%d  %d",4,-4);`
	Output to `s`:
	`4  -4`

Table 44.1 StrVPrintF Format Specification *(continued)*

Character	Description	

	`u`	An unsigned integer argument is converted to decimal notation. Example:
		`StrPrintF(s,"%u  %u",4,-4);`
		Output to `s`:
		`4  65532`

	`x`	An integer argument is converted to hexadecimal notation. Example:
		`StrPrintF(s,"%x",125);`
		Output to `s`:
		`0000007D`

	`s`	A string (`char *`) argument is copied to the destination string. Example:
		`StrPrintF(s,"ABC%s","DEF");`
		Output to `s`:
		`ABCDEF`

	`c`	A single character (`int`) argument is copied to the destination string. Example:
		`StrPrintF(s,"Telephone%c",'s');`
		Output to `s`:
		`Telephones`

	`%`	A % character is copied to the destination string. Example:
		`StrPrintF(s,"%%");`
		Output to `s`:
		`%`

Example Here's an example of how to use this call:

```
#include <stdarg.h>
void MyPrintF(Char* s, Char* formatStr, ...)
{
  va_list args;
  Char text[0x100];
```

```
    va_start(args, formatStr);
    StrVPrintF(text, formatStr, args);
    va_end(args);
    MyPutS(text);
  }
```

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrPrintF`, Using the StrVPrintF Function

45

System Event Manager

This chapter describes functions available in the system event manager. The system event manager API is declared in the header files `Event.h` and `SysEvtMgr.h`.

For more information on the system event manager, see the chapter "Event Loop" in the *Palm OS Programmer's Companion*. The reference for specific events sent by the system are documented in Palm OS Events.

System Event Manager Data Structures

The following system event manager data structures are documented elsewhere:

- eventsEnumfs
- EventType
- EventPtr

System Event Manager Functions

EvtAddEventToQueue

Purpose Add an event to the event queue.

Prototype

```
void EvtAddEventToQueue (const EventType *event)
```

Parameters

`event`	Pointer to the structure that contains the event.

Result Returns nothing.

EvtAddUniqueEventToQueue

Purpose Look for an event in the event queue of the same event type and ID (if specified). The routine replaces it with the new event, if found.

If no existing event is found, the new event is added.

If an existing event is found, the routine proceeds as follows:

- If `inPlace` is `true`, the existing event is replaced with the new event.
- If `inPlace` is `false`, the existing event is removed and the new event will be added to the end.

Prototype

```
void EvtAddUniqueEventToQueue
(const EventType *eventP, UInt32 id,
Boolean inPlace)
```

Parameters

`eventP`	Pointer to the structure that contains the event
`id`	ID of the event. 0 means match only on the type.
`inPlace`	If `true`, existing event are replaced. If `false`, existing event is deleted and new event added to end of queue.

Result Returns nothing.

Compatibility Implemented only if 2.0 New Feature Set is present.

EvtCopyEvent

Purpose Copy an event.

Prototype

```
void EvtCopyEvent (const EventType *source,
EventType *dest)
```

Parameters

`source`	Pointer to the structure containing the event to copy.

`dest`	Pointer to the structure to copy the event to.

Result Returns nothing.

EvtDequeuePenPoint

Purpose Get the next pen point out of the pen queue. This function is called by recognizers.

Prototype

```
Err EvtDequeuePenPoint (PointType* retP)
```

Parameters

`retP`	Return point.

Result Always returns 0.

Comments Called by a recognizer that wishes to extract the points of a stroke. Returns the point (-1, -1) at the end of a stroke.

Before calling this routine, you must call EvtDequeuePenStrokeInfo.

See Also `EvtDequeuePenStrokeInfo`

EvtDequeuePenStrokeInfo

Purpose Initiate the extraction of a stroke from the pen queue.

Prototype

```
Err EvtDequeuePenStrokeInfo (PointType* startPtP,
PointType* endPtP)
```

Parameters

`startPtP`	Start point returned here.
`endPtP`	End point returned here.

Result Always returns 0.

Comments Called by the system function `EvtGetSysEvent` when a `penUp` event is being generated. This routine must be called before EvtDequeuePenPoint is called.

Subsequent calls to EvtDequeuePenPoint return points at the starting point in the stroke and including the end point. After the end point is returned, the next call to EvtDequeuePenPoint returns the point -1, -1.

See Also `EvtDequeuePenPoint`

EvtEnableGraffiti

Purpose Set Graffiti® enabled or disabled.

Prototype

```
void EvtEnableGraffiti (Boolean enable)
```

Parameters

`enable`	`true` to enable Graffiti, `false` to disable Graffiti.

Result Returns nothing.

EvtEnqueueKey

Purpose Place keys into the key queue.

Prototype

```
Err EvtEnqueueKey (WChar ascii, UInt16 keycode,
UInt16 modifiers)
```

Parameters

`ascii`	ASCII code of key.
`keycode`	Virtual key code of key.
`modifiers`	Modifiers for key event.

Result Returns 0 if successful, or `evtErrParamErr` if an error occurs.

Comments Called by the keyboard interrupt routine and the Graffiti and soft key recognizers. Note that because both interrupt- and noninterrupt-level code can post keys into the queue, this routine disables interrupts while the queue header is being modified.

Most keys in the queue take only 1 byte if they have no modifiers and no virtual key code, and are 8-bit ASCII. If a key event in the

queue has modifiers or is a non-standard ASCII code, it takes up to 7 bytes of storage and has the following format:

evtKeyStringEscape	1 byte
ASCII code	2 bytes
virtual key code	2 bytes
modifiers	2 bytes

EvtEventAvail

Purpose Return `true` if an event is available.

Prototype

```
Boolean EvtEventAvail (void)
```

Parameters None

Result Returns `true` if an event is available, `false` otherwise.

Compatibility Implemented only if 2.0 New Feature Set is present.

EvtFlushKeyQueue

Purpose Flush all keys out of the key queue.

Prototype

```
Err EvtFlushKeyQueue (void)
```

Parameters None.

Result Always returns 0.

Comments Called by the system function `EvtSetPenQueuePtr`.

EvtFlushNextPenStroke

Purpose Flush the next stroke out of the pen queue.

Prototype
```
Err EvtFlushNextPenStroke (void)
```

Parameters None

Result Always returns 0.

Comments Called by recognizers that need only the start and end points of a stroke. If a stroke has already been partially dequeued (by `EvtDequeuePenStrokeInfo`) this routine finishes the stroke dequeueing. Otherwise, this routine flushes the next stroke in the queue.

See Also `EvtDequeuePenPoint`

EvtFlushPenQueue

Purpose Flush all points out of the pen queue.

Prototype
```
Err EvtFlushPenQueue (void)
```

Parameters None

Result Always returns 0.

Comments Called by the system function `EvtSetKeyQueuePtr`.

See Also `EvtPenQueueSize`

EvtGetEvent

Purpose Return the next available event.

Prototype

```
void EvtGetEvent (EventType *event, Int32 timeout)
```

Parameters

`event`	Pointer to the structure to hold the event returned.
`timeout`	Maximum number of ticks to wait before an event is returned (-1 means wait indefinitely).

Comments Pass `timeout`= -1 in most instances. When running on the device, this makes the CPU go into doze mode until the user provides input. For applications that do animation, pass `timeout` >= 0.

Result Returns nothing.

EvtGetPen

Purpose Return the current status of the pen.

Prototype

```
void EvtGetPen (Int16 *pScreenX, Int16 *pScreenY,
Boolean *pPenDown)
```

Parameters

`pScreenX`	x location relative to display.
`pScreenY`	y location relative to display.
`pPenDown`	`true` or `false`.

Result Returns nothing.

Comments Called by various UI routines.

See Also `KeyCurrentState`

EvtGetPenBtnList

Purpose Return a pointer to the silk-screen button array.

Prototype
```
const PenBtnInfoType* EvtGetPenBtnList
(UInt16* numButtons)
```

Parameters

`numButtons`	Pointer to the variable to contain the number of buttons in the array.

Result Returns a pointer to the array.

Comments The array returned contains the bounds of each silk-screened button and the ASCII code and modifiers byte to generate for each button.

See Also `EvtProcessSoftKeyStroke`

EvtGetSilkscreenAreaList

Purpose Returns a pointer to the silk screen area array. This array contains the bounds of each silk screen area.

Prototype
```
*const SilkscreenAreaType*
EvtGetSilkscreenAreaList(UInt16* numAreas)
```

Parameters

`numAreas`	pointer to area count variable

Result returns a pointer to the array and the number of elements in the array.

EvtKeydownIsVirtual

Purpose Macro that indicates if `eventP` is a pointer to a virtual character key down event.

Prototype

```
#define EvtKeydownIsVirtual(eventP)
(((eventP)->data.keyDown.modifiers &
virtualKeyMask) != 0)
```

Parameters

`eventP`	pointer to the structure that contains the event.

Result Returns `true` if the character is a letter in an alphabet or a numeric digit, `false` otherwise.

Comments The macro assumes that the caller has already determined the event is a key down. With earlier versions of the OS, use `TxtGlueCharIsVirtual` in the PalmOSGlue Library.

Compatibility Implemented in the Palm OS® 3.5 SDK, but will work on older devices (at least on 3.0, perhaps on 2.0 and 1.0.)

See Also `TxtGlueCharIsVirtual`

EvtKeyQueueEmpty

Purpose Return `true` if the key queue is currently empty.

Prototype

```
Boolean EvtKeyQueueEmpty (void)
```

Parameters None.

Result Returns `true` if the key queue is currently empty, otherwise returns `false`.

Comments Usually called by the key manager to determine if it should enqueue auto-repeat keys.

EvtKeyQueueSize

Purpose Return the size of the current key queue in bytes.

Prototype
```
UInt32 EvtKeyQueueSize (void)
```

Parameters None.

Result Returns size of queue in bytes.

Comments Called by applications that wish to see how large the current key queue is.

EvtPenQueueSize

Purpose Return the size of the current pen queue in bytes.

Prototype
```
UInt32 EvtPenQueueSize (void)
```

Parameters None.

Result Returns size of queue in bytes.

Comments Call this function to see how large the current pen queue is.

EvtProcessSoftKeyStroke

Purpose Translate a stroke in the system area of the digitizer and enqueue the appropriate key events in to the key queue.

Prototype
```
Err EvtProcessSoftKeyStroke(PointType* startPtP,
PointType* endPtP)
```

Parameters

`startPtP`	Start point of stroke.

`endPtP`	End point of stroke.

Result Returns 0 if recognized, -1 if not recognized.

See Also `EvtGetPenBtnList`, `GrfProcessStroke`

EvtResetAutoOffTimer

Purpose Reset the auto-off timer to assure that the device doesn't automatically power off during a long operation without user input (for example, serial port activity).

Prototype

```
Err EvtResetAutoOffTimer (void)
```

Parameters None.

Result Always returns 0.

Comments Called by the serial link manager; can be called periodically by other managers. `EvtResetAutoOffTimer` just resets the timer, while `EvtSetAutoOffTimer` allows you to specify a time.

See Also `SysSetAutoOffTime` `EvtSetAutoOffTimer`

EvtSetAutoOffTimer

Purpose `EvtSetAutoOffTimer` can be called periodically by other managers to reset the auto-off timer.

Prototype

```
Err EvtSetAutoOffTimer(EvtSetAutoOffCmd cmd,
UInt16 timeoutSecs)
```

Parameters

`cmd`	One of the defined commands.
`timeout`	A new timeout in seconds, ignored for the 'reset' command.

Result Returns 0 if no error.

Comments This assures that the device doesn't automatically power off during a long operation that doesn't have user input (like a lot of serial port activity, for example). It is also used to manage the auto-off timer in general.

These commands are currently defined:

SetAtLeast	Turn off in at least xxx seconds
SetExactly:	Set the timer to turn off in xxx seconds
SetAtMost:	Set the device to turn off in <= xxx seconds
SetDefault:	Change default auto-off timeout to xxx seconds
ResetTimer:	Reset the auto off timer.

NOTE: This functionality is only available in Palm OS 3.5 and later.

See Also `EvtResetAutoOffTimer` `SysSetAutoOffTime`

EvtSetNullEventTick

Purpose Set the tick when a null event is due, unless one is due sooner.

Prototype

```
Boolean EvtSetNullEventTick(UInt32 tick)
```

Parameters

`tick`	the tick when a null event should occur.

Result Returns true if null tick count setting changed.

EvtSysEventAvail

Purpose Return `true` if a low-level system event (such as a pen or key event) is available.

Prototype

```
Boolean EvtSysEventAvail (Boolean ignorePenUps)
```

Parameters

`ignorePenUps`	If `true`, this routine ignores pen-up events when determining if there are any system events available.

Result Returns `true` if a system event is available.

Comment Call EvtEventAvail to determine whether high-level software events are available.

Compatibility Implemented only if 2.0 New Feature Set is present.

EvtWakeup

Purpose Force the event manager to wake up and send a `nilEvent` to the current application.

Prototype

```
Err EvtWakeup (void)
```

Parameters None.

Result Always returns 0.

Comments Called by interrupt routines, like the sound manager and alarm manager.

46

System Manager

This chapter provides reference material for the system manager. The system manager API is declared in the header files `SystemMgr.h` and `SysUtils.h`.

For more information on the system manager, see the chapters "Application Startup and Stop" and "Palm System Features" in the *Palm OS Programmer's Companion*.

System Functions

SysAppLaunch

Purpose

Open an application from a specified database and card, with the appropriate launch flags. Generally used to launch an application as a subroutine of the caller.

Prototype

```
Err SysAppLaunch (UInt16 cardNo, LocalID dbID,
UInt16 launchFlags, UInt16 cmd, MemPtr cmdPBP,
UInt32* resultP)
```

Parameters

`cardNo`, `dbID`	`cardNo` and `dbID` identify the application.
`launchFlags`	Set to 0.
`cmd`	Launch code.
`cmdPBP`	Launch code parameter block.
`resultP`	Pointer to what's returned by the application's `PilotMain` routine.

Result

Returns 0 if no error, or one of `sysErrParamErr`, `memErrNotEnoughSpace`, `sysErrOutOfOwnerIDs`.

Comments Launching an application with all launch bits cleared makes the application a subroutine call from the point of view of the caller.

Do not use this function to open the system-supplied Application Launcher application. If another application has replaced the default launcher with one of its own, this function will open the custom launcher instead of the system-supplied one. To open the system-supplied launcher reliably, enqueue a `keyDownEvent` that contains a `launchChr`, as shown in the section "Application Launcher" of the user interface chapter in the *Palm OS Programmer's Companion*.

If the launch flag `sysAppLaunchFlagNewThread` is set, and you are passing a parameter block (the `cmdPBP` parameter), you must set the owner of the parameter block chunk to the operating system. To do this, and for more information, see `MemPtrSetOwner`. If the parameter block structure contains references by pointer or handle to any other chunks, you also must set the owner of those chunks by using `MemHandleSetOwner` or `MemPtrSetOwner`.

NOTE: For important information regarding the correct use of this function, see the "Application Startup and Stop" chapter in the *Palm OS Programmer's Companion*.

See Also `SysBroadcastActionCode`, `SysUIAppSwitch`, `SysCurAppDatabase`

SysBatteryInfo

Purpose Retrieve settings for the batteries. Set `set` to `false` to retrieve battery settings. (Applications should ***not*** change any of the settings).

Use this function only to **retrieve** settings!

Prototype

```
UInt16 SysBatteryInfo (Boolean set,
UInt16* warnThresholdP,
UInt16* criticalThresholdP, UInt16* maxTicksP,
SysBatteryKind* kindP, Boolean* pluggedIn
UInt8* percentP)
```

Parameters

`set`	If `false`, parameters with non-`NULL` pointers are retrieved. Never set this parameter to `true`.
`warnThresholdP`	Pointer to battery voltage warning threshold in volts*100, or `NULL`.
`criticalThresholdP`	Pointer to the battery voltage critical threshold in volts*100, or `NULL`.
`maxTicksP`	Pointer to the battery timeout, or `NULL`.
`kindP`	Pointer to the battery kind, or `NULL`.
`pluggedIn`	Pointer to `pluggedIn` return value, or `NULL`.
`percentP`	Percentage of power remaining in the battery.

Result

Returns the current battery voltage in volts*100.

Comments

Call this function to make sure an upcoming activity won't be interrupted by a low battery warning.

`warnThresholdP` and `maxTicksP` are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a `lowBatteryChr` key event is put on the queue. Normally, applications call `SysHandleEvent` which calls `SysBatteryDialog` in response to this event.

`criticalThresholdP` is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn't turn on until battery voltage is above it again.

Compatibility

This function was revised for Palm OS® 3.0. In Palm OS 3.0, the `percentP` parameter was added. This enhancement is implemented only if 3.0 New Feature Set is present.

See Also

`SysBatteryInfoV20`

SysBatteryInfoV20

Purpose Retrieve settings for the batteries. Set to `false` to retrieve battery settings. (Applications should ***not*** change any of the settings).

Use this function only to **retrieve** settings!

Prototype

```
UInt16 SysBatteryInfoV20 (Boolean set,
UInt16* warnThresholdP,
UInt16* criticalThresholdP, UInt16* maxTicksP,
SysBatteryKind* kindP, Boolean* pluggedIn)
```

Parameters

`set`	If `false`, parameters with non-`NULL` pointers are retrieved. Never set this parameter to `true`.
`warnThresholdP`	Pointer to battery voltage warning threshold in volts*100, or `NULL`.
`criticalThresholdP`	Pointer to the battery voltage critical threshold in volts*100, or `NULL`.
`maxTicksP`	Pointer to the battery timeout, or `NULL`.
`kindP`	Pointer to the battery kind, or `NULL`.
`pluggedIn`	Pointer to `pluggedIn` return value, or `NULL`.

Result Returns the current battery voltage in volts*100.

Comments Call this function to make sure an upcoming activity won't be interrupted by a low battery warning.

`warnThresholdP` and `maxTicksP` are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a `lowBatteryChr` key event is put on the queue. Normally, applications call `SysHandleEvent` which calls `SysBatteryDialog` in response to this event.

`criticalThresholdP` is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn't turn on until battery voltage is above it again.

Compatibility This function corresponds to the Palm OS 2.0 version of `SysBatteryInfo`. Implemented only if 3.0 New Feature Set is present.

See Also `SysBatteryInfo`

SysBinarySearch

Purpose Search elements in a sorted array for the specified data according to the specified comparison function.

Prototype

```
Boolean SysBinarySearch (void const *baseP,
const UInt16 numOfElements, const Int16 width,
SearchFuncPtr searchF, void const *searchData,
const Int32 other, Int32* position,
const Boolean findFirst)
```

Parameters

`baseP`	Base pointer to an array of elements
`numOfElements`	Number of elements to search, starting at 0 to `numOfElements` -1. Must be greater than 0.
`width`	Width of an element comparison function.
`searchF`	Search function.
`searchData`	Data to search for. This data is passed to the `searchF` function.
`other`	Data to be passed as the third parameter (the `other` parameter) to the comparison function.
`position`	Pointer to the position result.

`findFirst`	If set to `true`, the first matching element is returned. Use this parameter if the array contains duplicate entries to ensure that the first such entry will be the one returned.

Result

Returns `true` if an exact match was found. In this case, `position` points to the element number where the data was found.

Returns `false` if an exact match was not found. If `false` is returned, `position` points to the element number where the data should be inserted if it was to be added to the array in sorted order.

Comments

The array must be sorted in ascending order prior to the search. Use `SysInsertionSort` or `SysQSort` to sort the array.

The search starts at element 0 and ends at element (`numOfElements` - 1).

The search function's (`searchF`) prototype is:

```
Int16 _searchF (void const *searchData,
void const *arrayData, Int32 other);
```

The first parameter is the data for which to search, the second parameter is a pointer to an element in the array, and the third parameter is any other necessary data.

The function returns:

- > 0 if the search data is greater than the element
- < 0 if the search data is less than the element
- 0 if the search data is the same as the element

Compatibility

Implemented only if 2.0 New Feature Set is present.

SysBroadcastActionCode

Purpose Send the specified action code (launch code) and parameter block to the latest version of every UI application.

Prototype

```
Err SysBroadcastActionCode (UInt16 cmd,
MemPtr cmdPBP)
```

Parameters

`cmd`	Action code to send.
`cmdPBP`	Action code parameter block to send.

Result Returns 0 if no error, or one of the following errors: `sysErrParamErr, memErrNotEnoughSpace, sysErrOutOfOwnerIDs.`

See Also `SysAppLaunch`, Chapter 3, "Application Startup and Stop." of the *Palm OS Programmer's Companion*

SysCopyStringResource

Purpose Copy a resource string to a passed string.

Prototype

```
void SysCopyStringResource (Char* string,
Int16 theID)
```

Parameters

`string`	String to copy the resource string to.
`theID`	Resource string ID.

Result Stores a copy of the resource string in `string`.

SysCreateDataBaseList

Purpose Generate a list of databases found on the memory cards matching a specific type and return the result. If `lookupName` is `true` then a name in a `tAIN` resource is used instead of the database's name and

the list is sorted. Only the last version of a database is returned. Databases with multiple versions are listed only once.

Prototype

```
Boolean SysCreateDataBaseList (UInt32 type,
UInt32 creator, UInt16* dbCount, MemHandle *dbIDs,
Boolean lookupName)
```

Parameters

`type`	Type of database to find (0 for wildcard).
`creator`	Creator of database to find (0 for wildcard).
`dbCount`	Pointer to contain count of matching databases.
`dbIDs`	Pointer to handle allocated to contain the database list.
`lookupName`	Use `tAIN` names and sort the list.

Result

Returns `false` if no databases were found, `true` if databases were found. `dbCount` is updated to the number of databases found; `dbIDs` is updated to the list of matching databases found.

Compatibility

Implemented only if 2.0 New Feature Set is present.

SysCreatePanelList

Purpose

Generate a list of panels found on the memory cards and return the result. Multiple versions of a panel are listed once.

Prototype

```
Boolean SysCreatePanelList (UInt16* panelCount,
MemHandle *panelIDs)
```

Parameters

`panelCount`	Pointer to set to the number of panels.
`panelIDs`	Pointer to handle containing a list of panels.

Result

Returns `false` if no panels were found, `true` if panels were found. `panelCount` is updated to the number of panels found; `panelIDs` is updated to the IDs of panels found.

Compatibility

Implemented only if 2.0 New Feature Set is present.

SysCurAppDatabase

Purpose Return the card number and database ID of the current application's resource database.

Prototype

```
Err SysCurAppDatabase (UInt16* cardNoP,
LocalID* dbIDP)
```

Parameters

`cardNoP`	Pointer to the card number; 0 or 1.
`dbIDP`	Pointer to the database ID.

Result Returns 0 if no error, or `SysErrParamErr` if an error occurs.

See Also `SysAppLaunch, SysUIAppSwitch`

SysErrString

Purpose Returns text to describe an error number. This routine looks up the textual description of a system error number in the appropriate List resource and creates a string that can be used to display that error.

The actual string will be of the form: "<error message> (XXXX)" where XXXX is the hexadecimal error number.

This routine looks for a resource of type 'tstl' and resource ID of (err>>8). It then grabs the string at index (err & 0x00FF) out of that resource.

The first string in the resource is called index #1 by Constructor, NOT #0. For example, an error code of 0x0101 will fetch the first string in the resource.

Prototype

```
Char* SysErrString (Err err, Char* strP,
UInt16 maxLen)
```

Parameters

`err`	Error number
`strP`	Pointer to space to form the string

`maxLen`	Size of `strP` buffer.

Result Stores the error number string.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysFormPointerArrayToStrings

Purpose Form an array of pointers to strings in a block. Useful for setting the items of a list.

Prototype

```
MemHandle SysFormPointerArrayToStrings (Char* c,
Int16 stringCount)
```

Parameters

`c`	Pointer to packed block of strings, each terminated by `NULL`.
`stringCount`	Count of strings in block.

Result Unlocked handle to allocated array of pointers to the strings in the passed block. The returned array points to the strings in the passed packed block.

SysGetOSVersionString

Purpose Return the version number of the Palm operating system.

Prototype

```
Char* SysGetOSVersionString()
```

Parameters None.

Result Returns a string such as "v. 3.0."

Comments You must free the returned string using the `MemPtrFree` function.

Compatibility Implemented only if 3.0 New Feature Set is present.

SysGetROMToken

Purpose Return from ROM a value specified by token.

Prototype

```
Err SysGetROMToken (UInt16 cardNo, UInt32 token,
UInt8 **dataP, UInt16 *sizeP)
```

Parameters

-> `cardNo`	The card on which the ROM to be queried resides. Currently, no Palm hardware provides multiple cards, so this value must be 0.
-> `token`	The value to retrieve, as specified by one of the following tokens: `sysROMTokenSnum` The serial number of the ROM, expressed as a text string with no null terminator.
<- `dataP`	Pointer to a text buffer that holds the requested value when the function returns.
<- `sizeP`	The number of bytes in the `dataP` buffer.

Result Returns the requested value if no error, or an error code if an error occurs. If this function returns an error, or if the returned pointer to the buffer is `NULL`, or if the first byte of the text buffer is `0xFF`, then no serial number is available.

Comments The serial number is shown to the user in the Application Launcher, along with a checksum digit you can use to validate input when your users read the ID from their device and type it in or tell it to someone else.

Compatibility Implemented only if 3.0 New Feature Set is present. Serial numbers are available only on flash ROM-based units.

See Also "Retrieving the ROM Serial Number" section in the *Palm OS Programmer's Companion* shows how to retrieve the ROM serial number and calculate its associated checksum.

SysGetStackInfo

Purpose Return the start and end of the current thread's stack.

Prototype

```
Boolean SysGetStackInfo (MemPtr *startPP,
MemPtr *endPP)
```

Parameters

`startPP`	Upon return, points to the start of the stack.
`endPP`	Upon return, points to the end of the stack.

Result Returns `true` if the stack has not overflowed, that is, the value of the stack overflow address has not been changed. Returns `false` if the stack overflow value has been overwritten, meaning that a stack overflow has occurred.

Compatibility Implemented only if 3.0 New Feature Set is present.

SysGetTrapAddress

Purpose Return the address of a function given its system trap.

Prototype

```
void* SysGetTrapAddress (UInt16 trapNum)
```

Parameters

-> `trapNum`	One of the routine selectors defined in `SysTraps.h` (`sysTrap...`) or `CoreTraps.h` on Palm OS version 3.5 and higher.

Result Returns the address of the corresponding function. Returns `NULL` if an invalid routine selector is passed.

Comments Use this function for performance reasons. You can then use the address it returns to call the function without having to go through the trap dispatch table. This function is mostly useful for optimizing the performance of functions called in a tight loop.

The Palm OS trap dispatch mechanism allows the trap table entries to be modified at any time, either as the result of a system update or a hack. For this reason, it's important to call this function

immediately before entering the tight loop. If the trap address changes in between when you call `SysGetTrapAddress` and you use the address, the wrong function will be called.

SysGraffitiReferenceDialog

Purpose Pop up the Graffiti® Reference Dialog.

Prototype

```
void SysGraffitiReferenceDialog
(ReferenceType referenceType)
```

Parameters

`referenceType`	Which reference to display. See `GraffitiReference.h` for more information.

Result Nothing returned.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysGremlins

Purpose Query the Gremlins facility. You pass a selector for a function and parameters for that function. Gremlins performs the function call and returns the result.

Prototype

```
UInt32 SysGremlins (GremlinFunctionType selector,
GremlinParamsType *params)
```

Parameters

`selector`	The selector for a function to pass to Gremlins.
`params`	Pointer to a parameter block used to pass parameters to the function specified by `selector`.

Result Returns the result of the function performed in Gremlins.

Comments Currently, only one selector is defined, `GremlinIsOn`, which takes no parameters. `GremlinIsOn` returns 0 if Gremlins is not running, non-zero if it is running.

Currently, non-zero values are returned only from the version of Gremlins in the Palm OS emulator. The Gremlins running in the Simulator on a Macintosh and over the serial line via the Palm Debugger return zero for `GremlinIsOn`.

Use this function if you need to alter the application's behavior when Gremlins is running. For example, the debug 3.0 ROM refuses to bring up the digitizer panel when Gremlins is running under the emulator.

Compatibility Implemented only if 3.0 New Feature Set is present.

In Palm OS 3.2 and later, `SysGremlins` is replaced by the functions defined in the file `HostControl.h`. Specifically, the one selector supported by `SysGremlins` is replaced with the function `HostGremlinIsRunning`. For backward compatibility, `SysGremlins` is mapped to `HostGremlinIsRunning`.

SysHandleEvent

Purpose Handle defaults for system events such as hard and soft key presses.

Prototype

```
Boolean SysHandleEvent (EventPtr eventP)
```

Parameters

`eventP`	Pointer to an event.

Result Returns `true` if the system handled the event.

Comments Applications should call this routine immediately after calling `EvtGetEvent` unless they want to override the default system behavior. However, overriding the default system behavior is almost never appropriate for an application.

See Also `EvtProcessSoftKeyStroke`, `KeyRates`

SysInsertionSort

Purpose Sort elements in an array according to the passed comparison function.

Prototype

```
void SysInsertionSort (void* baseP,
Int16 numOfElements, Int16 width,
const CmpFuncPtr comparF, const Int32 other)
```

Parameters

`baseP`	Base pointer to an array of elements.
`numOfElements`	Number of elements to sort (must be at least 2).
`width`	Width of an element.
`comparF`	Comparison function (see Comments).
`other`	Other data passed to the comparison function.

Result Returns nothing.

Comments Only elements which are out of order move. Moved elements are moved to the end of the range of equal elements. If a large amount of elements are being sorted, try to use the quick sort (see `SysQSort`).

This is the insertion sort algorithm: Starting with the second element, each element is compared to the preceding element. Each element not greater than the last is inserted into sorted position within those already sorted. A binary search for the insertion point is performed. A moved element is inserted after any other equal elements.

In Palm OS 2.0 and later, `DmComparF` has 6 parameters.

These parameters allow a Palm OS application to pass more information to the system than before, most noticeably the record (and all associated information) which allows sorting by unique ID, so that the Palm OS device and the desktop always match.

The revised callback is used by new sorting routines (and can be used the same way by your application):

```
typedef Int16 DmComparF (void *, void *, Int16
other, SortRecordInfoPtr, SortRecordInfoPtr,
MemHandle appInfoH);
```

As a rule, this change in the number of arguments doesn't cause problems when a 1.0 application is run on a 2.0 or later device, because the system only pulls the arguments from the stack that are there.

Note, however, that some optimized applications built with tools other than Metrowerks CodeWarrior for Palm OS may have problems as a result of the change in arguments when running on a 2.0 or later device.

The 2.0 comparison function (`comparF`) has this prototype:

```
Int comparF (VoidPtr, VoidPtr, Long other);
```

The 1.0 comparison function (`comparF`) had this prototype:

```
Int comparF (BytePtr A, BytePtr B, Long other);
```

The function returns:

- > 0 if A > B
- < 0 if A< B
- 0 if A = B

See Also `SysQSort`

SysKeyboardDialog

Purpose Pop up the system keyboard if there is a field object with the focus. The field object's text chunk is edited directly.

Prototype

```
void SysKeyboardDialog (KeyboardType kbd)
```

Parameters

`kbd`	The keyboard type. See `Keyboard.h`.

Result Returns nothing. Changes the field's text chunk.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `SysKeyboardDialogV10`, `FrmSetFocus`

SysKeyboardDialogV10

Purpose Pop up the system keyboard if there is a field object with the focus. The field object's text chunk is edited directly.

Prototype

```
void SysKeyboardDialogV10 ()
```

Parameters None.

Result Returns nothing. The field's text chunk is changed.

Compatibility Corresponds to the 1.0 implementation of `SysKeyboardDialog`.

See Also `SysKeyboardDialog`, `FrmSetFocus`

SysLibFind

Purpose Return a reference number for a library that is already loaded, given its name.

Prototype

```
Err SysLibFind (const Char* nameP,
UInt16* refNumP)
```

Parameters

`nameP`	Pointer to the name of a loaded library.
`refNumP`	Pointer to a variable for returning the library reference number (on failure, this variable is undefined)

Result 0 if no error; otherwise: `sysErrLibNotFound` (if the library is not yet loaded), or another error returned from the library's install entry point.

Comments Most built-in libraries (NetLib, serial, IR) are preloaded automatically when the system is reset. Third-party libraries must be loaded before this call can succeed (use `SysLibLoad`). You can check if a library is already loaded by calling `SysLibFind` and checking for a 0 error return value (it will return a non-zero value if the library is not loaded).

SysLibLoad

Purpose Load a library given its database creator and type.

Prototype

```
Err SysLibLoad (UInt32 libType, UInt32 libCreator,
UInt16* refNumP)
```

Parameters

`libType`	Type of library database.
`libCreator`	Creator of library database.
`refNumP`	Pointer to variable for returning the library reference number (on failure, `sysInvalidRefNum` is returned in this variable)

Result 0 if no error; otherwise: `sysErrLibNotFound`, `sysErrNoFreeRAM`, `sysErrNoFreeLibSlots`, or other error returned from the library's install entry point.

Comments Presently, the "load" functionality is *not* supported when you use the Palm OS Simulator.

When an application no longer needs a library that it *successfully* loaded via `SysLibLoad`, it is responsible for unloading the library by calling `SysLibRemove` and passing it the library reference number returned by `SysLibLoad`. More information is available in the white paper on shared libraries, which you can find on the Palm developer support web site.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysLibRemove

Purpose Unload a library previously loaded with `SysLibLoad`.

Prototype

```
Err SysLibRemove (UInt16 refNum)
```

Parameters

`-> refNum`	The library reference number.

Result 0 if no error; otherwise `sysErrParamErr` if the refNum is not a valid library reference number.

Comments `SysLibRemove` releases the dynamic memory allocated to the shared library's dispatch table, resources, and global variables.

SysQSort

Purpose Sort elements in an array according to the passed comparison function. Equal records can be in any position relative to each other because a quick sort tends to scramble the ordering of records. As a result, calling `SysQSort` multiple times can result in a different order if the records are not completely unique. If you don't want this behavior, use the insertion sort instead (see `SysInsertionSort`).

To pick the pivot point, the quick sort algorithm picks the middle of three records picked from around the middle of all records. That way, the algorithm can take advantage of partially sorted data.

These optimizations are built in:

- The routine contains its own stack to limit uncontrolled recursion. When the stack is full, an insertion sort is used because it doesn't require more stack space.
- An insertion sort is also used when the number of records is low. This avoids the overhead of a quick sort which is noticeable for small numbers of records.
- If the records seem mostly sorted, an insertion sort is performed to move only those few records that need to be moved.

Prototype

```
void SysQSort (void* baseP, Int16 numOfElements,
Int16 width, const CmpFuncPtr comparF,
const Int32 other)
```

Parameters

`baseP`	Base pointer to an array of elements.
`numOfElements`	Number of elements to sort (must be at least 2).
`width`	Width of an element.
`comparF`	Comparison function. See Comments for `SysInsertionSort`.
`other`	Other data passed to the comparison function.

Result Returns nothing.

See Also `SysInsertionSort`

SysRandom

Purpose Return a random number anywhere from 0 to `sysRandomMax`.

Prototype

```
Int16 SysRandom (UInt32 newSeed)
```

Parameters

`newSeed`	New seed value, or 0 to use existing seed.

Result Returns a random number.

SysReset

Purpose Perform a soft reset and reinitialize the globals and the dynamic memory heap.

Prototype

```
void SysReset (void)
```

Parameters None.

Result No return value.

Comments This routine resets the system, reinitializes the globals area and all system managers, and reinitializes the dynamic heap. All database information is preserved. This routine is called when the user presses the hidden reset switch on the device.

When running an application using the simulator, this routine looks for two data files that represent the memory of card 0 and card 1. If these are found, the Palm OS memory image is created using them. If they are not found, they are created.

When running an application on the device, this routine simply looks for the memory cards at fixed locations.

SysSetAutoOffTime

Purpose Set the time out value in seconds for auto-power-off. Zero means never power off.

Prototype

```
UInt16 SysSetAutoOffTime (UInt16 seconds)
```

Parameters

`seconds`	Time out in seconds, or 0 for no time out.

Result Returns previous value of time out in seconds.

SysSetTrapAddress

Purpose Set the address of the function corresponding to a system trap.

Prototype

```
Err SysSetTrapAddress (UInt16 trapNum,
void* procP)
```

Parameters

-> `trapNum`	One of the routine selectors defined in `SysTraps.h` (`sysTrap...`) or `CoreTraps.h` on Palm OS version 3.5 and higher.
-> `procP`	Pointer to a function that the trap identified by `trapNum` is to point to.

Result Returns 0 if no error, or `SysErrParamErr` if `trapNum` is greater than the number of traps in the trap table.

Comments This function is useful for patching a system trap, in combination with `SysGetTrapAddress`. To patch a system trap in your application, first call `SysGetTrapAddress` to get the trap address and then save this value somewhere. Next use `SysSetTrapAddress` to set the trap address to point to your function. Before your application exits, remove the patch by calling `SysSetTrapAddress` and passing in the original trap address you saved.

WARNING! If your application patches a system trap using this function, you **must** remove the patch before your application exits. Do **not** use this mechanism to permanently patch system traps as it may cause unpredictable results for the system and other applications.

SysStringByIndex

Purpose Copy a string out of a string list resource by index. String list resources are of type 'tSTL' and contain a list of strings and a prefix string.

ResEdit always displays the items in the list as starting at 1, not 0. Consider this when creating your string list.

Prototype

```
Char* SysStringByIndex (UInt16 resID,
UInt16 index, Char* strP, UInt16 maxLen)
```

Parameters

`resID`	Resource ID of the string list.
`index`	String to get out of the list.
`strP`	Pointer to space to form the string.
`maxLen`	Size of `strP` buffer.

Result Returns a pointer to the copied string. The string returned from this call will be the prefix string appended with the designated index string. Indices are 0-based; index 0 is the first string in the resource.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysTaskDelay

Purpose Put the processor into doze mode for the specified number of ticks.

Prototype
```
Err SysTaskDelay (Int32 delay)
```

Parameters

`delay`	Number of ticks to wait (see `SysTicksPerSecond`)

Result Returns 0 if no error.

See Also `EvtGetEvent`

SysTicksPerSecond

Purpose Return the number of ticks per second. This routine allows applications to be tolerant of changes to the ticks per second rate in the system.

Prototype
```
UInt16 SysTicksPerSecond (void)
```

Parameters None

Result Returns the number of ticks per second.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysUIAppSwitch

Purpose Try to make the current UI application quit and then launch the UI application specified by card number and database ID.

Prototype
```
Err SysUIAppSwitch (UInt16 cardNo, LocalID dbID,
UInt16 cmd, MemPtr cmdPBP)
```

Parameters

`cardNo`	Card number for the new application; currently only card 0 is valid.

`dbID`	ID of the new application.
`cmd`	Action code (launch code).
`cmdPBP`	Action code (launch code) parameter block.

Result Returns 0 if no error.

Comments Do not use this function to open the system-supplied Application Launcher application. If another application has replaced the default launcher with one of its own, this function will open the custom launcher instead of the system-supplied one. To open the system-supplied launcher reliably, enqueue a `keyDownEvent` that contains a `launchChr`, as shown in the section "Application Launcher" of the user interface chapter in the *Palm OS Programmer's Companion*.

If you are passing a parameter block (the `cmdPBP` parameter), you must set the owner of the parameter block chunk to the operating system. To do this, and for more information, see `MemPtrSetOwner`. If the parameter block structure contains references by pointer or handle to any other chunks, you also must set the owner of those chunks by using `MemHandleSetOwner` or `MemPtrSetOwner`.

See Also `SysAppLaunch`, Chapter 3, "Application Startup and Stop." in the *Palm OS Programmer's Companion*.

47

Text Manager

This chapter provides information about the text manager by discussing these topics:

- Text Manager Data Structures
- Text Manager Functions

The header file `TextMgr.h` declares the API that this chapter describes. For more information on the text manager, see the chapter "Localized Applications" in the *Palm OS Programmer's Companion*.

Text Manager Data Structures

CharEncodingType

The `CharEncodingType` enum specifies possible character encodings. A given device supports a single character encoding. The currently available devices support either Windows code page 1252 (an extension of ISO Latin 1) or Windows code page 932 (an extension of Shift JIS).

```
typedef enum {
  charEncodingUnknown = 0,

  charEncodingAscii,
  charEncodingISO8859_1,
  charEncodingPalmLatin,
  charEncodingShiftJIS,
  charEncodingPalmSJIS,
  charEncodingUTF8,
  charEncodingCP1252,
  charEncodingCP932
} CharEncodingType;
```

Value Descriptions

`charEncodingUnknown`	Unknown to this version of Palm OS®.
`charEncodingAscii`	ISO 646-1991.
`charEncodingISO8859_1`	ISO 8859 Part 1 (also known as ISO Latin 1). This encoding is commonly used for the Roman alphabet.
`charEncodingPalmLatin`	Palm OS version of Microsoft Windows code page 1252
`charEncodingShiftJIS`	Encoding for 0208-1990 with single-byte Japanese Katakana. This encoding is commonly used for Japanese alphabets.
`charEncodingPalmSJIS`	Palm OS version of Microsoft Windows code page 932
`charEncodingCP1252`	Microsoft Windows extensions to ISO 8859 Part 1.
`charEncodingCP932`	Microsoft Windows extensions to Shift JIS.
`charEncodingUTF8`	Eight-bit safe encoding for Unicode.

Text Manager Functions

TxtByteAttr

Purpose Return the possible locations of a given byte within a multi-byte character.

Prototype

```
UInt8 TxtByteAttr (UInt8 inByte)
```

Parameters

`-> inByte`	A byte representing all or part of a valid character.

Result Returns a byte with one or more of the following bits set:

`byteAttrFirst`	First byte of multi-byte character.
`byteAttrLast`	Last byte of multi-byte character.
`byteAttrMiddle`	Middle byte of multi-byte character.
`byteAttrSingle`	Single-byte character.

Comments If `inByte` is valid in more than one location of a character, multiple return bits are set. For example, 0x40 in the Shift JIS character encoding is valid as a single-byte character and as the low-order byte of a double-byte character. Thus, the return value for `TxtByteAttr(0x40)` on a Shift JIS system has both the `byteAttrSingle` and `byteAttrLast` bits set.

Text manager functions that need to determine the byte positioning of a character use `TxtByteAttr` to do so. You rarely need to use this function yourself.

Compatibility Implemented only if International Feature Set is present.

TxtCaselessCompare

Purpose Perform a case-insensitive comparison of two text buffers.

Prototype

```
Int16 TxtCaselessCompare (const Char* s1,
UInt16 s1Len, UInt16* s1MatchLen, const Char* s2,
UInt16 s2Len, UInt16* s2MatchLen)
```

Parameters

`-> s1`	Pointer to the first text buffer to compare. Must not be `NULL`.
`-> s1Len`	Length in bytes of the text pointed to by `s1`.
`<- s1MatchLen`	Points to the length in bytes of the text in `s1` that matched text in `s2`. Pass `NULL` for this parameter if you don't need to know this number.
`-> s2`	Pointer to the second text buffer to compare. Must not be `NULL`.
`-> s2Len`	Length in bytes of the text pointed to by `s2`.
`<- s2MatchLen`	Points to the length in bytes of the text in `s2` that matched text in `s1`. Pass `NULL` for this parameter if you don't need to know this number.

Result Returns one of the following values:

< 0	If `s1` occurs before `s2` in alphabetical order.
> 0	If `s1` occurs after `s2` in alphabetical order.
0	If the two substrings that were compared are equal.

Comments In certain character encodings (such as Shift JIS), one character may be accurately represented as either a single-byte character or a multi-byte character. `TxtCaselessCompare` accurately matches a single-byte character with its multi-byte equivalent. For this reason, the values returned in `s1MatchLen` and `s2MatchLen` are not always equal.

You must make sure that the parameters `s1` and `s2` point to a the start of a valid character. That is, they must point to the first byte of a multi-byte character or they must point to a single-byte character. If they don't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also `StrCaselessCompare`, `TxtCompare`, `StrCompare`

TxtCharAttr

Purpose Return a character's attributes.

Prototype

```
UInt16 TxtCharAttr (WChar inChar)
```

Parameters

`-> inChar`	Any valid character.

Result Returns a 16-bit unsigned value with any of the following bits set:

`charAttrPrint`	Printable
`charAttrSpace`	Blank space, tab, or newline
`charAttrAlNum`	Alphanumeric
`charAttrAlpha`	Alphabetic
`charAttrCntrl`	Control character
`charAttrGraph`	Character that appears on the screen; that is, is not whitespace, a control character, or a virtual character.
`charAttrDelim`	Word delimiter (whitespace or punctuation).

Comments The character passed to this function must be a valid character given the system encoding.

This function is used in the text manager's character attribute macros (`TxtCharIsAlNum`, `TxtCharIsCntrl`, and so on). The macros perform operations analogous to the standard C functions

isPunct, isPrintable, and so on. Usually, you'd use one of these macros instead of calling TxtCharAttr directly.

To obtain attributes specific to a given character encoding, use TxtCharXAttr.

Compatibility Implemented only if International Feature Set is present.

See Also TxtCharIsValid

TxtCharBounds

Purpose Return the boundaries of a character containing the byte at a specified offset in a string.

Prototype

```
WChar TxtCharBounds (const Char* inText,
UInt32 inOffset, UInt32* outStart, UInt32* outEnd)
```

Parameters

-> inText	Pointer to the text buffer to search. Must not be NULL.
-> inOffset	A valid offset into the buffer inText. This location may contain a byte in any position (start, middle, or end) of a multi-byte character.
<- outStart	Points to the starting offset of the character containing the byte at inOffset.
<- outEnd	Points to the ending offset of the character containing the byte at inOffset.

Result Returns the character located between the offsets outStart and outEnd.

Comments Use this function to determine the boundaries of a character in a string or text buffer.

If the byte at inOffset is valid in more than one location of a character, the function must search back toward the beginning of the text buffer until it finds an unambiguous byte to determine the appropriate boundaries. For this reason, TxtCharBounds is often slow and should be used only where needed.

You must make sure that the parameter `inText` points to the beginning of the string. That is, if the string begins with a multi-byte character, `inText` must point to the first byte of that character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

TxtCharEncoding

Purpose Return the minimum encoding required to represent a character.

Prototype

```
CharEncodingType TxtCharEncoding (WChar inChar)
```

Parameters

`-> inChar` A valid character.

Result A `CharEncodingType` value that indicates the minimum encoding required to represent `inChar`. If the character isn't recognizable, `charEncodingUnknown` is returned.

Comments The minimum encoding is the encoding that takes the lowest number of bytes to represent the character. For example, if the character is a blank or a tab character, the minimum encoding is `charEncodingAscii` because these characters can be represented in single-byte ASCII. If the character is a ü, the minimum encoding is `charEncodingISO8859_1`.

Because Palm OS® only supports a single character encoding at a time, the result of this function is always logically equal to or less than the encoding used on the current system. That is, you'll only receive a return value of `charEncodingISO8859_1` if you're running on a US or European system and you pass a non-ASCII character.

Use this function for informational purposes only. Your code should not assume that the character encoding returned by this function is the Palm OS system character encoding. (Instead use `FtrGet` as shown in the `TxtCharXAttr` function description.)

Use `TxtMaxEncoding` to determine the order of encodings.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtStrEncoding`, `TxtMaxEncoding`

TxtCharIsAlNum

Purpose Macro that indicates if the character is alphanumeric.

Prototype

```
TxtCharIsAlNum (ch)
```

Parameters

`-> ch`	A valid character.

Result Returns `true` if the character is a letter in an alphabet or a numeric digit, `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharIsDigit`, `TxtCharIsAlpha`

TxtCharIsAlpha

Purpose Macro that indicates if a character is a letter in an alphabet.

Prototype

```
TxtCharIsAlpha (ch)
```

Parameters

`-> ch`	A valid character.

Result Returns `true` if the character is a letter in an alphabet, `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharIsAlNum`, `TxtCharIsLower`, `TxtCharIsUpper`

TxtCharIsCntrl

Purpose Macro that indicates if a character is a control character.

Prototype `TxtCharIsCntrl (ch)`

Parameters `-> ch` A valid character.

Result Returns `true` if the character is a non-printable character, such as the bell character or a carriage return; `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

TxtCharIsDelim

Purpose Macro that indicates if a character is a delimiter.

Prototype `TxtCharIsDelim (ch)`

Parameters `-> ch` A valid character.

Result Returns `true` if the character is a word delimiter (whitespace or punctuation), `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

TxtCharIsDigit

Purpose Macro that indicates if the character is a decimal digit.

Prototype `TxtCharIsDigit (ch)`

Parameters `-> ch` A valid character.

Result Returns `true` if the character is 0 through 9, `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharIsAlNum`, `TxtCharIsHex`

TxtCharIsGraph

Purpose Macro that indicates if a character is a graphic character.

Prototype

```
TxtCharIsGraph (ch)
```

Parameters

`-> ch`	A valid character.

Result Returns `true` if the character is a graphic character, false otherwise.

Comments A graphic character is any character visible on the screen, in other words, letters, digits, and punctuation marks. A blank space is not a graphic character because it is not visible.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharIsPrint`

TxtCharIsHardKey

Purpose Macro that returns true if the character is one of the hard keys on the device.

Prototype

```
TxtCharIsHardKey (m, ch)
```

Parameters

-> `m`	The modifier keys from the `keyDownEvent`.
-> `ch`	The character from the `keyDownEvent`.

Result `true` if the character is one of the four built-in hard keys on the device, `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

See Also `ChrIsHardKey`

TxtCharIsHex

Purpose Macro that indicates if a character is a hexadecimal digit.

Prototype

```
TxtCharIsHex (ch)
```

Parameters

`-> ch`	A valid character.

Result Returns `true` if the character is a hexadecimal digit from 0 to F, `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharIsDigit`

TxtCharIsLower

Purpose Macro that indicates if a character is a lowercase letter.

Prototype

```
TxtCharIsLower (ch)
```

Parameters

`-> ch`	A valid character.

Result Returns `true` if the character is a lowercase letter, `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharIsAlpha`, `TxtCharIsUpper`

TxtCharIsPrint

Purpose Macro that indicates if a character is printable.

Prototype
```
TxtCharIsPrint (ch)
```

Parameters

`-> ch`	A valid character.

Result Returns `true` if the character is not a control or virtual character, `false` otherwise.

Comments This function differs from `TxtCharIsGraph` in that it returns `true` if the character is whitespace. `TxtCharIsGraph` returns `false` if the character is whitespace.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharIsValid`

TxtCharIsPunct

Purpose Macro that indicates if a character is a punctuation mark.

Prototype
```
TxtCharIsPunct (ch)
```

Parameters

`-> ch`	A valid character.

Result Returns `true` if the character is a punctuation mark, `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

TxtCharIsSpace

Purpose Macro that indicates if a character is a whitespace character.

Prototype `TxtCharIsSpace (ch)`

Parameters `-> ch` A valid character.

Result Returns `true` if the character is whitespace such as a blank space, tab, or newline; `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

TxtCharIsUpper

Purpose Macro that indicates if a character is an uppercase letter.

Prototype `TxtCharIsUpper (ch)`

Parameters `-> ch` A valid character.

Result Returns `true` if the character is an uppercase letter, `false` otherwise.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharIsAlpha`, `TxtCharIsLower`

TxtCharIsValid

Purpose Determine whether a character is valid character given the Palm OS character encoding.

Prototype

```
Boolean TxtCharIsValid (WChar inChar)
```

Parameters

`-> inChar` A character.

Result Returns `true` if `inChar` is a valid character; `false` if `inChar` is not a valid character.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharAttr`, `TxtCharIsPrint`

TxtCharSize

Purpose Return the number of bytes required to store the character in a string.

Prototype

```
UInt16 TxtCharSize (WChar inChar)
```

Parameters

`-> inChar` A valid character.

Result The the number of bytes required to store the character in a string.

Comments Outside of strings, characters are always two-byte long `WChar` values; however, strings may store characters as a single-byte value. If the character can be represented by a single byte (its high-order byte is 0), it is stored in a string as a single-byte character.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharBounds`

TxtCharWidth

Purpose Return the width required to display the specified character in the current font. If the specified character does not exist within the current font, the missing character symbol is substituted.

Prototype

```
Int16 TxtCharWidth (WChar inChar)
```

Parameters

`-> inChar` A valid character.

Result Returns the width of the specified character (in pixels).

Comments Use this function instead of `FntCharWidth` to determine the width of a single-byte or multi-byte character.

Compatibility Implemented only if International Feature Set is present.

TxtCharXAttr

Purpose Return the extended attribute bits for a character.

Prototype

```
UInt16 TxtCharXAttr (WChar inChar)
```

Parameters

`-> inChar` A valid character.

Result Returns an unsigned 16-bit value with one or more extended attribute bits set. For specific return values, look in the header files that are specific to certain character encodings (`CharLatin.h` or `CharShiftJIS.h`).

Comments To interpret the results, you must know the character encoding being used. Use `FtrGet` with `sysFtrNumEncoding` as the feature number to determine the character encoding. This returns one of the `CharEncodingType` values. For example:

```
WChar ch;
UInt16 encoding, attr;
...
attr = TxtCharXAttr(ch);
```

```
if (FtrGet(sysFtrCreator, sysFtrNumEncoding,
  &encoding) != 0)
  encoding = charEncodingCP1252;;
if (encoding == charEncodingUTF8) {
}
```

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharAttr`

TxtCompare

Purpose Performs a case-sensitive comparison of all or part of two text buffers.

Prototype

```
Int16 TxtCompare (const Char* s1, UInt16 s1Len,
UInt16* s1MatchLen, const Char* s2, UInt16 s2Len,
UInt16* s2MatchLen)
```

Parameters

`-> s1`	Pointer to the first text buffer to compare. Must not be `NULL`.
`-> s1Len`	The length in bytes of the text pointed to by `s1`.
`<- s1MatchLen`	Points to the length in bytes of the text in `s1` that matched text in `s2`. Pass `NULL` for this parameter if you don't need to know this number.
`-> s2`	Pointer to the second text buffer to compare. Must not be `NULL`.
`-> s2Len`	The length in bytes of the text pointed to by `s2`.
`<- s2MatchLen`	Points to the length in bytes of the text in `s2` that matched text in `s1`. Pass `NULL` for this parameter if you don't need to know this number.

Result Returns one of the following values:

< 0	If `s1` occurs before `s2` in alphabetical order.
> 0	If `s1` occurs after `s2` in alphabetical order.
0	If the two substrings that were compared are equal.

Comments In certain character encodings (such as Shift JIS), one character may be accurately represented as either a single-byte character or a multi-byte character. `TxtCompare` accurately matches a single-byte character with its multi-byte equivalent. For this reason, the values returned in `s1MatchLen` and `s2MatchLen` are not always equal.

This function performs a case-sensitive comparison. If you want to perform a case-insensitive comparison, use `TxtCaselessCompare`.

You must make sure that the parameters `s1` and `s2` point to the start of a a valid character. That is, they must point to the first byte of a multi-byte character or they must point to a single-byte character. If they don't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also `StrCompare`, `TxtFindString`

TxtEncodingName

Purpose Obtain a character encoding's name.

Prototype

```
const Char* TxtEncodingName
(CharEncodingType inEncoding)
```

Parameters

`-> inEncoding`	One of the values from `CharEncodingType`, indicating a character encoding.

Result A constant string containing the name of the encoding.

`encodingNameAscii`	us ascii
`encodingNameISO8859_1`	ISO-8859-1

`encodingNameCP1252`	ISO-8859-1-Windows-3.1-Latin-1
`encodingNameShiftJIS`	Shift_JIS
`encodingNameCP932`	Windows-31J
`encodingNameUTF8`	UTF-8
`""`	The encoding is not known

Comments Use this function to obtain the official name of the character encoding, suitable to pass to an Internet application or any other application that requires the character encoding's name to be passed along with the data.

Compatibility Implemented only if International Feature Set is present.

See Also `CharEncodingType`

TxtFindString

Purpose Perform a case-insensitive search for a string in another string.

Prototype

```
Boolean TxtFindString (const Char* inSourceStr,
const Char* inTargetStr, UInt32* outPos,
UInt16* outLength)
```

Parameters

`-> inSourceStr`	Pointer to the string to be searched. Must not be `NULL`.
`-> inTargetStr`	Prepared version of the string to be found.
`<- outPos`	Pointer to the offset of the match in `inSourceStr`.

`<- outLength`	Pointer to the length in bytes of the matching text.

Result Returns `true` if the function finds `inTargetStr` within `inSourceStr`; `false` otherwise.

If found, the values pointed to by the `outPos` and `outLength` parameters are set to the starting offset and the length of the matching text. If not found, the values pointed to by `outPos` and `outLength` are set to 0.

Comments Use this function instead of `FindStrInStr` to support the global system find facility. This function contains an extra parameter, `outLength`, to specify the length of the text that matched. Pass this value to `FindSaveMatch` in the `appCustom` parameter. Then when your application receives `sysAppLaunchCmdGoTo`, the `matchCustom` field contains the length of the matching text. You use the length of matching text to highlight the match within the selected record.

You must make sure that the parameters `inSourceStr` and `inTargetStr` point to the start of a valid character. That is, they must point to the first byte of a multi-byte character, or they must point to a single-byte character. If they don't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCaselessCompare`

TxtGetChar

Purpose Retrieve the character starting at the specified offset within a text buffer.

Prototype

```
WChar TxtGetChar (const Char* inText,
UInt32 inOffset)
```

Parameters

`-> inText`	Pointer to the text buffer to be searched. Must not be `NULL`.

`-> inOffset`	A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

Result Returns the character at `inOffset` in `inText`.

Comments You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtGetNextChar`, `TxtSetNextChar`

TxtGetNextChar

Purpose Retrieve the character starting at the specified offset within a text buffer.

Prototype

```
UInt16 TxtGetNextChar (const Char* inText,
UInt32 inOffset, WChar* outChar)
```

Parameters

`-> inText`	Pointer to the text buffer to be searched. Must not be `NULL`.
`-> inOffset`	A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.
`<- outChar`	The character at `inOffset` in `inText`. Pass `NULL` for this parameter if you don't need the character returned.

Result Returns the size in bytes of the character at `inOffset`. If `outChar` is not `NULL` upon entry, it points to the character at `inOffset` upon return.

Comments You can use this function to iterate through a text buffer character-by-character in this way:

```
UInt16 i = 0;
while (i < bufferLength) {
    i += TxtGetNextChar(buffer, i, &ch);
    //do something with ch.
}
```

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtGetChar`, `TxtGetPreviousChar`, `TxtSetNextChar`

TxtGetPreviousChar

Purpose Retrieve the character before the specified offset within a text buffer.

Prototype

```
UInt16 TxtGetPreviousChar (const Char* inText,
UInt32 inOffset, WChar* outChar)
```

Parameters

`-> inText`	Pointer to the text buffer to be searched. Must not be `NULL`.
`-> inOffset`	A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.
`<- outChar`	The character immediately preceding `inOffset` in `inText`. Pass `NULL` for this parameter if you don't need the character returned.

Result Returns the size in bytes of the character preceding `inOffset` in `inText`. If `outChar` is not `NULL` upon entry, then it points to the character preceding `inOffset` upon return. Returns 0 if `inOffset` is at the start of the buffer (that is, `inOffset` is 0).

Comments You can use this function to iterate through a text buffer character-by-character in this way:

```
/* Find the start of the character containing
the last byte. */
TxtCharBounds (buffer, bufferLength - 1,
&start, &end);
i = start;
while (i > 0) {
    i -= TxtGetPreviousChar(buffer, i, &ch);
    //do something with ch.
}
```

This function is often slower to use than `TxtGetNextChar` because it must determine the appropriate character boundaries if the byte immediately before the offset is valid in more than one location (start, middle, or end) of a multi-byte character. To do this, it must work backwards toward the beginning of the string until it finds an unambiguous byte.

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

TxtGetTruncationOffset

Purpose Return the appropriate byte position for truncating a text buffer such that it is at most a specified number of bytes long.

Prototype

```
UInt32 TxtGetTruncationOffset (const Char* inText,
UInt32 inOffset)
```

Parameters

`-> inText`	Pointer to a text buffer. Must not be `NULL`.
`-> inOffset`	A valid offset into the buffer `inText`.

Result Returns the appropriate byte offset for truncating `inText` at a valid inter-character boundary. The return value may be less than or equal to `inOffset`.

Comments You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

TxtMaxEncoding

Purpose Return the higher of two encodings.

Prototype

```
CharEncodingType TxtMaxEncoding
(CharEncodingType a, CharEncodingType b)
```

Parameters

`-> a`	A character encoding to compare.
`-> b`	Another character encoding to compare.

Result Returns the higher of `a` or `b`. One character encoding is higher than another if it is more specific. For example code page 1252 is "higher" than ISO 8859-1 because it represents more characters than ISO 8859-1.

Comments This function is used by `TxtStrEncoding` to determine the encoding required for a string.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharEncoding`, `CharEncodingType`

TxtNextCharSize

Purpose Macro that returns the size of the character starting at the specified offset within a text buffer.

Prototype

```
TxtNextCharSize (inText, inOffset)
```

Parameters

`-> inText`	Pointer to the text buffer to be searched. Must not be `NULL`.
`-> inOffset`	A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

Result Returns (as a `UInt16`) the size in bytes of the character at `inOffset`.

Comments You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtGetNextChar`

TxtParamString

Purpose Replace substrings within a string with the specified values.

Prototype

```
Char* TxtParamString (const Char* inTemplate,
const Char* param0, const Char* param1,
const Char* param2, const Char* param3)
```

Parameters

-> `inTemplate`	The string containing the substrings to replace.
-> `param0`	String to replace ^0 with or `NULL`.
-> `param1`	String to replace ^1 with or `NULL`.
-> `param2`	String to replace ^2 with or `NULL`.

-> `param3`	String to replace ^3 with or `NULL`.

Result Returns a locked handle to a newly allocated string in the dynamic heap that contains the appropriate substitutions.

Comments This function searches `inTemplate` for occurrences of the sequences ^0, ^1, ^2, and ^3. When it finds these, it replaces them with the corresponding string passed to this function. Multiple instances of each sequence will be replaced.

You must make sure that the parameter `inTemplate` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

`TxtParamString` allocates space for the returned string in the dynamic heap. Your code is responsible for freeing this memory when it is no longer needed.

Compatibility Implemented if 3.5 New Feature Set is present.

See Also `TxtReplaceStr`, `FrmCustomAlert`

TxtPreviousCharSize

Purpose Macro that returns the size of the character before the specified offset within a text buffer.

Prototype

```
TxtPreviousCharSize (inText, inOffset)
```

Parameters

-> `inText`	Pointer to the text buffer. Must not be `NULL`.
-> `inOffset`	A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

Result Returns (as a `UInt16`) the size in bytes of the character preceding `inOffset` in `inText`. Returns 0 if `inOffset` is at the start of the buffer (that is, `inOffset` is 0).

Comments You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtGetPreviousChar`

TxtReplaceStr

Purpose Replace a substring of a given format with another string.

Prototype

```
UInt16 TxtReplaceStr (Char* ioStr,
UInt16 inMaxLen, const Char* inParamStr,
UInt16 inParamNum)
```

Parameters

`<-> ioStr`	The string in which to perform the replacing. Must not be `NULL`.
`-> inMaxLen`	The maximum length in bytes that `ioStr` can become.
`-> inParamStr`	The string that ^*inParamNum* should be replaced with. If `NULL`, no changes are made.
`-> inParamNum`	A single-digit number (0 to 9).

Result Returns the number of occurrences found and replaced.

Returns a fatal error message if `inParamNum` is greater than 9.

Comments This function searches `ioStr` for occurrences of the string ^*inParamNum*, where *inParamNum* is any digit from 0 to 9. When it finds the string, it replaces it with `inParamStr`. Multiple instances will be replaced as long as the resulting string doesn't contain more than `inMaxLen` bytes, not counting the terminating null.

You can set the `inParamStr` parameter to `NULL` to determine the required length of `ioStr` before actually doing the replacing. `TxtReplaceStr` returns the number of occurrences it finds of ^*inParamNum*. Multiply this value by the length of the

`inParamStr` you intend to use to determine the appropriate length of `ioStr`.

You must make sure that the parameter `ioStr` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

TxtSetNextChar

Purpose Set a character within a text buffer.

Prototype

```
UInt16 TxtSetNextChar (Char* ioText,
UInt32 inOffset, WChar inChar)
```

Parameters

`<-> ioText`	Pointer to a text buffer. Must not be `NULL`.
`-> inOffset`	A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.
`-> inChar`	The character to replace the character at `inOffset` with. Must not be a virtual character.

Result Returns the size of `inChar`.

Comments This function replaces the character in `ioText` at the location `inOffset` with the character `inChar`. Note that there must be enough space at `inOffset` to write the character.

You can use `TxtCharSize` to determine the size of `inChar`.

You must make sure that the parameter `ioText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also TxtGetNextChar

TxtStrEncoding

Purpose Return the encoding required to represent a string.

Prototype

```
CharEncodingType TxtStrEncoding
(const Char* inStr)
```

Parameters

`-> inStr`	A string. Must not be `NULL`.

Result A `CharEncodingType` value that indicates the encoding required to represent `inChar`. If any character in the string isn't recognizable, then `charEncodingUnknown` is returned.

Comments The encoding for the string is the maximum encoding of any character in that string. For example, if a two-character string contains a blank space and a ü, the appropriate encoding is `charEncodingISO8859_1`. The blank space's minimum encoding is ASCII. The minimum encoding for the ü is ISO 8859-1. The maximum of these two encodings is ISO 8859-1.

Because Palm OS only supports a single character encoding at a time, the results of this function is always logically equal to or less than the encoding used on the current system. That is, you'll only receive a return value of `charEncodingISO8859_1` if you're running on a USA or European system.

Use this function for informational purposes only. Your code should not assume that the character encoding returned by this function is the Palm OS system's character encoding. (Instead use `FtrGet` as shown in the `TxtCharXAttr` function description.)

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharEncoding`, `TxtMaxEncoding`

TxtTransliterate

Purpose Converts the specified number of bytes in a text buffer using the specified operation.

Prototype

```
Err TxtTransliterate (const Char* inSrcText,
UInt16 inSrcLength, Char* outDstText,
UInt16* ioDstLength, TranslitOpType inOp)
```

Parameters

`-> inSrcText`	Pointer to a text buffer. Must not be `NULL`.
`-> inSrcLength`	The length in bytes of `inSrcText`.
`<- outDstText`	The output buffer containing the converted characters.
`<->ioDstLength`	Upon entry, the maximum length of `outDstText`. Upon return, the actual length of `outDstText`.

`-> inOp`	A 16-bit unsigned value that specifies which transliteration operation is to be performed. The values possible for this field are specific to the character encoding used on a particular device. These operations are universally available:
	`translitOpUpperCase`
	Converts the character to uppercase letters.
	`translitOpLowerCase`
	Converts the characters to lowercase letters.
	`translitOpPreprocess`
	Don't actually perform the operation. Instead, return in `ioDstLength` the amount of space required for the output text.

Result Returns one of the following values:

0	Success
`txtErrUnknownTranslitOp`	`inOp`'s value is not recognized

`txtErrTranslitOverrun`	If `inSrcText` and `outDstText` point to the same memory location and the operation has caused the function to overwrite unprocessed data in the input buffer.
`txtErrTranslitOverflow`	If `outDstText` is not large enough to contain the converted string.

Comments

`inSrcText` and `outDstText` may point to the same location if you want to perform the operation in place. However, you should be careful that the space required for `outDstText` is not larger than `inSrcText` so that you don't generate a `txtErrTranslitOverrun` error.

For example, suppose on a Shift JIS encoded system, you want to convert a series of single-byte Japanese Katakana symbols to double-byte Katakana symbols. You cannot perform this operation in place because it replaces a single-byte character with a multi-byte character. When the first converted character is written to the buffer, it overwrites the second input character. Thus, a text overrun has occurred.

You can ensure that you have enough space for the output by ORing your chosen operation with `translitOpPreprocess`. For example, to convert a string to uppercase letters, do the following:

```
outSize = buf2Len;
error = TxtTransliterate(buf1, buf1len, &buf2,
&outSize,
translitOpUpperCase|translitOpPreprocess);
if (outSize > buf2len)
    /* allocate more memory for buf2 */
error = TxtTransliterate(buf1, buf1Len, &buf2,
&outSize, translitOpUpperCase);
```

You must make sure that the parameter `inSrcText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

TxtWordBounds

Purpose Find the boundaries of a word of text that contains the character starting at the specified offset.

Prototype

```
Boolean TxtWordBounds (const Char* inText,
UInt32 inLength, UInt32 inOffset,
UInt32* outStart, UInt32* outEnd)
```

Parameters

`-> inText`	Pointer to a text buffer. Must not be `NULL`.
`-> inLength`	The length in bytes of the text pointed to by `inText`.
`-> inOffset`	A valid offset into the text buffer `inText`. This offset must point to the beginning of a character.
`<- outStart`	The starting offset of the text word.
`<- outEnd`	The ending offset of the text word.

Result Returns `true` if a word is found. Returns `false` if the word doesn't exist or is punctuation or whitespace.

Comments Assuming the ASCII encoding, if the text buffer contains the string "Hi! How are you?" and you pass 5 as the offset, `TxtWordBounds` returns the start and end of the word containing the character at offset 5, which is the character "o". Thus, `outStart` and `outEnd` would point to the start and end of the word "How".

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present.

See Also `TxtCharBounds`, `TxtCharIsDelim`

48

Windows

This chapter provides information about windows by discussing these topics:

- Window Data Structures
- Window Functions

No resources are associated with window objects.

The header file `Window.h` declares the API that this chapter describes. For more information on windows, see the section "Forms, Windows, and Dialogs" in the *Palm OS Programmer's Companion*.

Window Data Structures

CustomPatternType

The `CustomPatternType` type holds an 8-by-8 bit pattern that is one bit deep. Each byte specifies a row of the pattern. When drawing, a pattern is tiled to fill a specified region. This pattern is used by `WinFillLine` and `WinFillRectangle`.

The `PatternType` specifies the name of the current pattern.

```
typedef UInt8 CustomPatternType [8];
```

Compatibility

In pre-3.5 systems, the `CustomPatternType` is an array of 4 16-bit words. Note the size of this data type has not changed.

DrawStateType

The `DrawStateType` structure defines the current drawing state, which is the Palm OS® implementation of a **pen**. This drawing state

is saved with WinPushDrawState and restored with WinPopDrawState.

WARNING! Palm Computing® does not support or provide backward compatibility for the DrawStateType structure. Access it only through the functions described below. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```
typedef struct DrawStateType {
  WinDrawOperation  transferMode;
  PatternType       pattern;
  UnderlineModeType underlineMode;
  FontID            fontId;
  FontPtr           font;
  CustomPatternType patternData;
  IndexedColorType  foreColor;
  IndexedColorType  backColor;
  IndexedColorType  textColor;
} DrawStateType;
```

Field Description

transferMode	The current transfer mode for color drawing. See WinDrawOperation. Use WinSetDrawMode to set this value.
pattern	The name of the current pattern. See PatternType. If set to customPattern, the patternData field contains the actual pattern. Use WinGetPatternType and WinSetPatternType to retrieve and set this value.
underlineMode	The current underline mode. See UnderlineModeType. Use WinSetUnderlineMode to set this value.
fontId	The ID of the current font. Use FntSetFont to set this value.

`font`	A pointer to the current font. Use `FntSetFont` to set this value.
`patternData`	The current pattern being used by the `WinFill` functions if `pattern` is `customPattern`. See `CustomPatternType`. Use `WinGetPattern` and `WinSetPattern` to retrieve and set this value.
`foreColor`	Index of the current color used for the foreground. Use `WinSetForeColor` to set this value.
`backColor`	Index of the current color used for the background. Use `WinSetBackColor` to set this value.
`textColor`	Index of the current color used for text. Use `WinSetTextColor` to set this value.

Compatibility

This type is implemented only if 3.5 New Feature Set is present.

FrameBitsType

The `FrameBitsType` structure specifies attributes of a window's frame.

WARNING! Palm Computing does not support or provide backward compatibility for the `FrameBitsType` bit field. Never access its bit field members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```
typedef union FrameBitsType {
  struct {
    UInt16 cornerDiam : 8;
    UInt16 reserved_3 : 3;
    UInt16 threeD     : 1;
    UInt16 shadowWidth: 2;
```

```
      UInt16 width      : 2;
    } bits;
    UInt16 word;
  } FrameBitsType;
```

Field Descriptions

`cornerDiam`	Corner radius of frame; maximum is 38.
`reserved_3`	Reserved.
`threeD`	Set this bit to draw a 3D button. This feature is not currently supported.
`shadowWidth`	Width of shadow.
`width`	Frame width.
`word`	Provides access to all bits as a unit. This field is often used to convert a `FrameType` to a `FrameBitsType` as shown:

```
FrameType frame;
FrameBitsType frameType;

frameType.word = frame;
if (frameType.bits.threeD)
   ...
```

FrameType

The `FrameType` type specifies a window frame style.

```
typedef UInt16 FrameType;
```

The `FrameType` can be set to one of the defined frame types shown in the table below, or a custom frame type as defined by a `FrameBitsType` structure.

Constant	Value	Description
`noFrame`	0	No frame
`simpleFrame`	1	Plain rectangular frame
`rectangleFrame`	1	Plain rectangular frame

Constant	Value	Description
`simple3DFrame`	0x0012	3D frame with width of 2. This frame type is not supported.
`roundFrame`	0x0401	Round frame with width of 1.
`boldRoundFrame`	0x0702	Round frame with width of 2.
`popupFrame`	0x0205	Popup frame style with slight corner roundness, width of 1 and shadow of 1.
`dialogFrame`	0x0302	Dialog frame style with slight corner roundness and width of 2.
`menuFrame`	`popupFrame`	Same as `popupFrame`.

IndexedColorType

The `IndexedColorType` type is used to specify a color by its index value; that is, by its location in a color table. Color tables are defined by the `ColorTableType` structure, which is declared in `Bitmap.h`. The `IndexedColorType` can hold a 1, 2, 4, or 8-bit index.

```
typedef UInt8 IndexedColorType;
```

Compatibility

This type is implemented only if 3.5 New Feature Set is present.

PatternType

The `PatternType` enumerated type specifies a pattern for drawing. This type is returned by `WinGetPatternType` and is used as a parameter to the `WinSetPatternType` function.

```
typedef enum { blackPattern, whitePattern,
grayPattern, customPattern } PatternType;
```

Value Descriptions

`blackPattern`	Pattern with all bits on.
`whitePattern`	Pattern with all bits off.

`grayPattern`	Pattern with alternating on and off bits.
`customPattern`	Custom pattern specified by `CustomPatternType`.

These patterns all operate with current foreground and background color instead of black and white. In effect, `blackPattern` is only black if the current foreground color is black.

UnderlineModeType

The `UnderlineModeType` enumerated type specifies possible values for the underline mode stored in `DrawStateType`.

```
typedef enum { noUnderline, grayUnderline,
solidUnderline, colorUnderline }
UnderlineModeType;
```

Value Descriptions

`noUnderline`	No underline.
`grayUnderline`	Underline is drawn using a dotted line in the current foreground color.
`solidUnderline`	Underline is drawn using a solid line in the foreground color.
`colorUnderline`	Underline is drawn using a solid line in the foreground color.

Compatibility

The `solidUnderline` and `colorUnderline` options are only available in Palm OS 3.1 and higher.

WindowFlagsType

The `WindowFlagsType` specifies different window attributes.

WARNING! Palm Computing does not support or provide backward compatibility for the `WindowFlagsType` bit field. Access it only through the functions described below. Never access its bit field members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```
typedef struct WindowFlagsType {
  UInt16 format:1;
  UInt16 offscreen:1;
  UInt16 modal:1;
  UInt16 focusable:1;
  UInt16 enabled:1;
  UInt16 visible:1;
  UInt16 dialog:1;
  UInt16 freeBitmap:1;
  UInt16 reserved :8;
} WindowFlagsType;
```

Field Descriptions

`format` If set, use the `genericFormat`. If 0, use `screenFormat`.
Screen format is the native format of the video system; windows in this format can be copied to the display faster. The generic format is device-independent. A window cannot be enabled (that is, accept pen input) unless it uses screen format.

`offscreen` If set, the window is offscreen. If 0, the window is onscreen.

`modal` If set, the window is modal. If 0, the window is not modal. You set this value when you create the window. This value is returned by `WinModal`.

`focusable` If set, the window can accept the focus. If 0, the window does not accept the focus. You set this value when you create the window.

`enabled`	If set, the window is enabled. If 0, the window is disabled.
`visible`	If set, the window is visible if it is onscreen. If 0, the window is not visible.
`dialog`	If set, the window is a form. If 0, the window is not a form. The `FrmInitForm` function sets this value.
`freeBitmap`	If set, free the bitmap when the window is freed. If 0, retain the bitmap after the window is freed.
`reserved`	Reserved for future use. Must be 0.

Compatibility

In OS versions previous to 3.5, the `freeBitmap` flag was not present. Instead, a `compressed` flag was present, where 0 specified uncompressed and 1 specified compressed. This compressed flag is now part of the `BitmapType`.

WindowType

The `WindowType` structure represents a window.

WARNING! Palm Computing does not support or provide backward compatibility for the `WindowType` structure. Access it only through the functions described below. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```
typedef struct WindowType {
  Coord              displayWidthV20;
  Coord              displayHeightV20;
  void *             displayAddrV20;
  WindowFlagsType    windowFlags;
  RectangleType      windowBounds;
  AbsRectType        clippingBounds;
  BitmapPtr          bitmapP;
  FrameBitsType      frameType;
  DrawStateType *    drawStateP;
```

```
    struct WindowType * nextWindow;
  } WindowType;
```

Field Descriptions

`displayWidthV20`	Width of the window in pre OS 3.5 devices. In OS 3.5, use `WinGetDisplayExtent` to return the window width.
`displayHeightV20`	Height of the window in pre OS 3.5 devices. In OS 3.5, use `WinGetDisplayExtent` to return the window height.
`displayAddrV20`	Pointer to the window display memory buffer in pre OS 3.5 devices. In OS 3.5 or later, call `WinGetBitmap` and then `BmpGetBits` to obtain the display's memory buffer.
`windowFlags`	Window attributes (see `WindowFlagsType`).
`windowBounds`	Display-relative bounds of the window. Use `WinGetWindowBounds` and `WinSetWindowBounds` to retrieve and set this value.
`clippingBounds`	Bounds for clipping any drawing within the window. Use `WinGetClip` and `WinSetClip` to retrieve and set this value.
`bitmapP`	Pointer to the window bitmap, which holds the window's contents. Use `WinGetBitmap` to retrieve this value.
`frameType`	Frame attributes; see `FrameBitsType`.
`drawStateP`	Pointer to a state of the current transfer mode, pattern mode, font, underline mode, and colors. See `DrawStateType`. Only one drawing state exists in the system. Each window points to the same structure.
`nextWindow`	Pointer to the next window in a linked list of windows. This linked list of windows is called the active window list.

Compatibility

In OS versions previous to 3.5, this structure is slightly different. Specifically, the `bitmapP` field is instead a `viewOrigin` field of type `PointType` and specified the window origin point on the display. The `drawStateP` was named `gstate` and was of type `GraphicStatePtr`. The complete definition is shown below:

```
typedef struct WinTypeStruct {
  Word                    displayWidth;
  Word                    displayHeight;
  VoidPtr                 displayAddr;
  WindowFlagsType         windowFlags;
  RectangleType           windowBounds;
  AbsRectType             clippingBounds;
  PointType               viewOrigin;
  FrameBitsType           frameType;
  GraphicStatePtr         gstate;
  struct WinTypeStruct*     nextWindow;
} WindowType;
```

WinDrawOperation

The `WinDrawOperation` enumerated type specifies the transfer mode for color drawing. This type is used as a parameter to the `WinCopyRectangle` and `WinSetDrawMode` functions.

```
typedef enum {winPaint, winErase, winMask,
winInvert, winOverlay, winPaintInverse,
winSwap} WinDrawOperation;
```

Value Descriptions

`winPaint`	Destination replaced with source pixels (copy mode).
`winErase`	Destination cleared where source pixels are off (AND mode).
`winMask`	Destination cleared where source pixels are on (AND NOT mode).
`winInvert`	Destination inverted where source pixels are on (XOR mode).

`winOverlay`	Destination set only where source pixels are on (OR mode).
`winPaintInverse`	Destination replaced with inverted source (copy NOT mode).
`winSwap`	Destination foreground and background colors are swapped, leaving any other colors unchanged (color invert operation).

Compatibility

This type is implemented only if 3.5 New Feature Set is present. In earlier releases, this type is named `ScrOperation` and its values begin with the prefix `scr` rather than `win`. `WinDrawOperation` is fully compatible with `ScrOperation`.

WinHandle

The `WinHandle` type is a pointer to a `WindowType` structure. Note that this may change.

```
typedef WindowType * WinHandle;
```

WinLineType

The `WinLineType` structure defines a line.

```
typedef struct WinLineType {
  Coord x1;
  Coord y1;
  Coord x2;
  Coord y2;
} WinLineType;
```

Field Descriptions

`x1`	X coordinate of the first endpoint of the line.
`y1`	Y coordinate of the first endpoint of the line.
`x2`	X coordinate of the second endpoint of the line.
`y2`	Y coordinate of the second endpoint of the line.

Compatibility

This type is implemented only if 3.5 New Feature Set is present.

WinPtr

The `WinPtr` type is a pointer to a `WindowType` structure.

```
typedef WindowType * WinPtr;
```

Window Functions

WinClipRectangle

Purpose Clip a rectangle to the clipping rectangle of the draw window.

Prototype

```
void WinClipRectangle (RectangleType *rP)
```

Parameters

`<-> rP`	Pointer to a structure holding the rectangle to clip. The rectangle returned is the intersection of the rectangle passed and the clipping bounds of the draw window.

Result Returns nothing.

Comments The draw window is the window to which all drawing functions send their output. It is returned by `WinGetDrawWindow`.

See Also `WinCopyRectangle`, `WinDrawRectangle`, `WinEraseRectangle`, `WinGetClip`

WinCopyRectangle

Purpose Copy a rectangular region from one place to another (either between windows or within a single window).

Prototype

```
void WinCopyRectangle (WinHandle srcWin,
WinHandle dstWin, RectangleType *srcRect,
Coord destX, Coord destY, WinDrawOperation mode)
```

Parameters

`-> srcWin`	Window from which the rectangle is copied. If `NULL`, use the draw window.
`-> dstWin`	Window to which the rectangle is copied. If `NULL`, use the draw window.
`-> srcRect`	Bounds of the region to copy.
`-> destX`	Top bound of the rectangle in destination window.
`-> destY`	Left bound of the rectangle in destination window.
`-> mode`	The method of transfer from the source to the destination window (see `WinDrawOperation`).

Result Returns nothing.

Comments Copies the bits of the window inside the rectangle region.

If the destination bitmap is compressed, the mode parameter must be `winPaint`, and the destination coordinates must be (0,0). If the width of the destination rectangle is less than 16 pixels or if the destination coordinates are not (0,0), then this function turns off compression for the destination bitmap. Normally, you do not copy to a compressed bitmap. Instead, you copy to an uncompressed bitmap and compress it afterwards.

Compatibility In OS versions before 3.5, the `mode` parameter was defined as type `ScrOperation`. It is defined as type `WinDrawOperation` only if 3.5 New Feature Set is present. `ScrOperation` and `WinDrawOperation` are fully compatible with each other.

In OS versions before 3.5, it was common practice to render a bitmap in an offscreen window and then use `WinCopyRectangle` to draw it on the screen. In version 3.5 and higher, the preferred method of doing this is to use `WinDrawBitmap` or `WinPaintBitmap`.

See Also `WinDrawBitmap`

WinCreateBitmapWindow

Purpose Create a new offscreen window.

Prototype

```
WinHandle WinCreateBitmapWindow
(BitmapType *bitmapP, UInt16 *error)
```

Parameters

`-> bitmapP`	Pointer to a bitmap to associate with the window. (See `BitmapType`.)
`<- error`	Pointer to any error this function encounters.

Result Returns the handle of the new window upon success, or `NULL` if an error occurs. The `error` parameter contains one of the following:

`errNone`	No error.
`sysErrParamErr`	The `bitmapP` parameter is invalid. The bitmap must be uncompressed and it must have a valid pixel size (1, 2, 4, or 8). It must not be the screen bitmap.
`sysErrNoFreeResource`	There is not enough memory to allocate a new window structure.

Comments Use `WinCreateBitmapWindow` if you want to draw into a previously created bitmap, such as a bitmap created using `BmpCreate`.

This function generates a window wrapper for the specified bitmap. The newly created window is offscreen, uses the generic format (for device independence), and is added to the active window list. Use

WinSetDrawWindow to make it the draw window, and then use the window drawing functions to modify the bitmap.

When you use this function to create a window and then delete the window with WinDeleteWindow, the bitmap is **not** freed when the window is freed.

WinCreateOffscreenWindow uses this function to create its offscreen window. If you call WinCreateOffscreenWindow instead of using this function, the bitmap is freed when WinDeleteWindow is called.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinCreateWindow, WinCreateOffscreenWindow

WinCreateOffscreenWindow

Purpose Create a new offscreen window and add it to the window list.

Prototype

```
WinHandle WinCreateOffscreenWindow (Coord width,
Coord height, WindowFormatType format,
UInt16 *error)
```

Parameters

-> width	Width of the window in pixels.
-> height	Height of the window in pixels.
-> format	Either screenFormat or genericFormat. In general, you should use genericFormat for offscreen windows.
<- error	Pointer to any error this function encounters.

Result Returns the handle of the new window upon success, or NULL if an error occurs. The error parameter contains one of the following:

errNone	No error.
sysErrParamErr	The width or height parameter is NULL or the current color table is invalid.

`sysErrNoFreeResource`
: There is not enough memory to complete the function.

The debug ROM gives a warning if you try to draw to a bad window address.

Comments

Windows created with this routine draw to a memory buffer instead of the display. Use this function for temporary drawing operations such as double-buffering or save-behind operations.

The memory buffer has two formats: screen format and generic format. Screen format is the native format of the video system; windows in this format can be copied to the display faster. The generic format is device-independent. A window cannot be enabled (that is, accept pen input) unless it uses screen format.

This function differs from `WinCreateBitmapWindow` in the following ways:

- `WinCreateOffscreenWindow` creates a new bitmap in the same depth as the current screen. `WinCreateBitmapWindow` uses the bitmap you pass in, which may or may not be in the same depth as the current screen.
- `WinCreateOffscreenWindow` uses the screen format you specify. `WinCreateBitmapWindow` always uses generic format.
- When you delete the window created with `WinCreateOffscreenWindow`, its bitmap is freed along with the window. The bitmap used in the `WinCreateBitmapWindow` is not freed when the window is freed.

See Also `WinCreateWindow`

WinCreateWindow

Purpose Create a new window and add it to the window list.

Prototype

```
WinHandle WinCreateWindow (RectangleType *bounds,
FrameType frame, Boolean modal, Boolean focusable,
UInt16 *error)
```

Parameters

`-> bounds`	Display-relative bounds of the window.
`-> frame`	Type of frame around the window (see `FrameType`).
`-> modal`	`true` if the window is modal.
`-> focusable`	`true` if the window can be the active window.
`<- error`	Pointer to any error encountered by this function.

Result Returns the handle of the new window upon success, or `NULL` if an error occurs. The `error` parameter contains one of the following:

`errNone`	No error.
`sysErrNoFreeResource`	There is not enough memory to complete the operation.

Comments Windows created by this routine draw to the display. See `WinCreateOffscreenWindow` for information on drawing off screen.

You typically don't call this function directly. Instead, you use `FrmInitForm` to create form windows from a resource. Forms are much more flexible and have better system support. All forms are windows, but not all windows are forms.

The window is created with the bounds and frame type that you specify and uses the bitmap and drawing state of the current draw window. Its clipping region is reset according to the bounds you specify.

All window flags are set to 0 except for the modal and focusable flags, which you pass as a parameter to this function. Specifically, newly created windows are disabled and invisible. You must specifically enable the window before the window can accept input. You can do so with `WinSetActiveWindow`.

See Also `WinDeleteWindow`

WinDeleteWindow

Purpose Remove a window from the window list and free the memory used by the window.

Prototype

```
void WinDeleteWindow (WinHandle winHandle,
Boolean eraseIt)
```

Parameters

`-> winHandle`	Handle of window to delete.
`-> eraseIt`	If `true`, the window is erased before it is deleted. If `false`, the window is not erased.

Result Returns nothing.

Comments This function frees all memory associated with the window. Windows created using `WinCreateOffscreenWindow` have their bitmaps freed; windows created using `WinCreateWindow` or `WinCreateBitmapWindow` do not.

The `eraseIt` parameter affects onscreen windows only; offscreen windows are never erased. As a performance optimization, you might set `eraseIt` to `false` for an onscreen window if you know that you are going to immediately redraw the area anyway. For example, when the form manager closes a form dialog, it restores the area with the save-behind bits it had stored for that form. For this reason, when the form manager deletes the dialog window, it passes `false` for `eraseIt` because the entire area will be redrawn.

WinDisplayToWindowPt

Purpose Convert a display-relative coordinate to a window-relative coordinate. The coordinate returned is relative to the display window.

Prototype

```
void WinDisplayToWindowPt (Coord *extentX,
Coord *extentY)
```

Parameters

`<-> extentX`	Pointer to x coordinate to convert.
`<-> extentY`	Pointer to y coordinate to convert.

Result Returns nothing.

See Also `WinWindowToDisplayPt`

WinDrawBitmap

Purpose Draw a bitmap at the given coordinates in `winPaint` mode (see `WinDrawOperation` for mode details).

Prototype

```
void WinDrawBitmap (BitmapPtr bitmapP, Coord x,
Coord y)
```

Parameters

`-> bitmapP`	Pointer to a bitmap.
`-> x`	The x coordinate of the top-left corner.
`-> y`	The y coordinate of the top-left corner.

Result Returns nothing.

Comments If the bitmap has multiple depths (is a bitmap family), the closest match less than or equal to the current draw window depth is used. If such a bitmap does not exist, the bitmap with the closest match greater than the draw window depth is used.

If the bitmap has its own color table, color conversion to the draw window color table will be applied (on OS 3.5 or later). This color conversion is slow and not recommended. Instead of including a

color table in the bitmap, consider using WinPalette to change the system color table, draw the bitmap, and then change the system color table back when the bitmap is no longer visible.

This function differs from WinPaintBitmap in that this function always uses winPaint mode (copy mode) as the transfer mode. WinPaintBitmap uses the current drawing state transfer mode.

See Also WinEraseRectangle

WinDrawChar

Purpose Draw the specified character in the draw window.

Prototype

```
void WinDrawChar (WChar theChar, Coord x, Coord y)
```

Parameters

-> theChar	The character to draw. This may be either a single-byte character or a multi-byte character.
-> x	x coordinate of the location where the character is to be drawn (left bound).
-> y	y coordinate of the location where the character is to be drawn (top bound).

Result Returns nothing.

Comments Before calling this function, call WinSetUnderlineMode and FntSetFont to set the desired underline and font to draw the characters.

This function differs from WinPaintChar in that this function always uses winPaint mode (see WinDrawOperation). This means the on bits are drawn in the text color, the off bits are in the background color, and underlines are in the foreground color. WinPaintChar uses the current drawing state transfer mode instead of winPaint.

Compatibility Implemented only if 3.1 New Feature Set is present.

See Also WinDrawChars, WinDrawInvertedChars, WinDrawTruncChars, WinEraseChars, WinInvertChars, WinPaintChars

WinDrawChars

Purpose Draw the specified characters in the draw window.

Prototype

```
void WinDrawChars (const Char *chars, Int16 len,
Coord x, Coord y)
```

Parameters

`-> chars`	Pointer to the characters to draw.
`-> len`	Length in bytes of the characters to draw.
`-> x`	x coordinate of the first character to draw (left bound).
`-> y`	y coordinate of the first character to draw (top bound).

Result Returns nothing.

Comments This function is useful for printing non-editable status or warning messages on the screen.

Before calling this function, call WinSetUnderlineMode and FntSetFont to set the desired underline and font to draw the characters.

This function differs from WinPaintChars in that this function always uses `winPaint` mode (see WinDrawOperation). This means the on bits are drawn in the text color, the off bits are in the background color, and underlines are in the foreground color. `WinPaintChar` uses the current drawing state transfer mode instead of `winPaint`.

See Also WinDrawChar, WinDrawInvertedChars, WinDrawTruncChars, WinEraseChars, WinInvertChars, WinPaintChar

WinDrawGrayLine

Purpose Draw a dashed line in the draw window.

Prototype

```
void WinDrawGrayLine (Coord x1, Coord y1,
Coord x2, Coord y2)
```

Parameters

`-> x1`	x coordinate of line start point.
`-> y1`	y coordinate of line start point.
`-> x2`	x coordinate of line endpoint.
`-> y2`	y coordinate of line endpoint.

Result Returns nothing.

Comments This routine does not draw in the gray color; it draws with alternating foreground and background pixels. That is, it uses the `grayPattern` pattern type.

See Also `WinDrawLine`, `WinEraseLine`, `WinFillLine`, `WinInvertLine`, `WinPaintLine`, `WinPaintLines`

WinDrawGrayRectangleFrame

Purpose Draw a gray rectangular frame in the draw window.

Prototype

```
void WinDrawGrayRectangleFrame (FrameType frame,
RectangleType *rP)
```

Parameters

`-> frame`	Type of frame to draw (see `FrameType`).
`-> rP`	Pointer to the rectangle to frame.

Result Returns nothing.

Comments This routine does not draw in the gray color; it draws with alternating foreground and background pixels. The standard gray

pattern is not used by this routine; rather, the frame is drawn so that the top-left pixel of the frame is always on.

See Also WinDrawRectangleFrame, WinEraseRectangleFrame, WinGetFramesRectangle, WinInvertRectangleFrame, WinPaintRectangleFrame

WinDrawInvertedChars

Purpose Draw the specified characters inverted (background color) in the draw window.

Prototype

```
void WinDrawInvertedChars (const Char *chars,
Int16 len, Coord x, Coord y)
```

Parameters

`-> chars`	Pointer to the characters to draw.
`-> len`	Length in bytes of the characters to draw.
`-> x`	x coordinate of the first character to draw (left bound).
`-> y`	y coordinate of the first character to draw (top bound).

Result Returns nothing.

Comments This routine draws the **on** bits and any underline in the background color and the **off** bits in the text color. (Black and white uses copy NOT mode.) This is the standard function for drawing inverted text.

Before calling this function, consider calling WinSetUnderlineMode and FntSetFont.

See Also WinDrawChar, WinDrawChars, WinDrawTruncChars, WinEraseChars, WinInvertChars, WinPaintChar, WinPaintChars

WinDrawLine

Purpose Draw a line in the draw window using the current foreground color.

Prototype

```
void WinDrawLine (Coord x1, Coord y1, Coord x2,
Coord y2)
```

Parameters

`-> x1`	x coordinate of line start point.
`-> y1`	y coordinate of line start point.
`-> x2`	x coordinate of line endpoint.
`-> y2`	y coordinate of line endpoint.

Result Returns nothing.

Comments This function differs from `WinPaintLine` in that it always uses `winPaint` mode (see `WinDrawOperation`). `WinPaintLine` uses the current drawing state transfer mode instead of `winPaint`.

See Also `WinDrawGrayLine`, `WinEraseLine`, `WinFillLine`, `WinInvertLine`, `WinPaintLine`, `WinPaintLines`

WinDrawPixel

Purpose Draw a pixel in the draw window using the current foreground color.

Prototype

```
void WinDrawPixel (Coord x, Coord y)
```

Parameters

`-> x`	Pointer to the x coordinate of a pixel.
`-> y`	Pointer to the y coordinate of a pixel.

Result Returns nothing. May display a fatal error message if the draw window's bitmap is compressed.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinErasePixel`, `WinInvertPixel`, `WinPaintPixel`, `WinPaintPixels`

WinDrawRectangle

Purpose Draw a rectangle in the draw window using the current foreground color.

Prototype

```
void WinDrawRectangle (RectangleType *rP,
UInt16 cornerDiam)
```

Parameters

`-> rP`	Pointer to the rectangle to draw.
`-> cornerDiam`	Radius of rounded corners. Specify zero for square corners.

Result Returns nothing.

Comments The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

This function differs from `WinPaintRectangle` in that it always uses `winPaint` mode (see `WinDrawOperation`). `WinPaintRectangle` uses the current drawing state transfer mode instead of `winPaint`.

See Also `WinEraseRectangle`, `WinFillRectangle`, `WinInvertRectangle`

WinDrawRectangleFrame

Purpose Draw a rectangular frame in the draw window using the current foreground color.

Prototype

```
void WinDrawRectangleFrame (FrameType frame,
RectangleType *rP)
```

Parameters

`-> frame`	Type of frame to draw (see `FrameType`).
`-> rP`	Pointer to the rectangle to frame.

Result Returns nothing.

Comments The frame is drawn outside the specified rectangle.

This function differs from `WinPaintRectangleFrame` in that it always uses `winPaint` mode (see `WinDrawOperation`). `WinPaintRectangleFrame` uses the current drawing state transfer mode instead of `winPaint`.

See Also `WinDrawGrayRectangleFrame`, `WinEraseRectangleFrame`, `WinGetFramesRectangle`, `WinInvertRectangleFrame`

WinDrawTruncChars

Purpose Draw the specified characters in the draw window, truncating the characters to the specified width.

Prototype

```
void WinDrawTruncChars (const Char *chars,
Int16 len, Coord x, Coord y, Coord maxWidth)
```

Parameters

`-> chars`	Pointer to the characters to draw.
`-> len`	Length in bytes of the characters to draw.
`-> x`	x coordinate of the first character to draw (left bound).
`-> y`	y coordinate of the first character to draw (top bound).

`-> maxWidth`	Maximum width in pixels of the characters that are to be drawn.

Result Returns nothing.

Comments Before calling this function, consider calling `WinSetUnderlineMode` and `FntSetFont`.

If drawing all of the specified characters requires more space than `maxWidth` allows, `WinDrawTruncChars` draws one less than the number of characters that can fit in `maxWidth` and then draws an ellipsis (...) in the remaining space. (If the boundary characters are narrower than the ellipsis, more than one character may be dropped to make room.) If `maxWidth` is narrower than the width of an ellipsis, nothing is drawn.

Use this function to truncate text that may contain multi-byte characters.

Compatibility Implemented only if 3.1 New Feature Set is present.

See Also `WinDrawChar`, `WinDrawChars`, `WinDrawInvertedChars`, `WinEraseChars`, `WinInvertChars`, `WinPaintChar`, `WinPaintChars`

WinEraseChars

Purpose Erase the specified characters in the draw window.

Prototype

```
void WinEraseChars (const Char *chars, Int16 len,
Coord x, Coord y)
```

Parameters

`-> chars`	Pointer to the characters to erase.
`-> len`	Length in bytes of the characters to erase.
`-> x`	x coordinate of the first character to erase (left bound).

`-> y`	y coordinate of the first character to erase (top bound).

Result Returns nothing.

Comments The `winMask` transfer mode is used to erase the characters. See `WinDrawOperation` for more information. This has the effect of erasing only the on bits for the characters rather than the entire text rectangle. This function only works if the foreground color is black and the background color is white.

See Also `WinDrawChar`, `WinDrawChars`, `WinDrawInvertedChars`, `WinDrawTruncChars`, `WinInvertChars`, `WinPaintChar`, `WinPaintChars`

WinEraseLine

Purpose Draw a line in the draw window using the current background color.

Prototype

```
void WinEraseLine (Coord x1, Coord y1, Coord x2,
Coord y2)
```

Parameters

`-> x1`	x coordinate of line start point.
`-> y1`	y coordinate of line start point.
`-> x2`	x coordinate of line endpoint.
`-> y2`	y coordinate of line endpoint.

Result Returns nothing.

See Also `WinDrawGrayLine`, `WinDrawLine`, `WinFillLine`, `WinInvertLine`, `WinPaintLine`, `WinPaintLines`

WinErasePixel

Purpose Draw a pixel in the draw window using the current background color.

Prototype

```
void WinErasePixel (Coord x, Coord y)
```

Parameters

`-> x`	Pointer to the x coordinate of a pixel.
`-> y`	Pointer to the y coordinate of a pixel.

Result Returns nothing.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawPixel`, `WinInvertPixel`, `WinPaintPixel`, `WinPaintPixels`

WinEraseRectangle

Purpose Draw a rectangle in the draw window using the current background color.

Prototype

```
void WinEraseRectangle (RectangleType *rP,
UInt16 cornerDiam)
```

Parameters

`-> rP`	Pointer to the rectangle to erase.
`-> cornerDiam`	Radius of rounded corners. Specify zero for square corners.

Result Returns nothing.

Comments The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

See Also `WinDrawRectangle`, `WinFillRectangle`, `WinInvertRectangle`, `WinPaintRectangle`

WinEraseRectangleFrame

Purpose Draw a rectangular frame in the draw window using the current background color.

Prototype

```
void WinEraseRectangleFrame (FrameType frame,
RectangleType *rP)
```

Parameters

`-> frame`	Type of frame to draw (see `FrameType`).
`-> rP`	Pointer to the rectangle to frame.

Result Returns nothing.

See Also `WinDrawGrayRectangleFrame`, `WinDrawRectangleFrame`, `WinGetFramesRectangle`, `WinInvertRectangleFrame`, `WinPaintRectangleFrame`

WinEraseWindow

Purpose Erase the contents of the draw window.

Prototype

```
void WinEraseWindow (void)
```

Parameters None.

Result Returns nothing.

Comments `WinEraseRectangle` is used to erase the window. This routine doesn't erase the frame around the draw window. See `WinEraseRectangleFrame` and `WinGetWindowFrameRect`.

WinFillLine

Purpose Fill a line in the draw window with the current pattern.

Prototype

```
void WinFillLine (Coord x1, Coord y1, Coord x2,
Coord y2)
```

Parameters

`-> x1`	x coordinate of line start point.
`-> y1`	y coordinate of line start point.
`-> x2`	x coordinate of line endpoint.
`-> y2`	y coordinate of line endpoint.

Result Returns nothing.

Comments You can set the current pattern with `WinSetPattern`.

See Also `WinDrawGrayLine`, `WinDrawLine`, `WinEraseLine`, `WinInvertLine`, `WinPaintLine`, `WinPaintLines`

WinFillRectangle

Purpose Draw a rectangle in the draw window with current pattern.

Prototype

```
void WinFillRectangle (RectangleType *rP,
UInt16 cornerDiam)
```

Parameters

`-> rP`	Pointer to the rectangle to draw.
`-> cornerDiam`	Radius of rounded corners. Specify zero for square corners.

Result Returns nothing.

Comments You can set the current pattern with `WinSetPattern`.

The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

See Also `WinDrawRectangle`, `WinEraseRectangle`, `WinInvertRectangle`, `WinPaintRectangle`

WinGetActiveWindow

Purpose Return the window handle of the active window.

Prototype

```
WinHandle WinGetActiveWindow (void)
```

Parameters None.

Result Returns the handle of the active window. All user input is directed to the active window.

See Also `WinSetActiveWindow`, `WinGetDisplayWindow`, `WinGetFirstWindow`, `WinGetDrawWindow`

WinGetBitmap

Purpose Return a pointer to a window's bitmap, which holds the window contents.

Prototype

```
BitmapType *WinGetBitmap (WinHandle winHandle)
```

Parameters

`-> winHandle`	Handle of window from which to get the bitmap.

Result Returns a pointer to the bitmap or `NULL` if `winHandle` is invalid.

Comments For onscreen windows, the bitmap returned always represents the whole screen. Thus, the top-left corner of the returned bitmap may not be the top-left corner of the window.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinGetClip

Purpose Return the clipping rectangle of the draw window.

Prototype
```
void WinGetClip (RectangleType *rP)
```

Parameters

`<- rP`	Pointer to a structure to hold the clipping bounds.

Result Returns nothing.

See Also WinSetClip

WinGetDisplayExtent

Purpose Return the width and height of the display (the screen).

Prototype
```
void WinGetDisplayExtent (Coord *extentX,
Coord *extentY)
```

Parameters

`<- extentX`	Pointer to the width of the display in pixels.
`<- extentY`	Pointer to the height of the display in pixels.

Result Returns nothing.

WinGetDisplayWindow

Purpose Return the window handle of the display (screen) window.

Prototype
```
WinHandle WinGetDisplayWindow (void)
```

Parameters None.

Result Returns the handle of display window.

Comments The display window is created by the system at start-up; it has the same size as the Palm OS drawable area of the physical display (screen).

See Also `WinGetDisplayExtent`, `WinGetActiveWindow`, `WinGetDrawWindow`

WinGetDrawWindow

Purpose Return the window handle of the current draw window.

Prototype
```
WinHandle WinGetDrawWindow (void)
```

Parameters None.

Result Returns handle of draw window.

See Also `WinGetDisplayWindow`, `WinGetActiveWindow`, `WinSetDrawWindow`

WinGetFirstWindow

Purpose Return a pointer to the first window in the linked list of windows.

Prototype
```
WinHandle WinGetFirstWindow (void)
```

Parameters None.

Result Returns handle of first window.

Comments This function is usually used by the system only.

See Also `WinGetActiveWindow`

WinGetFramesRectangle

Purpose Return the rectangle that includes a rectangle together with the specified frame around it.

Prototype

```
void WinGetFramesRectangle (FrameType frame,
RectangleType *rP, RectangleType *obscuredRectP)
```

Parameters

`-> frame` Type of rectangle frame (see `FrameType`).

`-> rP` Pointer to the rectangle to frame.

`<- obscuredRectP` Pointer to the rectangle that includes both the specified rectangle and its frame.

Result Returns nothing.

Comments Frames are always drawn around (outside) a rectangle.

See Also `WinGetWindowFrameRect`, `WinGetWindowBounds`

WinGetPattern

Purpose Return the current fill pattern.

Prototype

```
void WinGetPattern (CustomPatternType *patternP)
```

Parameters

`<- patternP` Buffer where the current pattern is returned (see `CustomPatternType`).

Result Returns nothing.

Comments The fill pattern is used by `WinFillLine` and `WinFillRectangle`.

This function returns the value of `patternData` in the current drawing state. (See `DrawStateType`.) The `patternData` field is only set if the pattern field is `customPattern`. Therefore, it's a

good idea to use WinGetPatternType instead of this function on systems that support WinGetPatternType.

See Also WinSetPattern

WinGetPatternType

Purpose Return the current pattern type.

Prototype

```
PatternType WinGetPatternType (void)
```

Parameters None.

Result Returns the current draw window pattern type (see PatternType). If the return value is `customPattern`, you can retrieve the pattern with WinGetPattern.

Comments The fill pattern is used by WinFillLine and WinFillRectangle.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinSetPatternType

WinGetPixel

Purpose Return the color value of a pixel in the current draw window.

Prototype

```
IndexedColorType WinGetPixel (Coord x, Coord y)
```

Parameters

`-> x`	Pointer to the x coordinate of a pixel.
`-> y`	Pointer to the y coordinate of a pixel.

Result Returns the indexed color value of the pixel. See IndexedColorType. A return value of 0 means either that the coordinates do not lie in the current draw window or that they do and the color of that pixel is index 0 (typically white).

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinIndexToRGB

WinGetWindowBounds

Purpose Return the bounds of the current draw window in display-relative coordinates.

Prototype

```
void WinGetWindowBounds (RectangleType *rP)
```

Parameters

`<- rP`	Pointer to a rectangle.

Result Returns nothing.

See Also WinGetWindowExtent, WinSetWindowBounds

WinGetWindowExtent

Purpose Return the width and height of the current draw window.

Prototype

```
void WinGetWindowExtent (Coord *extentX,
Coord *extentY)
```

Parameters

`<- extentX`	Pointer to the width in pixels of the draw window.
`<- extentY`	Pointer to the height in pixels of the draw window.

Result Returns nothing.

See Also WinGetWindowBounds, WinGetWindowFrameRect,

WinGetWindowFrameRect

Purpose Return a rectangle, in display-relative coordinates, that defines the size and location of a window and its frame.

Prototype

```
void WinGetWindowFrameRect (WinHandle winHandle,
RectangleType *r)
```

Parameters

`-> winHandle`	Handle of window whose coordinates are desired.
`<- r`	Pointer to the coordinates of the window.

Result Returns nothing.

See Also WinGetWindowBounds

WinIndexToRGB

Purpose Convert an index in the currently active color table to an RGB value.

Prototype

```
void WinIndexToRGB (IndexedColorType i,
RGBColorType *rgbP)
```

Parameters

`-> i`	A color index value. See IndexedColorType.
`<- rgbP`	Pointer to an RGB color value corresponding to the index value `i`. See RGBColorType.

Result Returns nothing.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinRGBToIndex

WinInvertChars

Purpose Invert the specified characters in the draw window.

Prototype

```
void WinInvertChars (const Char *chars, Int16 len,
Coord x, Coord y)
```

Parameters

`-> chars`	Pointer to the characters to invert.
`-> len`	Length in bytes of the characters to invert.
`-> x`	x coordinate of the first character to invert (left bound).
`-> y`	y coordinate of the first character to invert (top bound).

Result Returns nothing.

Comments This function applies the `winInvert` operation of `WinDrawOperation` to the characters in the draw window.

To perform color inverting, use `WinSetDrawMode` to set the current draw mode to `winSwap`, and then use `WinPaintChars` to draw the characters.

See Also `WinDrawChar`, `WinDrawChars`, `WinDrawInvertedChars`, `WinDrawTruncChars`, `WinEraseChars`, `WinPaintChar`, `WinPaintChars`

WinInvertLine

Purpose Invert a line in the draw window (using the `WinDrawOperation` `winInvert`).

Prototype

```
void WinInvertLine (Coord x1, Coord y1, Coord x2,
Coord y2)
```

Parameters

`-> x1`	x coordinate of line start point.
`-> y1`	y coordinate of line start point.

`-> x2`	x coordinate of line endpoint.
`-> y2`	y coordinate of line endpoint.

Result Returns nothing.

See Also `WinDrawGrayLine`, `WinDrawLine`, `WinEraseLine`, `WinFillLine`, `WinPaintLine`, `WinPaintLines`

WinInvertPixel

Purpose Invert a pixel in the draw window (using the `WinDrawOperation` `winInvert`).

Prototype

```
void WinInvertPixel (Coord x, Coord y)
```

Parameters

`-> x`	Pointer to the x coordinate of a pixel.
`-> y`	Pointer to the y coordinate of a pixel.

Result Returns nothing.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawPixel`, `WinErasePixel`, `WinPaintPixel`, `WinPaintPixels`

WinInvertRectangle

Purpose Invert a rectangle in the draw window (using the `WinDrawOperation` `winInvert`).

Prototype

```
void WinInvertRectangle (RectangleType *rP,
UInt16 cornerDiam)
```

Parameters

`-> rP`	Pointer to the rectangle to invert.

`-> cornerDiam`	Radius of rounded corners. Specify zero for square corners.

Result Returns nothing.

Comments The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

The operating system itself does not use the inverting routines. Instead, it uses the `winSwap` transfer mode, or it changes the color selection and uses the `WinPaint...` routines.

See Also `WinDrawRectangle`, `WinEraseRectangle`, `WinFillRectangle`, `WinPaintRectangle`

WinInvertRectangleFrame

Purpose Invert a rectangular frame in the draw window (using the `WinDrawOperation` `winInvert`).

Prototype

```
void WinInvertRectangleFrame (FrameType frame,
RectangleType *rP)
```

Parameters

`-> frame`	Type of frame to draw (see `FrameType`).
`-> rP`	Pointer to the rectangle to frame.

Result Returns nothing.

See Also `WinDrawGrayRectangleFrame`, `WinDrawRectangleFrame`, `WinEraseRectangleFrame`, `WinGetFramesRectangle`, `WinPaintRectangleFrame`

WinModal

Purpose Return `true` if the specified window is modal.

Prototype
```
Boolean WinModal (WinHandle winHandle)
```

Parameters

`-> winHandle`	Handle of a window.

Result Returns `true` if the window is modal, otherwise `false`.

Comments A window is modal if it cannot lose the focus.

See Also `FrmAlert`, `FrmCustomAlert`, `FrmDoDialog`

WinPaintBitmap

Purpose Draw a bitmap in the current draw window at the specified coordinates with the current draw mode.

Prototype
```
void WinPaintBitmap (BitmapType *bitmapP, Coord x,
Coord y)
```

Parameters

`-> bitmapP`	Pointer to a bitmap.
`-> x`	The x coordinate of the top-left corner.
`-> y`	The y coordinate of the top-left corner.

Result Returns nothing.

Comments If the bitmap has multiple depths (is a bitmap family), the closest match less than or equal to the current draw window depth is used. If such a bitmap does not exist, the bitmap with the closest match greater than the draw window depth is used.

Using `WinPaintBitmap` is now recommended instead of the previous practice of rendering bitmaps into an offscreen window and then using `WinCopyRectangle` to draw them on screen.

The current draw mode is set by `WinSetDrawMode`.

If the bitmap has its own color table, color conversion to the draw window color table will be applied (on OS 3.5 or later). This color conversion is slow and not recommended. Instead of including a color table in the bitmap, consider using `WinPalette` to change the system color table, draw the bitmap, and then change the system color table back when the bitmap is no longer visible.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawBitmap`, `WinEraseRectangle`

WinPaintChar

Purpose Draw a character in the draw window using the current drawing state.

Prototype

```
void WinPaintChar (WChar theChar, Coord x,
Coord y)
```

Parameters

`-> theChar`	The character to draw. This may be either a single-byte character or a multi-byte character.
`-> x`	x coordinate of the location where the character is to be drawn (left bound).
`-> y`	y coordinate of the location where the character is to be drawn (top bound).

Result Returns nothing.

See Also `WinPaintChar` draws the **on** bits in the text color and the **off** bits in the background color, with underlines (if any) drawn in the foreground color using the current drawing mode.

This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the values you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawChar, WinDrawChars, WinDrawInvertedChars, WinDrawTruncChars, WinEraseChars, WinInvertChars, WinPaintChars

WinPaintChars

Purpose Draw the specified characters in the draw window with the current draw state.

Prototype

```
void WinPaintChars (const Char *chars, Int16 len,
Coord x, Coord y)
```

Parameters

`-> chars`	Pointer to the characters to draw.
`-> len`	Length in bytes of the characters to draw.
`-> x`	x coordinate of the first character to draw (left bound).
`-> y`	y coordinate of the first character to draw (top bound).

Result Returns nothing.

Comments `WinPaintChars` draws the **on** bits in the text color and the **off** bits in the background color, with underlines (if any) drawn in the foreground color using the current drawing mode.

This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Before calling this function, consider calling `WinSetUnderlineMode` and `FntSetFont`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawChar`, `WinDrawChars`, `WinDrawInvertedChars`, `WinDrawTruncChars`, `WinEraseChars`, `WinInvertChars`, `WinPaintChar`

WinPaintLine

Purpose Draw a line in the draw window using the current drawing state.

Prototype

```
void WinPaintLine (Coord x1, Coord y1, Coord x2,
Coord y2)
```

Parameters

`-> x1`	x coordinate of line beginning point.
`-> y1`	y coordinate of line beginning point.
`-> x2`	x coordinate of line endpoint.
`-> y2`	y coordinate of line endpoint.

Result Returns nothing.

Comments This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawLine`, `WinDrawGrayLine`, `WinEraseLine`, `WinFillLine`, `WinInvertLine`, `WinPaintLines`

WinPaintLines

Purpose Draw several lines in the draw window using the current drawing state.

Prototype
```
void WinPaintLines (UInt16 numLines,
WinLineType lines[])
```

Parameters

`-> numLines`	Number of lines to paint.
`-> lines`	Array of lines. See `WinLineType`.

Result Returns nothing.

Comments This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawLine`, `WinDrawGrayLine`, `WinEraseLine`, `WinFillLine`, `WinInvertLine`, `WinPaintLine`

WinPaintPixel

Purpose Render a pixel in the draw window using the current drawing state.

Purpose
```
void WinPaintPixel (Coord x, Coord y)
```

Parameters

`-> x`	Pointer to the x coordinate of a pixel.
`-> y`	Pointer to the y coordinate of a pixel.

Result Returns nothing.

Comments This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to

learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawPixel`, `WinErasePixel`, `WinInvertPixel`, `WinPaintPixels`

WinPaintPixels

Purpose Render several pixels in the draw window using the current drawing state.

Prototype

```
void WinPaintPixels (UInt16 numPoints,
PointType pts[])
```

Parameters

`-> numPoints`	Number of pixels to paint.
`-> pts`	Array of pixels.

Result Returns nothing.

Comments This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawPixel`, `WinErasePixel`, `WinInvertPixel`, `WinPaintPixel`

WinPaintRectangle

Purpose Draw a rectangle in the draw window using the current drawing state.

Prototype

```
void WinPaintRectangle (RectangleType *rP,
UInt16 cornerDiam)
```

Parameters

`-> rP`	Pointer to the rectangle to draw.
`-> cornerDiam`	Radius of rounded corners. Specify zero for square corners.

Result Returns nothing.

Comments The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawRectangle`, `WinEraseRectangle`, `WinFillRectangle`, `WinInvertRectangle`

WinPaintRectangleFrame

Purpose Draw a rectangular frame in the draw window using the current drawing state.

Prototype

```
void WinPaintRectangleFrame (FrameType frame,
RectangleType *rP)
```

Parameters

`-> frame`	Type of frame to draw (see `FrameType`).

`-> rP` Pointer to the rectangle to frame.

Result Returns nothing.

Comments The frame is drawn outside the specified rectangle.

This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinDrawGrayRectangleFrame`, `WinDrawRectangleFrame`, `WinEraseRectangleFrame`, `WinGetFramesRectangle`, `WinInvertRectangleFrame`

WinPalette

Purpose Set or retrieve the palette for the draw window.

Prototype

```
Err WinPalette (UInt8 operation, Int16 startIndex,
UInt16 paletteEntries, RGBColorType *tableP)
```

Parameters `-> operation` Specify one of the following values:

`winPaletteGet`
Retrieve the palette. Entries are read from the palette beginning at `startIndex` and placed into `tableP` beginning at index 0.

`winPaletteSet`
Set the palette. Entries from `tableP` (beginning at index 0) are set into the palette beginning at `startIndex` in the palette.

`winPaletteSetToDefault`
Set the palette to the default system palette.

`-> startIndex` Identifies where in the palette to start reading or writing. Specify `WinUseTableIndexes` to indicate that the entries are not to be set or read sequentially; instead, the `index` value in each `RGBColorType` entry in `tableP` determines which slot in the palette is to be set or read. You can use this technique to get or set several discontiguous palette entries with a single function call.

`-> paletteEntries`
Number of palette entries to get or set.

`<-> tableP` A pointer to a buffer of `RGBColorType` entries that is either read from or written to, depending on the `operation` parameter; the table entries from 0 to `paletteEntries` – 1 are affected by this routine.

Result Returns one of the following values:

`errNone`	Success.
`winErrPalette`	The current draw window does not have a color table, a set operation has overflowed the color table, or one of the entries in `tableP` has an invalid index value
`sysErrParamErr`	The `startIndex` value is invalid.

Comments Here are some examples of how this routine works:

- If `startIndex` is 0 and `paletteEntries` is 10, the first 10 elements of the palette will be set from `tableP` or will be copied into `tableP`.
- If `startIndex` is 10 and `paletteEntries` is 5, then entries 10, 11, 12, 13, and 14 in the palette will be set from or copied to elements 0, 1, 2, 3, and 4 in `tableP`.
- If `startIndex` is `WinUseTableIndexes` and `paletteEntries` is 1, then the index value in the `RGBColorType` of element 0 of `tableP` will be read from or copied to `tableP`; in this case, the `index` field of the `RGBColorType` will not change.

During a set operation, this function broadcasts the `sysNotifyDisplayChangeEvent` to notify any interested observer that the color palette has changed. For information on this and other notifications, see Chapter 36, "Notification Manager."

One use for this function is if you need to display a bitmap that uses a color table other than the one in use by the system. You can attach a custom color table to a bitmap, and if you do, the bitmap is drawn using that color table. However, this is a performance drain. As an optimization, you can use `WinPalette` to change the system color table to match that used by the bitmap, display the bitmap, and use `WinPalette` to reset the color table when the bitmap is no longer visible.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinPopDrawState

Purpose Restore the draw state values to the last saved set on the stack.

Prototype

```
void WinPopDrawState (void)
```

Parameters None.

Result Returns nothing.

Comments Use this routine to restore the draw state saved by the previous call to `WinPushDrawState`.

After you call this function, the current draw window's `drawStateP` field points to the restored drawing state.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinPushDrawState

Purpose Save the current draw state values onto the draw state stack.

Prototype
```
void WinPushDrawState (void)
```

Parameters None.

Result Returns nothing.

Comments Use this routine to save the current draw state before making changes to it using the functions listed in the `DrawStateType` structure's description. Call `WinPopDrawState` to restore the saved settings.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinResetClip

Purpose Reset the clipping rectangle of the draw window to the portion of the draw window that is within the bounds of the display.

Prototype
```
void WinResetClip (void)
```

Parameters None.

Result Returns nothing.

See Also `WinSetClip`

WinRestoreBits

Purpose Copy the contents of the specified window to the draw window and delete the passed window.

Prototype

```
void WinRestoreBits (WinHandle winHandle,
Coord destX, Coord destY)
```

Parameters

`-> winHandle`	Handle of window to copy and delete.
`-> destX`	x coordinate in the draw window to copy to.
`-> destY`	y coordinate in the draw window to copy to.

Result Returns nothing.

Comments This routine is generally used to restore a region of the display that was saved with `WinSaveBits`.

See Also `WinSaveBits`

WinRGBToIndex

Purpose Convert an RGB value to the index of the closest color in the currently active color lookup table (CLUT).

Prototype

```
IndexedColorType WinRGBToIndex
(const RGBColorType *rgbP)
```

Parameters

`-> rgbP`	Pointer to an RGB color value.

Result Returns the index of the closest matching color in the CLUT.

Comments Palm OS 3.5 supports a maximum of 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, an exact match may not be available for `rgbP`. If there is no exact RGB match, then a luminance best-fit is used if the color lookup table is entirely gray scale (red, green, and blue values for each entry are identical), or a shortest-distance fit in RGB space is used if the

palette contains colors. RGB shortest distance may not always produce the actual closest perceptible color, but it's relatively fast and works for the system palette.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinIndexToRGB`

WinSaveBits

Purpose Create an offscreen window and copy the specified region from the draw window to the offscreen window.

Prototype

```
WinHandle WinSaveBits (RectangleType *sourceP,
UInt16 *error)
```

Parameters

`-> sourceP`	Pointer to the bounds of the region to save, relative to the display.
`<- error`	Pointer to any error encountered by this function.

Result Returns the handle of the window containing the saved image, or zero if an error occurred.

Comments The offscreen window is the same size as the region to copy.

This function tries to copy the window's bitmap using compressed format if possible. It may display a fatal error message if an error occurs when it tries to shrink the pointer for the compressed bits.

See Also `WinRestoreBits`

WinScreenLock

Purpose "Lock" the current screen by switching the UI concept of the screen base address to an area that is not reflected on the display.

Prototype

```
UInt8* WinScreenLock (WinLockInitType initMode)
```

Parameters

`-> initMode` Indicates how to initialize the new screen area. Specify one of the following values:

`winLockCopy`
Copy old screen to new.

`winLockErase`
Erase new screen to white.

`winLockDontCare`
Don't do anything

Result Returns a pointer to the new screen base address, or `NULL` if this routine fails.

Comments This routine can be used to "freeze" the display while doing lengthy drawing operations to avoid a flickering effect. Call `WinScreenUnlock` to unlock the display and cause it to be updated with any changes. The screen must be unlocked as many times as it is locked to actually update the display.

Because this function copies the screen, using it is a relatively expensive operation.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinScreenMode

Purpose Sets or returns display parameters, including display geometry, bit depth, and color support.

Prototype

```
Err WinScreenMode
(WinScreenModeOperation operation, UInt32 *widthP,
UInt32 *heightP, UInt32 *depthP,
Boolean *enableColorP)
```

Parameters The `widthP`, `heightP`, `depthP`, and `enableColorP` parameters are used in different ways for different operations. See Comments at the end of this description for details.

`-> operation` The work this function is to perform, as specified by one of the following selectors:

`winScreenModeGet`
Return the current settings for the display.

`winScreenModeGetDefaults`
Return the default settings for the display.

`winScreenModeGetSupportedDepths`
Return in `depthP` a hexadecimal value indicating the supported screen depths. The binary representation of this value defines a bitfield in which the value 1 indicates support for a particular display depth. The position representing a particular bit depth corresponds to the value $2^{(bitDepth-1)}$. See the Example at the end of this function description for more information.

`winScreenModeGetSupportsColor`
Return `true` as the value of the `enableColorP` parameter when color mode can be enabled.

`winScreenModeSet`
Change display settings to the values specified by the other arguments to the `WinScreenMode` function.

`winScreenModeSetToDefaults`
Change display settings to default values.

`<-> widthP` Pointer to new/old screen width.

`<-> heightP` Pointer to new/old screen height.

`<-> depthP` Pointer to new/old/available screen depth.

`<-> enableColorP`
Pass `true` to enable color drawing mode.

Result If no error, returns values as specified by the `operation` argument. Various invalid arguments may cause this function to return a `sysErrParamErr` result code. In rare cases, a failed allocation can cause this function to return a `memErrNotEnoughSpace` error.

Comments The `widthP`, `heightP`, `depthP`, and `enableColorP` parameters are used in different ways for different operations. All "get" operations overwrite these values with a result when the function returns. The `winScreenModeSet` operation changes current display parameters when passed valid argument values that are not `NULL` pointers. The `winScreenModeSetToDefaults` operation ignores values passed for all of these parameters.

Table 48.1 summarizes parameter usage for each operation this function performs.

Table 48.1 Use of parameters to WinScreenMode function

Operation `winScreenMode...`	widthP	heightP	depthP	enableColorP
...`Get`	returned	returned	returned	returned
...`GetDefaults`	returned	returned	returned	returned
...`GetSupportedDepths`	pass in	pass in	returned	pass in
...`GetSupportsColor`	pass in	pass in	pass in	returned
...`Set`	pass in	pass in	pass in	pass in
...`SetToDefaults`	ignored	ignored	ignored	ignored

This function ignores `NULL` pointer arguments to the `widthP`, `heightP`, `depthP`, and `enableColorP` parameters; thus, you can

pass a NULL pointer for any of these values to leave the current value unchanged. Similarly, when getting values, this function does not return a value for any NULL pointer argument.

If you change the display depth, it is recommended that you restore it to its previous state when your application closes, even though the system sets display parameters back to their default values when launching an application.

Note that none of the other operations interprets the depth parameter the same way that winScreenModeGetSupportedDepths does. For example, to set the display depth to 8-bit mode, you use 8 (decimal) for the display depth, not 0x08 (128 decimal).

Compatibility

Implemented only if 3.5 New Feature Set is present. In OS versions prior to 3.5, this function is called ScrDisplayMode. The prototype for ScrDisplayMode is similar to WinScreenMode:

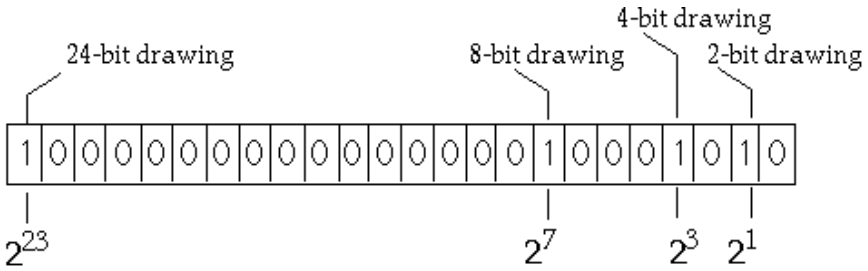
```
Err ScrDisplayMode (
ScrDisplayModeOperation operation,
DWordPtr widthP, DWordPtr heightP,
DWordPtr depthP, BooleanPtr enableColorP)
```

The only other difference between ScrDisplayMode and WinScreenMode is that the ScrDisplayModeOperation constants begin with the prefix scrDisplayMode rather than winScreenMode.

Example

Here are some additional examples of return values provided by the winScreenModeGetSupportedDepths mode of the WinScreenMode function.

This function indicates support for 4-bit drawing by returning a value of 0x08, or 2^3, which corresponds to a binary value of 1000. Support for bit depths of 2 and 1 is indicated by a return value of 0x03. Support for bit depths of 4, 2, and 1 is indicated by 0x0B, which is a binary value of 1011. Support for bit depths of 24, 8, 4 and 2 is indicated by 0x80008A. The figure immediately following depicts this final example graphically.



Bit depth support indicated by interpreting `0x80008A` as binary value

WinScreenUnlock

Purpose Unlock the screen and update the display.

Prototype
```
void WinScreenUnlock (void)
```

Parameters None.

Result Returns nothing.

Comments The screen must be unlocked as many times as it is locked to actually update the display.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinScreenLock`

WinScrollRectangle

Purpose Scroll a rectangle in the draw window.

Prototype
```
void WinScrollRectangle (RectangleType *rP,
WinDirectionType direction, Coord distance,
RectangleType *vacatedP)
```

Parameters

`-> rP`	Pointer to the rectangle to scroll.

`-> direction`	Direction to scroll (`winUp`, `winDown`, `winLeft`, or `winRight`).
`-> distance`	Distance to scroll in pixels.
`<- vacatedP`	Pointer to the rectangle that needs to be redrawn because it has been vacated as a result of the scroll.

Result Returns nothing.

Comments The rectangle scrolls within its own bounds. Any portion of the rectangle that is scrolled outside its bounds is clipped.

WinSetActiveWindow

Purpose Make a window the active window.

Prototype

```
void WinSetActiveWindow (WinHandle winHandle)
```

Parameters

`-> winHandle`	Handle of a window.

Result Returns nothing.

Comments The active window is not actually set in this routine; flags are set to indicate that a window is being exited and another window is being entered. The routine `EvtGetEvent` sends a `winExitEvent` and a `winEnterEvent` when it detects these flags. The active window is set by `EvtGetEvent` when it sends the `winEnterEvent`. The draw window is also set to the new active window when the active window is changed.

The window is enabled before it is made active.

All user input is directed to the active window.

See Also `WinGetActiveWindow`, `EvtGetEvent`

WinSetBackColor

Purpose Set the background color to use in subsequent draw operations.

Prototype

```
IndexedColorType WinSetBackColor
(IndexedColorType backColor)
```

Parameters

`-> backColor`	Color to set; specify a value of type `IndexedColorType`.

Result Returns the previous background color index.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

To set the foreground color to a predefined UI color default, use `UIColorGetTableEntryIndex` as an input to this function. For example:

```
curColor = WinSetBackColor
(UIColorGetTableEntryIndex(UIFieldBackground));
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinSetForeColor`, `WinSetTextColor`

WinSetClip

Purpose Set the clipping rectangle of the draw window.

Prototype

```
void WinSetClip (RectangleType *rP)
```

Parameters

`-> rP`	Pointer to a structure holding the clipping bounds.

Result Returns nothing.

See Also `WinClipRectangle`, `WinSetClip`, `WinGetClip`

WinSetDrawMode

Purpose Set the transfer mode to use in subsequent draw operations.

Prototype
```
WinDrawOperation WinSetDrawMode
(WinDrawOperation newMode)
```

Parameters

`-> newMode`	Transfer mode to set; specify one of the `WinDrawOperation` values.

Result Returns the previous transfer mode.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinSetDrawWindow

Purpose Set the draw window. (All drawing operations are relative to the draw window.)

Prototype
```
WinHandle WinSetDrawWindow (WinHandle winHandle)
```

Parameters

`-> winHandle`	Handle of a window.

Result Returns the previous draw window.

Compatibility OS versions before 3.5 allowed you to use `NULL` as a parameter to this function to set the draw window to the display window (or screen window). In version 3.5 and higher, this practice is discouraged. If `winHandle` is `NULL`, the debug ROM sets the draw window to `badDrawWindowValue`, and you are warned if you try to draw to it.

See Also `WinGetDrawWindow`, `WinSetActiveWindow`

WinSetForeColor

Purpose Set the foreground color to use in subsequent draw operations.

Prototype

```
IndexedColorType WinSetForeColor
(IndexedColorType foreColor)
```

Parameters

`-> foreColor`	Color to set; specify a value of type `IndexedColorType`.

Result Returns the previous foreground color index.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

To set the foreground color to a predefined UI color default, use `UIColorGetTableEntryIndex` as an input to this function. For example:

```
curColor = WinSetForeColor
  (UIColorGetTableEntryIndex
  (UIObjectForeground));
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinSetBackColor`, `WinSetTextColor`

WinSetPattern

Purpose Set the current fill pattern.

Prototype

```
void WinSetPattern (const CustomPatternType
*patternP)
```

Parameters

`-> patternP`	Pattern to set (see `CustomPatternType`).

Result Returns nothing.

Comments The fill pattern is used by `WinFillLine` and `WinFillRectangle`.

This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

See Also `WinGetPattern`

WinSetPatternType

Purpose Set the current pattern type.

Prototype

```
void WinSetPatternType (PatternType newPattern)
```

Parameters

`-> newPattern`	Pattern type to set for the draw window (see `PatternType`).

Result Returns nothing.

Comments This function sets the `pattern` field of the drawing state to `newPattern` and sets the `patternData` field to `NULL`. To set `patternData` to a custom pattern use `WinSetPattern`.

The fill pattern is used by `WinFillLine` and `WinFillRectangle`.

This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinGetPatternType`

WinSetTextColor

Purpose Set the color to use for drawing characters in subsequent draw operations.

Prototype

```
IndexedColorType WinSetTextColor
(IndexedColorType textColor)
```

Parameters

`-> textColor`	Color to set; specify a value of type `IndexedColorType`.

Result Returns the previous text color index.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

To set the foreground color to a predefined UI color default, use `UIColorGetTableEntryIndex` as an input to this function. For example:

```
curColor = WinSetTextColor
  (UIColorGetTableEntryIndex(UIFieldText));
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinSetBackColor`, `WinSetForeColor`

WinSetUnderlineMode

Purpose Set the graphic state to enable or disable the underlining of characters.

Prototype

```
UnderlineModeType WinSetUnderlineMode
(UnderlineModeType mode)
```

Parameters

`<-> mode`	New underline mode type; see `UnderlineModeType`.

Result Returns the previous underline mode type.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

See Also `WinDrawChars`

WinSetWindowBounds

Purpose Set the bounds of the window to display-relative coordinates.

Prototype

```
void WinSetWindowBounds (WinHandle winHandle,
RectangleType *rP)
```

Parameters

`-> winHandle`	Handle for the window for which to set the bounds.
`-> rP`	Pointer to a rectangle to use for bounds.

Result Returns nothing.

Comments A visible window cannot have its bounds modified.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `WinGetWindowBounds`

WinValidateHandle

Purpose Return `true` if the specified handle references a valid window object.

Prototype

```
Boolean WinValidateHandle (WinHandle winHandle)
```

Parameters

`-> winHandle` The handle to be tested.

Result Returns `true` if the specified handle references a non-`NULL` pointer to a window in the active window list, `false` if the handle references a window whose values are out of sync with the current system state.

Comments For debugging purposes only. Do not include this function in commercial applications.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmValidatePtr`, `FrmRemoveObject`

WinWindowToDisplayPt

Purpose Convert a window-relative coordinate to a display-relative coordinate.

Prototype

```
void WinWindowToDisplayPt (Coord *extentX,
Coord *extentY)
```

Parameters

`<-> extentX` Pointer to x coordinate to convert.

`<-> extentY` Pointer to y coordinate to convert.

Result Returns nothing.

Comments The coordinate passed is assumed to be relative to the draw window.

See Also `WinDisplayToWindowPt`

49

Miscellaneous System Functions

This chapter describes miscellaneous system functions. The functions in this chapter are declared in the header files `Crc.h`, `IntlMgr.h`, and `Localize.h`.

Crc16CalcBlock

Purpose Calculate the 16-bit CRC of a data block using the table lookup method.

Prototype

```
UInt16 Crc16CalcBlock (const void *bufP,
UInt16 count, UInt16 crc)
```

Parameters

`bufP`	Pointer to the data buffer.
`count`	Number of bytes in the buffer.
`crc`	Seed crc value.

Result A 16-bit CRC for the data buffer.

IntlGetRoutineAddress

Purpose Return the address of an international manager or text manager function.

Prototype

```
void *IntlGetRoutineAddress
(IntlSelector inSelector)
```

Parameters

`-> inSelector` One of the routine selectors defined in `IntlMgr.h`.

Result Returns the address of the corresponding function. Returns `NULL` if an invalid routine selector is passed.

Comments Use this function for performance reasons. It returns the address of an international manager or text manager function. You can then use this address to call the function without having to go through the international manager's trap dispatch table. This function is mostly useful for optimizing the performance of text manager routines that are called in a tight loop.

Compatibility Implemented only if International Feature Set is present.

LocGetNumberSeparators

Purpose Get localized number separators.

Prototype

```
void LocGetNumberSeparators
(NumberFormatType numberFormat,
Char *thousandSeparator, Char *decimalSeparator)
```

Parameters

`numberFormat` The format to use

`thousandSeparator` Return a localized thousand separator here (allocate 1 char).

`decimalSeparator`
Return a localized decimal separator here (allocate 1 char).

Result Returns nothing.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrLocalizeNumber`, `StrDelocalizeNumber`, "Localized Applications" in the *Palm OS Programmer's Companion*

Part III: Communications

50

Connection Manager

The connection manager allows other applications to access, add, and delete connection profiles contained in the Connection Panel.

This chapter provides reference material for the connection manager API:

- Connection Manager Functions

The header file ConnectionMgr.h declares the connection manager API. For more information on the connection manager, see the chapter Serial Communication in the *Palm OS Programmer's Companion*.

Connection Manager Functions

CncAddProfile

Purpose Adds a profile to the connection manager.

Prototype

```
Err CncAddProfile(Char *name, UInt32 port,
UInt32 baud, UInt16 volume, UInt16 handShake,
Char *initString, Char *resetString,
Boolean isModem, Boolean isPulse)
```

Parameters

<-> `name`	Pointer to the profile name to be added. If the name is already taken in the Connection Panel then a duplication number is appended to it. The name added is returned here.
-> `port`	The port identification used by the profile.
-> `baud`	The baud rate used by the profile.
-> `volume`	The volume setting for the device (for Modem only).

-> `handShake`	Flow control setting (hardware handshaking). 0 specifies automatic (on at speeds > 2400 baud), 1 specifies always on, and 2 specifies always off.
-> `initString`	Pointer to the initialization string used by a modem (for Modem only).
-> `resetString`	Pointer to the reset string used by a modem (for Modem only).
-> `isModem`	`true` if Modem, `false` if Direct.
-> `isPulse`	true if Pulse dial, `false` if TouchTone.

Result

0	No error.
`cncErrAddProfileFailed`	The add operation failed.
`cncErrProfileListFull`	The add operation failed because the profile list is full.
`cncErrConDBNotFound`	The connection database is missing.

Comments

All profiles within the connection manager must have a unique name. The connection manager tries to append a duplication number to the end of the name if you specify a name that is already taken.

There is a maximum limit to the number of profiles that can be maintained by the connection manager. If the limit is passed, an error is returned and that profile will not be added. Profiles that do not need certain fields may pass 0 in the place of a value.

Compatibility

Implemented only if New Serial Manager Feature Set is present.

Example

```
AddMyProfile()
{
  Char *myConNameP;
  Err err;

  myConNameP = MemPtrNew(cncProfileNameSize);
```

```
  StrCopy(myConNameP, "Foobar");

  err = CncAddProfile(myConNameP, 'u328',
57600, 0, 0, "AT&FX4", 0,  true, false);

  MemPtrFree(myConNameP);
}
```

CncDeleteProfile

Purpose Removes a profile from the connection manager.

Prototype

```
Err CncDeleteProfile(Char *name)
```

Parameters

-> `name`	Pointer to the name of the profile to be deleted.

Result

0	No error.
`cncErrProfileReadOnly`	The profile could not be deleted because it is read only.
`cncErrProfileNotFound`	The profile could not be found
`cncErrConDBNotFound`	The connection database is missing.

Comments The profiles that come preinstalled on the unit are read only and cannot be deleted.

Compatibility Implemented only if New Serial Manager Feature Set is present.

Example

```
void DeleteProfile(Char *name)
  {
    Err err;
    //Call Connection Manager to delete the
    //named profile
    err = CncDeleteProfile(name);
  }
```

CncGetProfileInfo

Purpose Returns the settings for a profile.

Prototype

```
Err CncGetProfileInfo(Char *name, UInt32 *port,
UInt32 *baud, UInt16 *volume, UInt16 *handShake,
Char *initString, Char *resetString,
Boolean * isModem, Boolean * isPulse)
```

Parameters

-> `name`	Pointer to the name of the profile to be returned. Passing in `NULL` causes this function to return the settings for the profile currently selected in the Connection Panel.
<- `port`	Pointer to the port identifier that the profile uses.
<- `baud`	Pointer to the baud rate that has been set for this profile.
<- `volume`	Pointer to the volume of the device (applies only to modems).
<- `handShake`	Pointer to the flow control setting (hardware handshaking). 0 indicates automatic (on at speeds > 2400 baud), 1 indicates always on, and 2 indicates always off.
<- `initString`	Pointer to the initialization string for the device (applies only to modems).
<- `resetString`	Pointer to the reset string for the device (applies only to modems).
<- `isModem`	Pointer to a Boolean value: `true` for Modem, `false` for Direct.
<- `isPulse`	Pointer to a Boolean value: `true` for Pulse dial, `false` for TouchTone.

Result

0	No error.
`cncErrGetProfileFailed`	The get profile operation failed. The profile may or may not be there.
`cncErrProfileNotFound`	The profile could not be found
`cncErrConDBNotFound`	The connection database is missing.

Comments One or more of the parameters may be set to `NULL` if that information is not desired.

Compatibility Implemented only if New Serial Manager Feature Set is present.

Example

```
    {
      UInt32 portID, baud;
      UInt16 openPort;
      // get port id
      err = CncGetProfileInfo("Direct Serial",
&portID, &baud, 0, 0, 0, 0, 0, 0);
      if(!err)
      { // open the port
        SrmOpen(portID, baud, &openPort);
      }
    }
```

CncGetProfileList

Purpose Returns a list of available profiles that are available through the connection manager.

Prototype

```
Err CncGetProfileList(Char *** nameListP,
UInt16 * count)
```

Parameters

<- `nameListP`	Pointer to a pointer to a list of profile names.
<- `count`	Pointer to the number of profile names.

Result

0	No error.
`cncErrGetProfileListFailed`	The profile list could not be found.
`cncErrConDBNotFound`	The connection database is missing.

Comments

Allocation of the list is handled by the connection manager; deallocation is the responsibility of the calling application. Appended to the end of the list will be "-Current-", which represents the profile currently selected in the Connection Panel.

Compatibility

Implemented only if New Serial Manager Feature Set is present.

Example

```
//Declared globally
Char ** globalProfileList;
ListType *listP;
UInt16 globalProfileCount;

void SetConnectionList()
{
//Get the list from the Connection Manager
err = CncGetProfileList(&globalProfileList,
&globalProfileCount);
//Set the UI list
LstSetListChoices(listP, globalProfileList,
globalProfileCount);
}

void StopApplication()
{
UInt16 i;

//Deallocate the connection list
For(i = 0; i < globalProfileCount;  i++)
  MemPtrFree(globalProfileList[ i ]);
MemPtrFree(globalProfileList);
}
```

51

Exchange Manager

This chapter provides reference material for the exchange manager API:

- Exchange Manager Data Structures
- Exchange Manager Functions
- Application-Defined Functions

The header file `ExgMgr.h` declares the exchange manager API. For more information on the exchange manager, see the chapter "Beaming (Infrared Communication)" in the *Palm OS Programmer's Companion*.

Exchange Manager Data Structures

ExgAskResultType

The `ExgAskResultType` enum defines possible values for the `result` field of the `sysAppLaunchCmdExgAskUser` launch code parameter block.

```
typedef enum {
  exgAskDialog,
  exgAskOk,
  exgAskCancel }
ExgAskResultType;
```

Value Descriptions

`exgAskDialog`	The exchange manager should display the exchange dialog to prompt the user to confirm the receipt of data. See `ExgDoDialog`.
`exgAskOk`	Accept the data.
`exgAskCancel`	Reject the data.

ExgGoToType

The `ExgGoToType` structure defines information that is passed to the `sysAppLaunchCmdGoto` launch command, after an item is received. The `ExgGoToPtr` type points to a `ExgGoToType` structure.

```
typedef struct {
  UInt16 dbCardNo;
  LocalID dbID;
  UInt16 recordNum;
  UInt32 uniqueID;
  UInt32 matchCustom;
} ExgGoToType;

typedef ExgGoToType * ExgGoToPtr;
```

Field Descriptions

`dbCardNo`	Card number of the database.
`dbID`	LocalID of the database.
`recordNum`	Index of the record that contains a match.
`uniqueID`	Position in the record of the match.
`matchCustom`	Application-specific information.

ExgSocketType

The `ExgSocketType` structure defines an exchange manager socket. The `ExgSocketPtr` type points to a `ExgSocketType` structure.

```
typedef struct ExgSocketType {
  UInt16 libraryRef;
  UInt32 socketRef;
  UInt32 target;
  UInt32 count;
  UInt32 length;
  UInt32 time;
  UInt32 appData;
  UInt32 goToCreator;
```

```
    ExgGoToType goToParams;
    UInt16 localMode:1;
    UInt16 packetMode:1;
    UInt16 noGoTo:1;
    UInt16 noStatus:1;
    UInt16 reserved:12;
    Char *description;
    Char *type;
    Char *name;
  } ExgSocketType;

  typedef ExgSocketType* ExgSocketPtr;
```

Note that when data is received, some of the fields in this structure may not be filled in. The existing IR library does not send values for the `count`, `time`, `appData`, or `type` fields; however, it may do so in the future. When you are sending data, it is recommended that you provide values for all of these fields, but you should not rely on receiving values for them.

Field Descriptions

`libraryRef`	Identifies the exchange library in use.
`socketRef`	Identifies the connection (used by exchange library).
`target`	Creator ID of the application the data is being sent to.
`count`	Number of objects in this connection, usually 1 (optional).
`length`	Total byte count for all objects being sent (optional).
`time`	Last modified time of object (optional).
`appData`	Application-specific information (optional).
`goToCreator`	Creator ID of the application to launch via the `sysAppLaunchCmdGoto` launch code after the item is received if `noGoTo` is 0.

`goToParams`	If `goToCreator` is specified, then this contains information about where to go. See `ExgGoToType`.
`localMode`	Set to 1 to exchange with local machine only. Set to 0 to enable an exchange with a remote machine. Default is 0.
`packetMode`	Set to 1 to use connectionless packet mode (Ultra). Default is 0.
`noGoTo`	Set to 1 to disable launching the application with `sysAppLaunchCmdGoto`. This flag is only valid if `localMode` is 1. Default is 0.
`noStatus`	This field is not currently used.
`reserved`	Reserved system flags.
`description`	Pointer to text description of object (for user).
`type`	Pointer to Mime type of object (optional).
`name`	Pointer to name of object, generally a file name including extension. If you don't provide a name, the exchange manager sets this field to `Palm.exg`. Because the current IR library does not send the `type` field, the file extension is used to identify the data type. The built-in applications recognize the following extensions: txt — Memo vcf — AddressBook vcs — Datebook and ToDo prc, pdb, pqa — Launcher

Compatibility

The `noGoTo` and `noStatus` flags are only defined if 3.5 New Feature Set is present.

Exchange Manager Functions

ExgAccept

Purpose Accepts a connection from a remote device.

Prototype

```
Err ExgAccept (ExgSocketPtr socketP)
```

Parameters

-> `socketP`	Pointer to the socket structure.

Result Returns the following result codes:

`errNone`	No error
`exgErrBadLibrary`	Couldn't find default exchange library
`exgErrStackInit`	Couldn't initialize the IR stack (not enough battery power or unsupported hardware)

Comments An application calls this function when it has been called with the special application launch code `sysAppLaunchCmdExgReceiveData`. The application is passed `socketP` as a parameter. It should pass this parameter to `ExgAccept` to accept the connection and then call `ExgReceive` one or more times to receive the data.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgReceive`

ExgDBRead

Purpose

Reads a Palm OS® database in its internal format and writes it to storage RAM. For example, this function might read in a database transmitted by a beaming operation using the exchange manager.

Prototype

```
Err ExgDBRead (ExgDBReadProcPtr readProcP,
ExgDBDeleteProcPtr deleteProcP, void* userDataP,
LocalID* dbIDP, UInt16 cardNo,
Boolean* needResetP, Boolean keepDates)
```

Parameters

-> `readProcP`	A pointer to a function that you supply that reads in the database and passes it to `ExgDBRead`. See `ReadProc` for details.
-> `deleteProcP`	A pointer to a function that is called if a database with an identical name already exists on the device, so you can erase it before `ExgDBRead` stores the received database. See `DeleteProc` for details.
-> `userDataP`	A pointer to any data you want to pass to either the `readProcP` or `deleteProcP` functions.
<- `dbIDP`	The ID of the database that `ExgDBRead` created on the local device.
<- `cardNo`	The number of the card on which the database was stored by `ExgDBRead`.
<- `needResetP`	Set to `true` by `ExgDBRead` if the `dmHdrAttrResetAfterInstall` attribute bit is set in the received database.
-> `keepDates`	Specify `true` to retain the creation, modification, and last backup dates as set in the received database header. Specify `false` to reset these dates to the current date.

Result

Returns `errNone` if successful; otherwise, returns one of the data manager error codes (`dmErr...`) or a callback-specific error code (if the `readProcP` function returns an error, it is also returned by `ExgDBRead`).

Comments The read callback function passed in `readProcP` is called multiple times by `ExgDBRead`. Each time, `ExgDBRead` passes in `sizeP` the number of bytes it expects to receive in the next chunk that the read callback function is to return in `dataP`. It's important for the read callback function to set the number of bytes (in `sizeP`) that it actually placed in `dataP`, if it's not the same as what `ExgDBRead` expected. `ExgDBRead` stops calling the read callback function after it receives the entire database (it knows when it's got it all based on the header information).

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgDBWrite`

ExgDBWrite

Purpose Reads a given Palm OS database in its internal format from the local device and writes it out using a function you supply. For example, this function might read a local database and transmit it by a beaming operation using the exchange manager.

Prototype

```
Err ExgDBWrite (ExgDBWriteProcPtr writeProcP,
void* userDataP, const char* nameP, LocalID dbID,
UInt16 cardNo)
```

Parameters

-> `writeProcP`	A pointer to a function that you supply that writes out the database identified by `dbID`. See `WriteProc` for details.
-> `userDataP`	A pointer to any data you want to pass to the `writeProcP` function.
-> `nameP`	A pointer to the name of the database that you want `ExgDBWrite` to read and pass to `writeProcP`.
-> `dbID`	The id of the database that you want `ExgDBWrite` to read and pass to `writeProcP`. If you don't supply an ID, then `nameP` is used to search for the database by name.

-> `cardNo`	The number of the card on which to look for the database identified by `nameP`.

Result Returns `errNone` if successful; otherwise, returns one of the data manager error codes (`dmErr...`) or a callback-specific error code (if the `writeProcP` function returns an error, it is also returned by `ExgDBWrite`).

Comments The `writeProcP` parameter points to a function that you supply and that is called by `ExgDBWrite` to write out a database. For example, you might use this function to call exchange manager functions to beam the database to another unit.

The write callback function is called multiple times by `ExgDBWrite`. In the `sizeP` parameter, `ExgDBWrite` passes the number of bytes in `dataP`. Due to transport errors, timeouts, or other problems, you may not be able to successfully send all this data. If the write callback function didn't handle it all, it's important that it set in `sizeP` the number of bytes that it did handle successfully. `ExgDBWrite` stops calling the write callback function after you write out the entire database (it knows when you've done it all based on the header information and number of bytes you return in `sizeP` each time).

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgDBRead`

ExgDisconnect

Purpose Terminates an exchange manager transfer and disconnects.

Prototype

```
Err ExgDisconnect(ExgSocketPtr socketP, Err error)
```

Parameters

-> `socketP`	Pointer to the socket structure identifying the connection to terminate.
-> `error`	Any application error that occurred.

Result Returns the following result codes:

`errNone`	No error
`exgErrBadLibrary`	Couldn't find default exchange library
`exgMemError`	Couldn't read data to send
`exgErrUserCancel`	User cancelled transfer

Comments

In the `error` parameter, pass any error that occurs during the application loop, including errors returned from other exchange manager functions. This ensures that the connection is shut down knowing that it failed rather than succeeded.

It's especially important to check the result code from this function, since this will tell you if the transfer was successful. An `errNone` return value means that the item was delivered to the destination successfully. It does not mean that the user on the other end actually kept the data.

`ExgDisconnect` is used for sending and receiving. When receiving, the application can insert its creator ID into the `goToCreator` field in the socket structure and add other goto information. After the application returns from the `sysAppLaunchCmdExgReceiveData` call, the system will launch the application with a standard `sysAppLaunchCmdGoto` launch code built from the information in the socket header `gotoParams` field.

Compatibility

Implemented only if 3.0 New Feature Set is present.

See Also

`ExgPut`, `ExgReceive`, `ExgSend`

ExgDoDialog

Purpose Display a dialog that allows users to accept or reject the receipt of data.

Prototype

```
Boolean ExgDoDialog (ExgSocketPtr socketP,
ExgDialogInfoType *infoP, Err *errP)
```

Parameters

-> `socketP`	Pointer to the socket structure identifying the connection. You can obtain this pointer from the `sysAppLaunchCmdExgAskUser` launch code parameter block.
<-> `infoP`	A pointer to an `ExgDialogInfoType` structure (see the "Comment" section below).
<- `errP`	`errNone` if no error, or the error code if an error occurred. Currently, no errors are returned.

Result Returns `true` if the user clicks the OK button on the dialog, or `false` otherwise.

Comments This function displays the exchange dialog, which prompts the user to accept or reject incoming data.

By default, the exchange manager calls this function for you if you don't handle the `sysAppLaunchCmdExgAskUser` launch code or if you return `exgAskDialog` from the launch code handler. When the exchange manager calls `ExgDoDialog`, the dialog only displays a message similar to "Do you want to accept 'John Doe' into AddressBook?" and allows the user to accept or reject the data. If the user clicks OK, the data should be received as an unfiled record.

To allow users to select a category when accepting incoming data, handle `sysAppLaunchCmdExgAskUser` to call `ExgDoDialog` explicitly, and pass it a pointer to an `ExgDialogInfoType` structure. The `ExgDialogInfoType` structure is defined as follows:

```
typedef struct {
  UInt16    version;
```

```
    DmOpenRef db;
    UInt16    categoryIndex;
  } ExgDialogInfoType;
```

-> `version` Set this field to 0 to specify version 0 of this structure.

-> `db` Pointer to an open database that defines the categories the dialog should display.

<- `categoryIndex` Index of the category in which the user wants to file the incoming data.

If `db` is valid, the function extracts the category information from the specified database and displays it in a pop-up list. Upon return, the `categoryIndex` field contains the index of the category the user selected, or `dmUnfiledCategory` if the user did not select a category.

If the call to `ExgDoDialog` is successful, your application is responsible for retaining the value returned in `categoryIndex` and using it to file the incoming data as a record in that category. One way to do this is to store the `categoryIndex` in the socket's `appData` field (see `ExgSocketType`) and then extract it from the socket in your response to the launch code `sysAppLaunchCmdExgReceiveData`. For example:

```
if (cmd == sysAppLaunchCmdExgReceiveData) {
  UInt16 categoryID =
    (ExgSocketPtr)cmdPBP->appData;
  /* other declarations */

/* Receive the data, and create a new record
  using the received data. indexNew is the
  index of this record. */

  if (categoryID){
    UInt16 attr;
    Err err;
    err = DmRecordInfo(dbP, indexNew, &attr,
      NULL, NULL);

    // Set the category to the one the user
```

```
        // specified, and mark the record dirty.
        if ((attr & dmRecAttrCategoryMask) !=
          categoryID) {
          attr &= ~dmRecAttrCategoryMask;
          attr |= categoryID | dmRecAttrDirty;
          err = DmSetRecordInfo(dbP, indexNew,
            &attr, NULL);
        }
      }
    }
```

Some of the Palm OS built-in applications (AddressBook, Memo, and ToDo) use this method of setting the category on data received through beaming. Refer to the example code for these applications provided in the SDK for a more complete example of how to use `ExgDoDialog`.

When you explicitly call `ExgDoDialog`, you must set the `result` field of the `sysAppLaunchCmdExgAskUser` launch code's parameter block to either `exgAskOk` (upon success) or `exgAskCancel` (upon failure) to prevent the system from displaying the dialog a second time.

Compatibility Implemented only if 3.5 New Feature Set is present.

ExgPut

Purpose Initiates the transfer of data to the destination device.

Prototype

```
Err ExgPut (ExgSocketPtr socketP)
```

Parameters

-> `socketP`	Pointer to the socket structure containing connection information and information identifying the object to send.

Result Returns the following result codes:

`errNone`	No error
`exgErrBadLibrary`	Couldn't find default exchange library

`exgErrStackInit`	Couldn't initialize the IR stack (not enough battery power or unsupported hardware)
`exgMemError`	Not enough memory to initialize transfer

Comments If the connection does not already exist, this function establishes one. You must create and pass a pointer to an `ExgSocketType` structure containing information about the data to send and the destination application. All unused fields in the structure **must** be zeroed.

If no error is returned, this call **must** be followed by `ExgSend`, to begin sending data, or `ExgDisconnect`, to disconnect. You may need to call `ExgSend` multiple times to send all the data.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgDisconnect`, `ExgSend`

ExgReceive

Purpose Receives data from a remote device.

Prototype

```
UInt32 ExgReceive (ExgSocketPtr socketP,
void *bufP, const UInt32 bufLen, Err * err)
```

Parameters

-> `socketP`	Pointer to the socket structure.
-> `bufP`	Pointer to the buffer to receive the data.
-> `bufLen`	Number of bytes to receive.
<- `err`	Pointer to an error code result.

Result Returns the number of bytes actually received. A zero result indicates the end of the transmission. An error code is returned in the address indicated by `err`. The error code `exgErrUserCancel` is returned if the user cancels the operation.

Comments Call this function one or more times to receive all the data, following a successful call to `ExgAccept`. After receiving the data, call `ExgDisconnect` to terminate the connection.

This function blocks the application until the end of the transmission or until the requested number of bytes has been received. However, it does provide its own user interface that will be updated as necessary and will allow the user to cancel the operation in progress.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgAccept`, `ExgDisconnect`

ExgRegisterData

Purpose Registers an application to receive a specific type of data.

Prototype

```
Err ExgRegisterData (const UInt32 creatorID,
const UInt16 id, const Char * const dataTypesP)
```

Parameters

-> `creatorID`	Creator ID of the registering application.
-> `id`	Registry ID identifying the type of the items being registered. Specify `exgRegExtensionID` or `exgRegTypeID`.
-> `dataTypesP`	Pointer to a tab-delimited, null-terminated string listing the items to register. (Use `/t` for the tab character.) These include file extensions or MIME types. To unregister, pass a `NULL` value.

Result Returns `errNone` if successful, otherwise, one of the data manager error codes (`dmErr`...).

Comments Applications that wish to receive data from anything other than another Palm OS device running the same application must use this function to register for the kinds of data they can receive. Call this function when your application is loaded on the device.

Specify `exgRegExtensionID` to register to receive data that has a filename with a particular extension. For example, if your application wants to receive files with a .TXT extension, it could register like this:

```
ExgRegisterData(myCreator, exgRegExtensionID,
"TXT");
```

If the application wants to receive files with a .TXT extension or with a .DOC extension, it could register like this:

```
ExgRegisterData(myCreator, exgRegExtensionID,
"TXT/tDOC");
```

Specify `exgRegTypeID` to register to receive data with a specific MIME type. For example, if your application wants to receive "setext" text files, it could register like this:

```
ExgRegisterData(myCreator, exgRegTypeID,
"text/x-setext");
```

Note that in the current implementation of the IR library, registering for a MIME type has no effect because the IR library does not send data type information. Therefore, the type is always received as `NULL` and will not match "setext." However, applications may choose to register for a type anyway because this limitation may be removed in the future.

Registrations are active until a hard reset or until the application is removed. The registration information is backed up and restored across a soft reset. When an application is removed, its registry information is also automatically removed from the registry, so there is not normally a need to unregister. If you want to unregister, you can register with a `NULL` value.

Compatibility Implemented only if 3.0 New Feature Set is present.

ExgSend

Purpose Sends data to the destination device.

Prototype

```
UInt32 ExgSend (ExgSocketPtr socketP,
const void * const bufP, const UInt32 bufLen,
Err * err)
```

Parameters

-> `socketP`	Pointer to the socket structure.
-> `bufP`	Pointer to the data to send.
-> `bufLen`	Number of bytes to send.
<- `err`	Pointer to an error code result.

Result Returns the number of bytes sent, normally the same number as specified in `bufLen`. An error code is returned in the address indicated by `err`. The error code `exgErrUserCancel` is returned if the user cancels the operation.

Comments Call this function one or more times to send all the data, following a successful call to `ExgPut`. After sending the data, call `ExgDisconnect` to terminate the connection.

The lower level protocol may break large amounts of data into multiple packets or assemble small send commands together into larger packets, but the application will not be aware of these transport level details.

This function blocks the application until all the data is sent. However, it does provide its own user interface that will be updated as necessary and will allow the user to cancel the operation in progress.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgDisconnect`, `ExgPut`

Application-Defined Functions

The functions in this section are supplied by you and can be named anything. You supply pointers to the functions in exchange manager functions that you call (ExgDBRead and ExgDBWrite).

DeleteProc

Purpose Handle the case where a database with an identical name already exists on the device.

Prototype

```
Boolean DeleteProc (const char* nameP,
UInt16 version, UInt16 cardNo, LocalID dbID,
void* userDataP)
```

Parameters

-> nameP	A pointer to the name of the identical database that already exists.
-> version	The version of the identical database that already exists.
-> cardNo	The card number of the identical database that already exists.
-> dbID	The database ID of the identical database that already exists.
-> userDataP	The userDataP parameter passed to ExgDBRead is simply passed on to the delete function. You can use it for application-specific data.

Result Returns a Boolean value. true means that this function handled the situation successfully; that is, it deleted, renamed, or moved the database so there would no longer be a conflict with the one that ExgDBRead is writing. false means that this function did not handle the situation successfully; in this case, ExgDBRead exits with no error (same as if the user cancelled the operation).

Comments This delete callback function gives you a chance to delete the existing database, or take some other action (such as changing the database name, if appropriate).

ReadProc

Purpose Read in the database and pass it to `ExgDBRead`.

Prototype

```
Err ReadProc (void* dataP, UInt32* sizeP,
void* userDataP)
```

Parameters

-> `dataP`	A pointer to a buffer where this function should place the database data.
<-> `sizeP`	The size of `dataP`. This value is set by `ExgDBRead` to the number of bytes it expects to receive in `dataP`. You must set this value to the number of bytes you return in `dataP` (if it's not the same).
-> `userDataP`	The `userDataP` parameter passed to `ExgDBRead` is simply passed on to the read function. You can use it for application-specific data.

Result Returns an error number, or `errNone` if there is no error. If this function returns an error, `ExgDBRead` deletes the database it was creating, cleans up any memory it allocated, then exits, returning the error passed back from this function.

WriteProc

Purpose Writes out the database.

Prototype

```
Err WriteProc (const void* dataP, UInt32* sizeP,
void* userDataP)
```

Parameters

-> `dataP`	A pointer to a buffer containing the database data, placed there by `ExgDBWrite`.

<-> `sizeP`	The number of bytes placed in `dataP` by `ExgDBWrite`. If you were unable to write out or send all of the data in this chunk, on exit, you should set `sizeP` to the number of bytes you did write.
-> `userDataP`	The `userDataP` parameter passed to `ExgDBWrite` is simply passed on to the write function. You can use it for application-specific data.

Result Returns an error number, or `errNone` if there is no error. If this function returns an error, `ExgDBWrite` closes the database it was reading, cleans up any memory it allocated, then exits, returning the error passed back from this function.

52

IR Library

The IR (InfraRed) library is a shared library that provides a direct interface to the IR communications capabilities of the Palm OS®. This chapter provides reference material for the IR library API:

- IR Library Data Structures
- IR Stack Callback Events
- IR Library Functions
- IAS Functions
- Application-Defined Functions

The header file `irlib.h` declares the IR library API. For more information on the IR library, see the chapter "Beaming (Infrared Communication)" in the *Palm OS Programmer's Companion*.

IR Library Data Structures

This section lists some of the more important data types used by IR library functions.

IrConnect

This data structure is used to manage an IrLMP or Tiny TP connection.

Listing 52.1 IrConnect Data Structure

```
/* Forward declaration of the IrConnect structure
*/
typedef struct _hconnect IrConnect;

/*-----------------------------------------------
------------- */
typedef struct _hconnect {
```

```
UInt8 lLsap; /* Local LSAP this connection will
listen on */
UInt8 rLsap; /* Remote Lsap */

/*============== For Internal Use Only
========================
 *
 * The following is used internally by the stack
and should not be
 * modified by the user.
 *

*==================================================
============*/

UInt8 flags; /* Flags containing state, type, etc.
*/
UInt8 reserved; /* Explicitly account for 16-bit
alignment padding */
IrCallBack callBack; /* Pointer to callback
function */

/* Tiny TP fields */
IrPacket packet; /* Packet for internal use */
ListEntry packets; /* List of packets to send */
UInt16 sendCredit; /* Amount of credit from peer
*/
UInt8 availCredit; /* Amount of credit to give to
peer */
UInt8 dataOff; /* Amount of data less than IrLAP
size */
} _hconnect;
```

Field Descriptions

`lLsap`	Local LSAP this connection will listen on
`rLsap`	Remote Lsap
`flags`	Flags containing state, type, etc. Do NOT modify, internal use only.

`reserved`	Reserved for future use
`callBack`	Pointer to callback function. Do NOT modify, internal use only.
`packet`	Packet for internal use
`packets`	List of packets to send
`sendCredit`	Amount of credit from peer
`availCredit`	Amount of credit to give to peer
`dataOff`	Amount of data less than IrLAP size

IrPacket

This data structure is used for sending IrDA packets.

Listing 52.2 IrPacket Data Structure

```
typedef struct _IrPacket {
/* The node field must be the first field in the
structure. It is
 * used internally by the stack. */
ListEntry node;

/* The buff field is used to point to a buffer of
data to send and
 * len field indicates the number of bytes in
buff. */
UInt8 *buff;
UInt16 len;

/*==================  For Internal Use Only
======================
 *
 * The following is used internally by the stack
and should not be
 * modified by the upper layer.
 *
```

```
*==================================================
=============*/
IrConnect* origin; /* Pointer to connection which
owns packet */
UInt8 headerLen; /* Number of bytes in the header
*/
UInt8 header[14]; /* Storage for the header */
UInt8 reserved; /*Explicitly account for 16-bit
alignment padding*/
} IrPacket;
```

Field Descriptions

`node`	Reserved for internal use
`buff`	Points to a buffer of data to send
`len`	Number of bytes in `buff`
`origin`	Pointer to connection which owns packet. Do NOT modify, internal use only.
`headerLen`	Number of bytes in the header. Do NOT modify, internal use only.
`header`	Storage for the header. Do NOT modify, internal use only.
`reserved`	Reserved for future use

IrIASObject

This data structure is used as storage for an IAS object managed by the local IAS server. An object of this type is passed as the `obj` parameter to the `IrIAS_Add` function.

Listing 52.3 IrIASObject Data Structure

```
typedef struct _IrIasObject {
UInt8 *name; /* Pointer to name of object */
UInt8 len; /* Length of object name */
```

```
UInt8 nAttribs; /* Number of attributes */
IrIasAttribute* attribs; /* A pointer to an array
of attributes */
} IrIasObject;
```

Field Descriptions

`name`	Pointer to name of object
`len`	Length of object name
`nAttribs`	Number of attributes
`attribs`	Pointer to an array of attributes

IrIasQuery

This data structure is used for performing IAS queries. An object of this type is passed as the `token` parameter to the `IrIAS_Query` function (and several other functions as well).

Listing 52.4 IrIasQuery Data Structure

```
* Forward declaration of a structure used for
performing IAS
* Queries so that a callback type can be defined
for use in
* the structure. */
typedef struct _IrIasQuery IrIasQuery;
typedef void (*IrIasQueryCallBack)(IrStatus);

* Actual definition of the IrIasQuery structure.
*/
typedef struct _IrIasQuery
{
/* Query fields. The query buffer contains the
class name and
 * class attribute whose value is being queried--
it is as follows:
 *
 * 1 byte - Length of class name
```

```
 * "Length" bytes - class name
 * 1 byte - length of attribute name
 * "Length" bytes - attribute name
 *
 * queryLen - contains the total number of byte in
the query */
UInt8 queryLen; /* Total length of the query */
UInt8 reserved; /* Explicitly account for 16-bit
alignment padding */
UInt8 *queryBuf; /* Points to buffer containing
the query */

/* Fields for the query result */
UInt16 resultBufSize; /* Size of the result buffer
*/
UInt16 resultLen; /* Actual number of bytes in the
result buffer */
UInt16 listLen; /* Number of items in the result
list. */
UInt16 offset; /* Offset into results buffer */
UInt8 retCode; /* Return code of operation */
UInt8 overFlow; /* Set TRUE if result exceeded
result buffer size*/
UInt8 *result; /* Pointer to buffer containing
result; */

/* Pointer to callback function */
IrIasQueryCallBack callBack;
} _IrIasQuery;
```

Field Descriptions

`queryLen`	Total length of the query
`reserved`	Reserved for future use
`queryBuf`	Pointer to buffer containing the query
`resultBufSize`	Size of the result buffer
`resultLen`	Actual number of bytes in the result buffer

`listLen`	Number of items in the result list
`offset`	Offset into results buffer
`retCode`	Return code of operation
`overFlow`	Set TRUE if result exceeded result buffer size
`result`	Pointer to buffer containing result
`callBack`	Pointer to query callback function

IrCallbackParms

This data structure is used to pass information from the stack to the upper layer of the stack (application). Not all fields are valid at any given time. The type of event determines which fields are valid. An object of this type is passed as the second parameter to the `IrCallback` function.

Listing 52.5 IrCallbackParms Data Structure

```
typedef struct {
IrEvent event; /* Event causing callback */
UInt8 reserved1; /* Explicitly account for 16-bit
alignment padding */
UInt8 *rxBuff; /* Receive buffer already advanced
to app data */
UInt16 rxLen; /* Length of data in receive buffer
*/
IrPacket* packet; /* Pointer to packet being
returned */
IrDeviceList* deviceList; /* Pointer to discovery
device list */
IrStatus status; /* Status of stack */
UInt8 reserved2; /* Explicitly account for 16-bit
alignment padding */
} IrCallBackParms;
```

Field Descriptions

`event`	Event causing callback
`reserved1`	Reserved for future use
`rxBuff`	Receive buffer already advanced to app data
`rxLen`	Length of data in receive buffer
`packet`	Pointer to packet being returned
`deviceList`	Pointer to discovery device list
`status`	Status of stack
`reserved2`	Reserved for future use

IR Stack Callback Events

The IR stack calls the application via a callback function stored in each `IrConnect` structure. The callback function is called with a pointer to the `IrConnect` structure and a pointer to a parameter structure. The parameter structure contains an `event` field, which indicates the reason the callback is called, and other parameters, which have meaning based on the event.

The meaning of the events is described in the following sections.

LEVENT_DATA_IND

Data has been received. The received data is accessed using fields `rxBuff` and `rxLen`.

LEVENT_DISCOVERY_CNF

Indicates the completion of a discovery operation. The field `deviceList` points to the discovery list.

LEVENT_LAP_CON_CNF

The requested IrLAP connection has been made successfully. The callback function of all bound `IrConnect` structures is called.

LEVENT_LAP_CON_IND

Indicates that the IrLAP connection has come up. The callback of all bound `IrConnect` structures is called.

LEVENT_LAP_DISCON_IND

Indicates that the IrLAP connection has gone down. This means that all IrLMP connections are also down. A callback with event `LEVENT_LM_CON_IND` will not be given. The callback function of all bound `IrConnect` structures is called.

LEVENT_LM_CON_CNF

The requested IrLMP/Tiny TP connection has been made successfully. Connection data from the other side is found using fields `rxBuff` and `rxLen`.

LEVENT_LM_CON_IND

Other device has initiated a connection. `IrConnectRsp` should be called to accept the connection. Any data associated with the connection request can be found using fields `rxBuff` and `rxLen`, for the data pointer and length, respectively.

LEVENT_LM_DISCON_IND

The IrLMP/Tiny TP connection has been disconnected. Any data associated with the disconnect indication can be found using fields `rxBuff` and `rxLen`, for the data pointer and length, respectively.

LEVENT_PACKET_HANDLED

A packet is being returned. A pointer to the packet exists in field `packet`.

LEVENT_STATUS_IND

Indicates that a status event from the stack has occurred. The `status` field indicates the status generating the event. Possible statuses are as follows.

- `IR_STATUS_NO_PROGRESS` means that IrLAP has no progress for 3 seconds threshold time (e.g. the beam is blocked).
- `IR_STATUS_LINK_OK` indicates that the no progress condition has cleared.
- `IR_STATUS_MEDIA_NOT_BUSY` indicates that the IR media has transitioned from busy to not busy.

LEVENT_TEST_CNF

Indicates that a TEST command has completed. The `status` field indicates if the test was successful. `IR_STATUS_SUCCESS` indicates that operation was successful and the data in the test response can be found by using the `rxBuff` and `rxLen` fields. `IR_STATUS_FAILED` indicates that no TEST response was received. The packet passed to perform the test command is passed back in the `packet` field and is now available (no separate packet handled event will occur).

LEVENT_TEST_IND

Indicates that a TEST command frame has been received. A pointer to the received data is in `rxBuff` and `rxLen`. A pointer to the packet that will be sent in response to the test command is in the `packet` field. The packet is currently set up to respond with the same data sent in the command TEST frame. If different data is desired as a response, then modify the packet structure. This event is sent to the callback function in all bound `IrConnect` structures. The IAS connections ignore this event.

IR Library Functions

IrAdvanceCredit

Purpose Advances credit to the other side of the connection.

Prototype

```
void IrAdvanceCredit (IrConnect* con,
UInt8 credit)
```

Parameters

--> `con`	Pointer to `IrConnect` structure representing connection to which credit is advanced.
--> `credit`	Amount of credit to advance.

Result Returns nothing.

Comments The total amount of credit should not exceed 127. The credit passed by this function is added to the existing available credit, which is must not exceed 127. This function only makes sense for a Tiny TP connection.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrBind

Purpose Obtains a local LSAP selector and registers the connection with the protocol stack.

Prototype

```
IrStatus IrBind (UInt16 refNum, IrConnect* con,
IrCallBack callBack)
```

Parameters

--> `refnum`	IR library refNum.
<--> `con`	Pointer to `IrConnect` structure.

--> `callBack` Pointer to a callBack function that handles the indications and confirmation from the protocol stack.

Result `IR_STATUS_SUCCESS` means the operation completed successfully. The assigned LSAP can be found in `con->lLsap`.

`IR_STATUS_FAILED` means the operation failed for one of the following reasons:

- `con` is already bound to the stack
- no room in the connection table

Comments This `IrConnect` structure will be initialized. Any values stored in the structure will be lost. The assigned LSAP will be in the `lLsap` field of `con`. The type of the connection will be set to IrLMP. The `IrConnect` must be bound to the stack before it can be used.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrClose

Purpose Closes the IR library. This releases the global memory for the IR stack and any system resources it uses. This must be called when an application is done with the IR library.

Prototype

```
Err IrClose (UInt16 refnum)
```

Parameters --> `refnum` IR library refNum.

Result Returns 0 if successful.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrConnectIrLap

Purpose Starts an IrLAP connection.

Prototype

```
IrStatus IrConnectIrLap (UInt16 refNum,
IrDeviceAddr deviceAddr)
```

Parameters

--> `refnum`	IR library refNum.
--> `deviceAddr`	32-bit address of device to which connection should be made.

Result `IR_STATUS_PENDING` means the operation is started successfully; the result is returned via callback.

`IR_STATUS_MEDIA_BUSY` means the operation failed because the media is busy. Media busy is caused by one of the following reasons:

- Other devices are using the IR medium.
- An IrLAP connection already exists.
- A discovery process is in progress.

Comments The result is signaled to all bound `IrConnect` structures via the callback function. The callback event is `LEVENT_LAP_CON_CNF` if successful or `LEVENT_LAP_DISCON_IND` if unsuccessful.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrConnectReq

Purpose Requests an IrLMP or Tiny TP connection.

Prototype

```
IrStatus IrConnectReq (UInt16 refNum,
IrConnect* con, IrPacket* packet, UInt8 credit)
```

Parameters

--> `refnum`	IR library refNum.

--> `con`	Pointer to `IrConnect` structure for handling the connection. The `rLsap` field must contain the LSAP selector for the peer on the other device. Also the type of the connection must be set. Use `IR_SetConTypeLMP` to set the type to an IrLMP connection or `IR_SetConTypeTTP` to set the type to a Tiny TP connection.
--> `packet`	Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid `IrPacket` structure. The packet will be returned via the callback with the `LEVENT_PACKET_HANDLED` event if no errors occur. The maximum size of the packet is `IR_MAX_CON_PACKET` for an IrLMP connection or `IR_MAX_TTP_CON_PACKET` for a Tiny TP connection.
--> `credit`	Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127 unexpected results will occur. This parameter is ignored if the connection is an IrLMP connection.

Result

`IR_STATUS_PENDING` means the operation has been started successfully and the result will be returned via the callback function with the event `LEVENT_LM_CON_CNF` if the connection is made or `LEVENT_LM_DISCON_IND` if connection fails. The packet is returned via the callback with the event `LEVENT_PACKET_HANDLED`.

`IR_STATUS_FAILED` means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- Connection is busy (already involved in a connection)
- `IrConnect` structure is not bound to the stack
- Packet size exceeds maximum allowed

`IR_STATUS_NO_IRLAP` means the operation failed because there is no IrLAP connection (the packet is available immediately).

Comments The result is signaled via the callback specified in the `IrConnect` structure. The callback event is `LEVENT_LM_CON_CNF` indicates that the connection is up and `LEVENT_LM_DISCON_IND` indicates that the connection failed. Before calling this function the fields in the con structure must be properly set.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrConnectRsp

Purpose Accepts an incoming connection that has been signaled via the callback with the event `LEVENT_LM_CON_IND`.

Prototype

```
IrStatus IrConnectRsp (UInt16 refNum,
IrConnect* con, IrPacket* packet, UInt8 credit)
```

Parameters

--> `refnum`	IR library refNum.
--> `con`	Pointer to `IrConnect` structure to managed connection.
--> `packet`	Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid `IrPacket` structure. The packet will be returned via the callback with the `LEVENT_PACKET_HANDLED` event if no errors occur. The maximum size of the packet is `IR_MAX_CON_PACKET` for an IrLMP connection or `IR_MAX_TTP_CON_PACKET` for a Tiny TP connection.

--> `credit`	Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127 unexpected results will occur. This parameter is ignored if the connection is an IrLMP connection.

Result

`IR_STATUS_PENDING` means the operation has been started successfully and the packet will be returned via the callback function with the event `LEVENT_PACKET_HANDLED`.

`IR_STATUS_FAILED` means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- Connection is not in the proper state to require a response
- `IrConnect` structure is not bound to the stack
- Packet size exceeds maximum allowed

`IR_STATUS_NO_IRLAP` means the operation failed because there is no IrLAP connection (the packet is available immediately).

Comments

`IrConnectRsp` can be called during the callback or later to accept the connection. The type of the connection must already have been set to IrLMP or Tiny TP before the `LEVENT_LM_CON_IND` event.

Compatibility

Implemented only if 3.0 New Feature Set is present.

IrDataReq

Purpose

Sends a data packet.

Prototype

```
IrStatus IrDataReq (UInt16 refNum, IrConnect* con,
IrPacket* packet)
```

Parameters

--> `refnum`	IR library refNum.
--> `con`	Pointer to `IrConnect` structure that specifies the connection over which the packet should be sent.

--> `packet` Pointer to a valid `IrPacket` structure that contains data to send. The packet should not exceed the max size found with `IrMaxTxSize`.

Result `IR_STATUS_PENDING` means the packet has been queued by the stack. The packet will be returned via the callback with event `LEVENT_PACKET_HANDLED`.

`IR_STATUS_FAILED` means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- `IrConnect` structure is not bound to the stack
- Packet size exceeds maximum allowed
- `IrConnect` structure does not represent an active connection

Comments The packet is owned by the stack until it is returned via the callback with event `LEVENT_PACKET_HANDLED`. The largest packet that can be sent is found by calling `IrMaxTxSize`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrDisconnectIrLap

Purpose Disconnects an IrLAP connection.

Prototype

```
IrStatus IrDisconnectIrLap (UInt16 refNum)
```

Parameters --> `refnum` IR library refNum.

Result `IR_STATUS_PENDING` means the operation started successfully and all bound `IrConnect` structures will be called back when complete.

`IR_STATUS_NO_IRLAP` means the operation failed because no IrLAP connection exists.

Comments When the IrLAP connection goes down, the callback of all bound `IrConnect` structures is called with event `LEVENT_LAP_DISCON_IND`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrDiscoverReq

Purpose Starts an IrLMP discovery process.

Prototype

```
IrStatus IrDiscoverReq (UInt16 refNum,
IrConnect* con)
```

Parameters

--> `refnum`	IR library refNum.
--> `con`	Pointer to a bound `IrConnect` structure.

Result `IR_STATUS_PENDING` means the operation is started successfully; the result is returned via callback.

`IR_STATUS_MEDIA_BUSY` means the operation failed because the media is busy. Media busy is caused by one of the following reasons:

- Other devices are using the IR medium.
- A discovery process is already in progress.
- An IrLAP connection exists.

`IR_STATUS_FAILED` means the operation failed because the `IrConnect` structure is not bound to the stack.

Comments The result will be signaled via the callback function specified in the `IrConnect` structure with the event `LEVENT_DISCOVERY_CNF`. Only one discovery can be invoked at a time.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIsIrLapConnected

Purpose Determines if an IrLAP connection exists.

Prototype `BOOL IrIsIrLapConnected (UInt16 refNum)`

Parameters --> `refnum` IR library refNum.

Result True if IrLAP is connected, false otherwise.

Comments Only available if `IR_IS_LAP_FUNCS` is defined.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIsMediaBusy

Purpose Determines if the IR media is busy.

Prototype `BOOL IrIsMediaBusy (UInt16 refNum)`

Parameters --> `refnum` IR library refNum.

Result True if IR media is busy, false otherwise.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIsNoProgress

Purpose Determines if IrLAP is not making progress.

Prototype `BOOL IrIsNoProgress (UInt16 refNum)`

Parameters --> `refnum` IR library refNum.

Result True if IrLAP is not making progress, false otherwise.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIsRemoteBusy

Purpose Determines if the other device's IrLAP is busy.

Prototype

```
BOOL IrIsRemoteBusy (UInt16 refNum)
```

Parameters

--> `refnum` IR library refNum.

Result True if the other device's IrLAP is busy, false otherwise.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrLocalBusy

Purpose Sets the IrLAP local busy flag.

Prototype

```
void IrLocalBusy (UInt16 refNum, BOOL flag)
```

Parameters

--> `refnum` IR library refNum.

--> `flag` Value (true or false) to set for IrLAP's local busy flag.

Result Returns nothing.

Comments If local busy is set to true, then the local IrLAP layer will send RNR (Receive Not Ready) frames to the other side indicating it cannot receive any more data. If the local busy is set to false, IrLAP is ready to receive frames.

The setting takes effect the next time IrLAP sends an RR (Receive Ready) frame. If IrLAP has data to send, the data will be sent first, so it should be used carefully.

This function should not be used when using Tiny TP or when multiple connections exist.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrMaxRxSize

Purpose Returns the maximum size buffer that can be sent by the other device.

Prototype

```
UInt16 IrMaxRxSize (UInt16 refNum, IrConnect* con)
```

Parameters

--> `refnum`	IR library refNum.
--> `con`	Pointer to `IrConnect` structure that represents an active connection.

Result Returns the maximum size buffer that can be sent by the other device (maximum bytes that can be received). The value returned is only valid for active connections. The maximum size will vary for each connection and is based on the negotiated IrLAP parameters and the type of the connection.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrMaxTxSize

Purpose Returns the maximum size allowed for a transmit packet.

Prototype

```
UInt16 IrMaxTxSize (UInt16 refNum, IrConnect* con)
```

Parameters

--> `refnum`	IR library refNum.
--> `con`	Pointer to `IrConnect` structure that represents an active connection.

Result Returns the maximum size allowed for a transmit packet. The value returned is only valid for active connections. The maximum size will vary for each connection and is based on the negotiated IrLAP parameters and the type of the connection.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrOpen

Purpose Opens the IR library. This allocates the global memory for the IR stack and reserves the system resources it requires. This must be done before any other IR library calls are made.

Prototype

```
Err IrOpen (UInt16 refnum, UInt32 options)
```

Parameters

--> `refnum`	IR library refNum. This value is returned from the function `SysLibFind`, which you must call first to load the IR library.
--> `options`	Open options flags. See the Comments section for details.

Result Returns 0 if successful.

Comments The following flags can be specified for the `options` parameter to set the speed of the connection:

`irOpenOptSpeed115200`	Set maximum negotiated baud rate
`irOpenOptSpeed57600`	Set 57600 bps (default if no flags given)
`irOpenOptSpeed9600`	Set 9600 bps

Compatibility Implemented only if 3.0 New Feature Set is present.

IrSetConTypeLMP

Purpose Sets the type of the connection to IrLMP. This function must be called after the `IrConnect` structure is bound to the stack.

Prototype

```
void IrSetConTypeLMP (IrConnect* con)
```

Parameters

--> `con`	Pointer to `IrConnect` structure.

Result Returns nothing.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrSetConTypeTTP

Purpose Sets the type of the connection to Tiny TP. This function must be called after the `IrConnect` structure is bound to the stack.

Prototype

```
void IrSetConTypeTTP (IrConnect* con)
```

Parameters

--> `con`	Pointer to `IrConnect` structure.

Result Returns nothing.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrSetDeviceInfo

Purpose Sets the XID info string used during discovery to the given string and length.

Prototype

```
IrStatus IrSetDeviceInfo (UInt16 refNum,
UInt8 *info, UInt8 len)
```

Parameters

--> `refnum`	IR library refNum.
--> `info`	Pointer to array of bytes.
--> `len`	Number of bytes pointed to by `info`.

Result `IR_STATUS_SUCCESS` means the operation is successful.

`IR_STATUS_FAILED` means the operation failed because `info` is too big.

Comments The XID info string contains hints and the nickname of the device. The size cannot exceed `IR_MAX_DEVICE_INFO` bytes.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrTestReq

Purpose Requests a TEST command frame be sent in the NDM (Normal disconnect Mode) state.

Prototype

```
IrStatus IrTestReq (UInt16 refNum,
IrDeviceAddr devAddr, IrConnect* con,
IrPacket* packet)
```

Parameters

--> `refnum`	IR library refNum.
--> `devAddr`	Device address of device where TEST will be sent. This address is not checked so it can be the broadcast address or 0.
--> `con`	Pointer to `IrConnect` structure specifying the callback function to call to report the result.
--> `packet`	Pointer to an `IrPacket` structure that contains the data to send in the TEST command packet. The maximum size data that can be sent is `IR_MAX_TEST_PACKET`. Even if no data is to be sent, a valid packet must be passed.

Result `IR_STATUS_PENDING` means the operation has been started successfully and the result will be returned via the callback function with the event `LEVENT_TEST_CNF`. This is also the indication returning the packet.

`IR_STATUS_FAILED` means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- `IrConnect` structure is not bound to the stack
- Packet size exceeds maximum allowed

`IR_STATUS_MEDIA_BUSY` means the operation failed because the media is busy or the stack is not in the NDM state (the packet is available immediately).

Comments The result is signaled via the callback specified in the `IrConnect` structure. The callback event is `LEVENT_TEST_CNF` and the status field indicates the result of the operation. `IR_STATUS_SUCCESS`

indicates success and `IR_STATUS_FAILED` indicates no response was received. A packet must be passed containing the data to send in the TEST frame. The packet is returned when the `LEVENT_TEST_CNF` event is given.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrUnbind

Purpose Unbinds the `IrConnect` structure from the protocol stack, freeing it's LSAP selector.

Prototype

```
IrStatus IrUnbind (UInt16 refNum, IrConnect* con)
```

Parameters

--> `refnum`	IR library refNum.
--> `con`	Pointer to `IrConnect` structure to unbind.

Result `IR_STATUS_SUCCESS` means the operation completed successfully.

`IR_STATUS_FAILED` means the operation failed for one of the following reasons:

- the `IrConnect` structure was not bound
- the `lLsap` field contained an invalid number

Compatibility Implemented only if 3.0 New Feature Set is present.

IAS Functions

This section describes functions and macros related to IAS databases:

- `IrIAS_Add`
- `IrIAS_GetInteger`
- `IrIAS_GetIntLsap`
- `IrIAS_GetObjectID`
- `IrIAS_GetOctetString`
- `IrIAS_GetOctetStringLen`

- `IrIAS_GetType`
- `IrIAS_GetUserString`
- `IrIAS_GetUserStringCharSet`
- `IrIAS_GetUserStringLen`
- `IrIAS_Next`
- `IrIAS_Query`
- `IrIAS_SetDeviceName`
- `IrIAS_StartResult`

IrIAS_Add

Purpose Adds an IAS object to the IAS Database.

Prototype

```
IrStatus IrIAS_Add (UInt16 refNum,
IrIasObject* obj)
```

Parameters

--> `refnum`	IR library refNum.
--> `obj`	Pointer to an `IrIASObject` structure.

Result `IR_STATUS_SUCCESS` means the operation is successful.

IR_STATUS_FAILED means the operation failed for one of the following reasons:

- No space in the database.
- An entry with the same class name already exists.
- The attributes of the object violate the IrDA Lite rules (attribute name exceeds `IR_MAX_IAS_NAME`, or attribute value exceeds `IR_MAX_IAS_ATTR_SIZE`).
- The class name exceeds `IR_MAX_IAS_NAME`.

Comments The object is not copied, so the memory for the object must exist for as long as the object is in the database. The IAS database is designed to allow only objects with unique class names, and it checks for this. Class names and attributes names must not exceed

`IR_MAX_IAS_NAME`. Also, attribute values must not exceed `IR_MAX_IAS_ATTR_SIZE`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetInteger

Purpose Macro that returns an integer value, assuming that the current result item is of type `IAS_ATTRIB_INTEGER`. (Call `IrIAS_GetType` to determine the type of the current result item.)

Prototype

```
IrIAS_GetInteger (t)
```

Parameters

--> `t`	Pointer to an `IrIasQuery` structure.

Result Integer value returned as a `UInt32`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetIntLsap

Purpose Macro that returns an integer value that represents an LSAP, assuming that the current result item is of type `IAS_ATTRIB_INTEGER`. (Call `IrIAS_GetType` to determine the type of the current result item.) Usually integer values returned in a query are LSAP selectors.

Prototype

```
IrIAS_GetIntLsap (t)
```

Parameters

--> `t`	Pointer to an `IrIasQuery` structure.

Result Integer value returned as a `UInt8`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetObjectID

Purpose Macro that returns the unique object ID of the current result item.

Prototype `IrIAS_GetObjectID (t)`

Parameters --> `t` Pointer to an `IrIasQuery` structure.

Result Returns the object ID as a `UInt16` type.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetOctetString

Purpose Macro that returns a pointer to an octet string, assuming that the current result item is of type IAS_ATTRIB_OCTET_STRING. (Call `IrIAS_GetType` to determine the type of the current result item.)

Prototype `IrIAS_GetOctetString (t)`

Parameters --> `t` Pointer to an `IrIasQuery` structure.

Result Pointer to octet string of type `UInt8`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetOctetStringLen

Purpose Macro that returns the length of an octet string, assuming that the current result item is of type `IAS_ATTRIB_OCTET_STRING`. (Call `IrIAS_GetType` to determine the type of the current result item.)

Prototype `IrIAS_GetOctetStringLen (t)`

Parameters --> `t` Pointer to an `IrIasQuery` structure.

Result Length of octet string returned as a `UInt16`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetType

Purpose Macro that returns the type of the current result item.

Prototype

```
IrIAS_GetType (t)
```

Parameters

--> `t` Pointer to an `IrIasQuery` structure.

Result Type of result item, such as `IAS_ATTRIB_INTEGER`, `IAS_ATTRIB_OCTET_STRING` or `IAS_ATTRIB_USER_STRING`. The return value is of type `UInt8`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetUserString

Purpose Macro that returns a pointer to a user string, assuming that the current result item is of type `IAS_ATTRIB_USER_STRING`. (Call `IrIAS_GetType` to determine the type of the current result item.)

Prototype

```
IrIAS_GetUserString(t)
```

Parameters

--> `t` Pointer to an `IrIasQuery` structure.

Result Pointer to result string of type `UInt8`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetUserStringCharSet

Purpose Macro that returns the character set of the user string, assuming that the current result item is of type `IAS_ATTRIB_USER_STRING`. (Call `IrIAS_GetType` to determine the type of the current result item.)

Prototype

```
IrIAS_GetUserStringCharSet(t)
```

Parameters --> `t` Pointer to an `IrIasQuery` structure.

Result Character set returned as an `IrCharSet` value.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetUserStringLen

Purpose Macro that returns the length of a user string, assuming that the current result item is of type `IAS_ATTRIB_USER_STRING`. (Call `IrIAS_GetType` to determine the type of the current result item.)

Prototype

```
IrIAS_GetUserStringLen (t)
```

Parameters --> `t` Pointer to an `IrIasQuery` structure.

Result Length of user string returned as a `UInt8` value.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_Next

Purpose Moves the internal pointer to the next result item.

Prototype

```
UInt8* IrIAS_Next (UInt16 refNum,
IrIasQuery* token)
```

Parameters --> `refnum` IR library refNum.

--> `token` Pointer to an `IrIasQuery` structure.

Result Pointer to the next result item, or 0 if there are no more items.

Comments This function returns a pointer to the start of the next result item. If the pointer is 0, then there are no more result items.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_Query

Purpose Makes an IAS query of another device's IAS database.

Prototype

```
IrStatus IrIAS_Query (UInt16 refNum,
IrIasQuery* token)
```

Parameters

--> `refnum` IR library refNum.

--> `token` Pointer to an `IrIasQuery` structure initialized as described in the Comments section.

Result `IR_STATUS_SUCCESS` means the operation is started successfully and the result will be signaled via the callback function.

IR_STATUS_FAILED means the operation failed for one of the following reasons:

- The query exceeds `IR_MAX_QUERY_LEN`.
- The `result` field of `token` is 0.
- The `resultBufSize` field of `token` is 0.
- The `callback` field of `token` is 0.
- A query is already in progress.

`IR_STATUS_NO_IRLAP` means the operation failed because there is no IrLAP connection.

Comments An IrLAP connection must exist to the other device. The IAS query token must be initialized as described below. The result is signaled by calling the callback function whose pointer exists in the `IrIasQuery` structure. Only one query can be made at a time.

The `IrIasQuery` structure passed in the `token` parameter must be initialized as follows:

- pointer to a callback function in which the result will signaled.
- `result` points to a buffer large enough to hold the result of the query.
- `resultBufSize` is set to the size of the result buffer.
- `queryBuf` must point to a valid query.
- `queryLen` is set to the number of bytes in `queryBuf`. The length must not exceed `IR_MAX_QUERY_LEN`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_SetDeviceName

Purpose Sets the `value` field of the device name attribute of the "Device" object in the IAS database.

Prototype

```
IrStatus IrIAS_SetDeviceName (UInt16 refNum,
UInt8 *name, UInt8 len)
```

Parameters

--> `refnum`	IR library refNum.
--> `name`	Pointer to an IAS `value` field for the device name attribute of the device object. It includes the attribute type, character set and device name. This `value` field should be a constant and the pointer must remain valid until `IrIAS_SetDeviceName` is called with another pointer.
--> `len`	Total length of the `value` field. Maximum size allowed is `IR_MAX_IAS_ATTR_SIZE`.

Result `IR_STATUS_SUCCESS` means the operation is successful.

`IR_STATUS_FAILED` means `len` is too big, or the `value` field is not a valid user string.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_StartResult

Purpose Macro that puts the internal pointer to the start of the result buffer.

Prototype `IrIAS_StartResult(t)`

Parameters --> `t` Pointer to an `IrIasQuery` structure.

Result Returns nothing.

Compatibility Implemented only if 3.0 New Feature Set is present.

Application-Defined Functions

The functions in this section are supplied by you and can be named anything.

IrIasQueryCallBack

Purpose The result of IAS queries is signaled by calling this callback function that is pointed to by the `callBack` field of the `IrIasQuery` structure.

Prototype

```
void IrIasQueryCallBack (IrStatus status)
```

Parameters `--> status` The status of the query operation. The following values can be passed:

`IR_STATUS_SUCCESS` means the query operation finished successfully and the results can be parsed.

`IR_STATUS_DISCONNECT` means the link or IrLMP connection was disconnected during the query, so the results are not valid.

Result Returns nothing

53

Modem Manager

This chapter provides reference material for the modem manager API. The header file `ModemMgr.h` declares the modem manager API.

Modem Manager Functions

MdmDial

Purpose Initialize the modem, dial the phone number and wait for result.

Prototype

```
Err MdmDial (MdmInfoPtr modemP, Char *okDialP,
Char *userInitP, Char *phoneNumP)
```

Parameters

`modemP`	Pointer to modem info structure (filled in by caller)
`okDialP`	(NOT IMPLEMENTED) Pointer to string of chars allowed in dial string
`userInitP`	Pointer to modem setup string without the AT prefix.
`phoneNumP`	Pointer to phone number string

Result 0 if successful; otherwise `mdmErrNoTone`, `mdmErrNoDCD`, `mdmErrBusy`, `mdmErrUserCan`, `mdmErrCmdError`

Comments When executing this function, the system performs these steps:

- Switch to the requested initial baud rate.
- If HW hand-shake is requested, enable CTS/RTS hand-shaking; otherwise, disable it.
- Reset the modem.

- Execute the setup string (if any).
- Configure the modem with required settings.
- Dial the phone number.
- Wait for CONNECT XXXXX or other response.
- If auto-baud is requested, switch to the connected baud rate.

MdmHangUp

Purpose Hang up the modem.

Prototype

```
Err MdmHangUp (MdmInfoPtr modemP)
```

Parameters

`modemP` Pointer to modem info structure (filled in by caller)

Result 0 if successful.

WARNING! This function alters configuration of the serial port (without restoring it).

54

Net Library

This chapter describes the API available in the net library and its Berkeley sockets equivalents. The header file `NetMgr.h` declares the net library API. The chapter covers:

- Net Library Data Structures
- Net Library Constants
- Net Library Functions

For more information on the net library, see the chapter "Network Communication" in the *Palm OS Programmer's Companion*.

IMPORTANT: Applications cannot directly use the net library to make wireless connections. Use the INetLib for wireless connections.

Net Library Data Structures

NetHostInfoBufType

The `NetHostInfoBufType` struct contains information about a host. The `NetHostInfoType` struct, which maps to the `hostent` struct, points to fields in this struct for its information.

```
typedef struct {
  NetHostInfoType hostInfo;
  Char        name[netDNSMaxDomainName+1];
  Char *      aliasList[netDNSMaxAliases+1];
  Char        aliases[netDNSMaxAliases]
                [netDNSMaxAliases+1];
  NetIPAddr*  addressList[netDNSMaxAddresses];
  NetIPAddr   address[netDNSMaxAddresses];
} NetHostInfoBufType, *NetHostInfoBufPtr;
```

Field Descriptions

`hostInfo`	A `NetHostInfoType` struct, which maps to the Berkeley UNIX sockets `hostent` structure.
`name`	Official host name.
`aliasList` `aliases`	An array of aliases for the host name.
`addressList` `address`	An array of pointers to 32-bit IP addresses in host byte order.

NetHostInfoType

The `NetHostInfoType` structure maps to the Berkeley UNIX sockets `hostent` structure. It is defined as follows:

```
typedef struct {
  Char *    nameP;
  Char **   nameAliasesP;
  UInt16    addrType;
  UInt16    addrLen;
  UInt8 **  addrListP;
} NetHostInfoType, *NetHostInfoPtr;
```

Field Descriptions

`nameP`	Official host name.
`nameAliasesP`	An array of aliases for the host name.
`addrType`	The type of the return addresses. See `NetSocketAddrEnum`.
`addrLen`	The length in bytes of the return addresses.
`addrListP`	An array of pointers to addresses in host byte order.

NetServInfoBufType

The `NetServInfoBufType` struct contains information about a service. The `NetServInfoType` struct, which maps to the `servent` struct, points to fields in this struct for much of its information.

```
struct {
  NetServInfoType servInfo;
  Char            name[netServMaxName+1];
  Char *          aliasList[netServMaxAliases+1];
  Char            aliases[netServMaxAliases]
[netServMaxName];
  Char            protoName[netProtoMaxName+1];
  UInt8           reserved;
} NetServInfoBufType, *NetServInfoBufPtr;
```

Field Descriptions

`servInfo`	A `NetServInfoType` struct, which maps to the Berkeley UNIX sockets `servent` structure.
`name`	Official name of the service
`aliasList` `aliases`	Array of aliases for the service name.
`protoName`	Name of the protocol to use.
`reserved`	Reserved for future use.

NetServInfoType

The `NetServInfoType` structure maps to the `servent` structure in Berkeley UNIX sockets API. It contains information about a service.

```
struct {
  Char *    nameP;
  Char **   nameAliasesP;
  UInt16    port;
  Char *    protoP;
} NetServInfoType, *NetServInfoPtr;
```

Field Descriptions

`nameP`	Official name of the service
`nameAliasesP`	Array of aliases for the service name.
`port`	Port number for the service.
`protoP`	Name of the protocol to use.

NetSocketAddrEnum

The `NetSocketAddrEnum` enum specifies the address types supported by the net library.

```
typedef enum {
  netSocketAddrRaw = 0,
  netSocketAddrINET = 2
} NetSocketAddrEnum
```

Value Descriptions

`netSocketAddrRaw`	Raw address. Supported in Palm OS® version 3.0 and higher.
`netSocketAddrINET`	IP address.

NetSocketAddrINType

The `NetSocketAddrINType` struct holds an internet socket address, that is, a socket that uses one of the internet protocols. This structure directly maps to the BSD UNIX `sockaddr_in` structure.

```
typedef struct NetSocketAddrINType {
  Int16     family;
  UInt16    port;
  NetIPAddr addr;
} NetSocketAddrINType;
```

Field Descriptions

`family` Address family in host byte order. This is either `netSocketAddrINET` or `netSocketAddrRaw`.

`port` The port in network byte order.

`addr` The IP address in network byte order.

NetSocketAddrRawType

The `NetSocketAddrRawType` structure holds a raw socket address.

```
typedef struct NetSocketAddrRawType {
  Int16 family;
  UInt16 ifInstance;
  UInt32 ifCreator;
} NetSocketAddrRawType;
```

Field Descriptions

`family`	Address family in host byte order. This is either `netSocketAddrINET` or `netSocketAddrRaw`.
`ifInstance`	The instance number of the interface that the socket uses to send and receive data.
`ifCreator`	The creator of the interface that the socket uses.

Compatibility

Raw sockets are supported in Palm OS® version 3.0 and higher.

NetSocketAddrType

The `NetSocketAddrType` structure holds a generic socket address. This struct can hold any type of address including Internet addresses. It directly maps to the BSD UNIX `sockaddr` structure.

Note that this structure is the same size as `NetSocketAddrINType` and `NetSocketAddrRawType`. This means that one of those two structures can be used for parameters declared to be `NetSocketAddrType`.

```
typedef struct NetSocketAddrType {
  Int16 family;
  UInt8 data[14];
} NetSocketAddrType;
```

NetSocketRef

The `NetSocketRef` defines a socket descriptor. The socket descriptor is created and returned by `NetLibSocketOpen`. It is used in any function that requires access to a socket.

```
typedef Int16 NetSocketRef
```

NetSocketTypeEnum

The `NetSocketTypeEnum` enum specifies the available socket types.

```
typedef enum {
  netSocketTypeStream=1,
  netSocketTypeDatagram=2,
  netSocketTypeRaw=3,
  netSocketTypeReliableMsg=4
} NetSocketTypeEnum
```

Value Descriptions

`netSocketTypeStream`	Streams protocol over wireline.
`netSocketTypeDatagram`	UDP protocol.
`netSocketTypeRaw`	Raw mode.

Net Library Constants

I/O Flags

The I/O flags specify special handling instructions to functions that send and receive data. You can OR these values together to specify more than one.

`netIOFlagOutOfBand`	Process out-of-band data. Available for send calls only.
`netIOFlagPeek`	Peek at incoming message without dequeuing it.
`netIOFlagDontRoute`	Send without using routing. This constant is currently ignored.

Tracing Bits

The tracing bits are used to set the level of event tracing. An application can get a list of events in the trace buffer using the `NetLibMaster` call.

You can set the tracing for each network interface using `NetLibIFSettingSet` and for the net library in general with `NetLibSettingSet`.

`netTracingErrors`	Record run-time errors. This is the default.
`netTracingMsgs`	Record application trace messages.
`netTracingPkts`	Record packets I/O.
`netTracingFuncs`	Record function flow.
`netTracingAppMsgs`	Record application messages sent using `NetLibTracePrintF` and `NetLibTracePutS`.

Net Library Functions

NetHToNL

Purpose Macro that converts a 32-bit value from host to network byte order.

Prototype `NetHToNL (x)`

Parameters -> `x` 32-bit value to convert.

Result Returns `x` in network byte order.

Sockets Equivalent `htonl()`

See Also `NetNToHS`, `NetNToHL`, `NetHToNS`

NetHToNS

Purpose Macro that converts a 16-bit value from host to network byte order.

Prototype `NetHToNS (x)`

Parameters -> `x` 16-bit value to convert.

Result Returns `x` in network byte order.

Sockets Equivalent `htons()`

See Also `NetNToHS`, `NetNToHL`, `NetHToNL`

NetLibAddrAToIN

Purpose Converts an ASCII string representing a dotted decimal IP address into a 32-bit IP address in network byte order.

Prototype

```
NetIPAddr NetLibAddrAToIN (UInt16 libRefnum,
Char *a)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `a`	Pointer to ASCII dotted decimal string.

Result Returns a 32-bit network byte order IP address or -1 if `a` doesn't represent a dotted decimal IP address

Sockets Equivalent

```
UInt32 inet_addr(char* cp)
```

See Also `NetLibAddrINToA`

NetLibAddrINToA

Purpose Converts an IP address from 32-bit network byte order into a dotted decimal ASCII string.

Prototype

```
Char * NetLibAddrINToA (UInt16 libRefnum,
NetIPAddr inet, Char *spaceP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `inet`	32-bit IP address in network byte order.
<- `spaceP`	Buffer used to hold the return value.

Result Returns in `spaceP` the dotted decimal ASCII string representation of the IP address.

Sockets Equivalent `char* inet_ntoa(struct in_addr in)`

See Also NetLibAddrAToIN

NetLibClose

Purpose Closes the net library.

Prototype

```
Err NetLibClose (UInt16 libRefnum,
UInt16 immediate)
```

Parameters

-> `libRefnum`	Reference number of the net library.
-> `immediate`	If `true`, library will shut down immediately. If `false`, library will shut down only if close timer expires before another NetLibOpen is issued.

Result Returns one of the following values:

0	Success.
`netErrNotOpen`	Library was not open.
`netErrStillOpen`	Not really an error; returned if library is still in use by another application.

Sockets Equivalent None.

Comments Applications must call this function when they no longer need the net library. If the net library open count is greater than 1 before this call is made, the count is decremented and `netErrStillOpen` is returned. If the open count was 1, the library takes the following action:

- If `immediate` is `true`, the library shuts down immediately. All network interfaces are brought down, the net protocol stack task is terminated, and all memory used by the net library is freed.

- If `immediate` is `false`, a close timer is created and this call returns immediately without actually bringing the net library down. Instead it leaves it up and running but marks it as in the "close-wait" state. It remains in this state until either the timer expires or another `NetLibOpen` is issued. If the timer expires, the library is shut down. If another `NetLibOpen` call is issued before the timer expires (possibly by another application), the timer is cancelled and the library is marked as fully open.

In most cases, you should pass `false` for `immediate`. This allows the user to quit one Internet application and launch another within a short period of time without having to wait through the process of closing down and then re-establishing dial-up network connections.

See Also `NetLibOpen`, `NetLibOpenCount`

NetLibConnectionRefresh

Purpose This routine is a convenience call for applications. It checks the status of all connections and optionally tries to open any that were closed.

Prototype

```
Err NetLibConnectionRefresh (UInt16 refNum,
Boolean refresh, UInt8 *allInterfacesUpP,
UInt16 * netIFErrP)
```

Parameters

`-> refnum`	Reference number of the net library.
`-> refresh`	If `true`, any connections that aren't currently open are opened.
`<- allInterfacesUpP`	Set to `true` if all connections are open.
`<- netIFErrP`	First error encountered when reopening connections that were closed. (See `NetLibIFUp` for a list of possible values.)

Result Returns one of the following values:

0	Success.

`netErrBufTooSmall`

`netErrOutOfCmdBlocks`

`netErrNoInterfaces`

Sockets Equivalent

None.

Comments

This function determines whether a connection is up based on the internal status of the TCP/IP stack. To test the presence of a "physical connection" (phone line, modem, serial cable), a command should be sent once it's been determined that the logical connection is up. If the physical connection is broken, nothing returns and a timeout error eventually occurs.

NetLibDmReceive

Purpose

Receive data from a socket directly into a database record.

Prototype

```
Int16 NetLibDmReceive (UInt16 libRefNum,
NetSocketRef socket, void* recordP,
UInt32 recordOffset, UInt16 rcvLen, UInt16 flags,
void* fromAddrP, UInt16 *fromLenP, Int32 timeout,
Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
<- `recordP`	Pointer to beginning of record to receive data into. Must be locked for use.
-> `recordOffset`	Offset from beginning of record to read data into.
-> `rcvLen`	Maximum number of bytes to read.
-> `flags`	One or more `netIOFlagxxx` flags. See "I/O Flags."
<- `fromAddrP`	Pointer to buffer to hold address of sender (a `NetSocketAddrType` struct). Pass `NULL` if you don't need sender information.

<-> `fromLenP`	On entry, size of `fromAddrP` buffer. On exit, actual size of returned address in `fromAddrP`. Pass `NULL` if you don't need sender information.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result

Returns the number of bytes successfully received. If the return value is 0, the socket has been shut down by the remote host. If the return value is -1, an error has occurred and `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrWouldBlock`	
`netErrUserCancel`	
`netErrOutOfMemory`	

Comments

This call behaves similarly to `NetLibReceive` but reads the data directly into a database record, which is normally write-protected. The caller must pass a pointer to the start of the record and an offset into the record of where to start the read.

NetLibFinishCloseWait

Purpose Forces the net library to do a complete close if it's currently in the close-wait state.

Prototype

```
Err NetLibFinishCloseWait (UInt16 libRefnum)
```

Parameters

-> `libRefnum`	Reference number of the net library.

Result Returns one of the following values:

0	Success.
`netErrTimeout`	

Sockets Equivalent None.

Comments This call checks the current open state of the net library. If it's in the close-wait state (see `NetLibClose`), it forces the library to perform an immediate, complete close operation.

NetLibGetHostByAddr

Purpose Looks up a host name given its IP address.

Prototype

```
NetHostInfoPtr NetLibGetHostByAddr
(UInt16 libRefnum, UInt8 *addrP, UInt16 len,
UInt16 type, NetHostInfoBufPtr bufP,
Int32 timeout, Err *errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `addrP`	IP address of host to lookup.
-> `len`	Length, in bytes, of `*addrP`.
-> `type`	Type of `addrP`. See `NetSocketAddrEnum`.
<- `bufP`	Pointer to a `NetHostInfoBufType` struct in which to store the results of the lookup.

-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is 0.

Result Returns a pointer to the `NetHostInfoType` portion of `bufP` that contains results of the lookup. If the return value is 0, an error has occurred, and `errP` contains one of the following values:

0	No error
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.

`netErrDNSNameTooLong`

`netErrDNSBadName`

`netErrDNSLabelTooLong`

`netErrDNSAllocationFailure`

`netErrDNSTimeout`

`netErrDNSUnreachable`

`netErrDNSFormat`

`netErrDNSServerFailure`

`netErrDNSNonexistantName`

`netErrDNSNIY`

`netErrDNSRefused`

`netErrDNSImpossible`

`netErrDNSNoRRS`

`netErrDNSAborted`

`netErrDNSBadProtocol`

`netErrDNSTruncated`

`netErrDNSNoRecursion`

`netErrDNSIrrelevant`

`netErrDNSNotInLocalCache`

`netErrDNSNoPort`

Sockets Equivalent

```
struct hostent* gethostbyaddr (char* addr, int
len, int type);
```

Comments

This call queries the domain name server(s) to look up a host name given its IP address.

See Also

NetLibGetHostByName

NetLibGetHostByName

Purpose

Looks up a host IP address given a host name.

Prototype

```
NetHostInfoPtr NetLibGetHostByName
(UInt16 libRefnum, Char *nameP,
NetHostInfoBufPtr bufP, Int32 timeout, Err *errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `nameP`	Name of host to look up.
<- `bufP`	Pointer to a NetHostInfoBufType struct in which to store the results of the lookup.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is 0.

Result

Returns a pointer to the NetHostInfoType portion of `bufP`, which contains results of the lookup. If the return value is 0, an error has occurred and `errP` contains one of the following values:

0	No error
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrDNSNameTooLong`	
`netErrDNSBadName`	

```
netErrDNSLabelTooLong
netErrDNSAllocationFailure
netErrDNSTimeout
netErrDNSUnreachable
netErrDNSFormat
netErrDNSServerFailure
netErrDNSNonexistantName
netErrDNSNIY
netErrDNSRefused
netErrDNSImpossible
netErrDNSNoRRS
netErrDNSAborted
netErrDNSBadProtocol
netErrDNSTruncated
netErrDNSNoRecursion
netErrDNSIrrelevant
netErrDNSNotInLocalCache
netErrDNSNoPort
```

Sockets Equivalent

```
struct hostent *gethostbyname(char* name);
```

Comments

This call first checks the local name -> IP address host table in the net library preferences. If the entry is not found, it then queries the domain name server(s).

See Also

NetLibGetHostByAddr, NetLibGetMailExchangeByName

NetLibGetMailExchangeByName

Purpose Looks up the name of a host to use for a given mail exchange.

Prototype

```
Int16 NetLibGetMailExchangeByName
(UInt16 libRefNum, Char *mailNameP,
UInt16 maxEntries,
Char hostNames[][netDNSMaxDomainName+1],
UInt16 priorities[], Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `mailNameP`	Name of the mail exchange to look up.
-> `maxEntries`	Maximum number of host names to return.
<- `hostNames`	Array of character strings of length `netDNSMaxDomainName+1`. The host name results are stored in this array. This array must be able to hold at least `maxEntries` host names.
<- `priorities`	Array of `Word`s. The priorities of each host name found are stored in this array. This array must be at least `maxEntries` in length.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is less than 0.

Result Returns the number of entries successfully found. If the return value is a negative number, an error has occurred, and `errP` contains one of the following values:

0	No error
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrDNSNameTooLong`	
`netErrDNSBadName`	

`netErrDNSLabelTooLong`

`netErrDNSAllocationFailure`

`netErrDNSTimeout`

`netErrDNSUnreachable`

`netErrDNSFormat`

`netErrDNSServerFailure`

`netErrDNSNonexistantName`

`netErrDNSNIY`

`netErrDNSRefused`

`netErrDNSImpossible`

`netErrDNSNoRRS`

`netErrDNSAborted`

`netErrDNSBadProtocol`

`netErrDNSTruncated`

`netErrDNSNoRecursion`

`netErrDNSIrrelevant`

`netErrDNSNotInLocalCache`

`netErrDNSNoPort`

Sockets Equivalent

None

Comments

This call looks up the name(s) of host(s) to use for sending an e-mail. The caller passes the name of the mail exchange in `mailNameP` and gets back a list of host names to which the mail message can be sent.

See Also

`NetLibGetHostByAddr`, `NetLibGetHostByName`

NetLibGetServByName

Purpose

Looks up the port number for a standard TCP/IP service, given the desired protocol.

Prototype

```
NetServInfoPtr NetLibGetServByName
(UInt16 libRefnum, const Char *servNameP,
const Char *protoNameP, NetServInfoBufPtr bufP,
Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `servNameP`	Name of the service to look up. Possible services are "echo", "discard", "daytime", "qotd", "chargen", "ftp-data", "ftp", "telnet", "smtp", "time", "name", "finger", "pop2", "pop3", "nntp", "imap2".
-> `protoNameP`	Desired protocol to use, either "udp" or "tcp".
<- `bufP`	Pointer to a `NetServInfoBufType` struct in which to store the results of the lookup.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is 0.

Result

Returns a pointer to the `NetServInfoType` portion of `bufP` that contains results of the lookup. If the return value is 0, and error has occurred and `errP` contains one of the following values:

0	No error
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrUnknownProtocol`	
`netErrUnknownService`	

Sockets Equivalent

```
struct servent* getservbyname(char* addr,
char* proto);
```

Comments This call is a convenience call for looking up a standard port number given the name of a service and the protocol to use.

See Also NetLibGetHostByName

NetLibIFAttach

Purpose Attach a new network interface.

Prototype

```
Err NetLibIFAttach (UInt16 libRefnum,
UInt32 ifCreator, UInt16 ifInstance,
Int32 timeout)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `ifCreator`	Creator of interface to attach.
-> `ifInstance`	Instance number of interface to attach. The instance number is one of the values returned by NetLibIFGet.
-> `timeout`	Timeout in ticks; -1 means infinite timeout.

Result Returns one of the following values:

0	Success.
`netErrInterfaceNotFound`	
`netErrTooManyInterfaces`	

Sockets Equivalent None

Comments This call can be used to attach a new network interface to the net library. Network interfaces are self-contained databases of type `'neti'`. The `ifCreator` parameter to this function is used to locate the network interface database of the given creator.

If the net library is already open when this call is made, the network interface's database will be located and then called to initialize itself and attach itself to the protocol stack in real time. If the net library is not open when this call is made, the creator and instance number of

the interface are stored in the net library's preferences database and the interface is initialized and attached to the stack the next time the net library is opened.

See Also NetLibIFGet, NetLibIFDetach

NetLibIFDetach

Purpose Detach a network interface from the protocol stack.

Prototype

```
Err NetLibIFDetach (UInt16 libRefnum,
UInt32 ifCreator, UInt16 ifInstance,
Int32 timeout)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `ifCreator`	Creator of interface to detach.
-> `ifInstance`	Instance number of interface to detach.
-> `timeout`	Timeout in ticks; -1 means infinite timeout.

Result Returns one of the following values:

0	Success.
`netErrInterfaceNotFound`	

Sockets Equivalent None

Comments If the net library is already open when this call is made, the interface is brought down and detached from the protocol stack in real time. If the net library is not open when this call is made, the creator and instance number of the interface are removed in the net library's preferences database and the interface is not attached the next time the library is opened.

See Also NetLibIFGet, NetLibIFAttach

NetLibIFDown

Purpose Bring an interface down and hang up a connection.

Prototype

```
Err NetLibIFDown (UInt16 libRefnum,
UInt32 ifCreator, UInt16 ifInstance,
Int32 timeout)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `ifCreator`	Creator of interface to attach.
-> `ifInstance`	Instance number of interface to attach.
-> `timeout`	Timeout in ticks; -1 means wait forever.

Result Returns one of the following values:

0	Success.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrInterfaceNotFound`	

Sockets Equivalent None

Comments The net library must be open before this call can be made. For dial-up interfaces, this call terminates a connection and hangs up the modem if necessary.

`NetLibClose` automatically brings down any attached interfaces, so this routine doesn't normally have to be called.

If the interface is already down, this routine returns immediately with no error.

See Also `NetLibIFGet`, `NetLibIFAttach`, `NetLibIFDetach`, `NetLibIFUp`

NetLibIFGet

Purpose Get the creator and instance number of an installed interface by index.

Prototype

```
Err NetLibIFGet (UInt16 libRefnum, UInt16 index,
UInt16 * ifCreatorP, UInt16 * ifInstanceP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `index`	Index of the interface to get. Indices start at 0.
<- `ifCreatorP`	The interface's creator.
<- `ifInstanceP`	The interface's instance number.

Result Returns one of the following values:

0	Success.
`netErrInvalidInterface`	Index too high
`netErrPrefNotFound`	No current value for setting.

Sockets Equivalent None

Comments To get a list of all installed interfaces, call this function with successively increasing indices starting from 0 until the error `netErrInvalidInterface` is returned.

The `ifCreator` and `ifInstance` values returned from this call can then be used with the `NetLibSettingGet` call to get more information about that particular interface.

See Also `NetLibIFAttach`, `NetLibIFDetach`, "Settings for Interface Selection" in the *Palm OS Programmer's Companion*

NetLibIFSettingGet

Purpose Retrieves a network interface specific setting.

Prototype

```
Err NetLibIFSettingGet (UInt16 libRefnum,
UInt32 ifCreator, UInt16 ifInstance,
UInt16 setting, void *valueP, UInt16 *valueLenP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `ifCreator`	Creator of the network interface.
-> `ifInstance`	Instance number of the network interface.
-> `setting`	Setting to retrieve; one of the `NetIFSettingEnum` constants.
<- `valueP`	Space for return value of setting.
<-> `valueLenP`	On entry, size of `valueP`. On exit, actual size of setting.

Result Returns one of the following values:

0	Success.
`netErrUnknownSetting`	Invalid setting constant.
`netErrPrefNotFound`	No current value for setting.
`netErrBufTooSmall`	`valueP` was too small to hold entire setting. Setting value was truncated to fit in `valueP`.
`netErrUnimplemented`	
`netErrInterfaceNotFound`	
`netErrBufWrongSize`	

Sockets Equivalent None

Comments This call can be used to retrieve the current value of any network interface setting. The caller must pass a pointer to a buffer to hold the return value (`valueP`), the size of the buffer (`*valueLenP`), and the setting ID (`setting`). The setting ID is one of the constants in the `NetIFSettingEnum` type.

Some settings, such as the login script, are variable size. For these types of settings, you can obtain the actual size required for the buffer by passing 0 for `*valueLenP`. The required size is returned in `valueLenP`.

Table 54.1 lists the network interface settings and the size of each setting. Some are only applicable to certain types of interfaces. Settings not applicable to a specific interface can be safely ignored and not set to any particular value.

Table 54.1 Network Interface Settings

netIFSetting...	Type	Description
ResetAll	void	Use with `NetLibIFSettingSet` only. This clears all other settings for the interface to their default values.
Up	UInt8	Read-only. `true` if interface is currently up.
Name	Char[32]	Read-only. Name of this interface.
ReqIPAddr	UInt32	IP address of interface.
SubnetMask	UInt32	Subnet mask for interface. Doesn't need to be specified for PPP or SLIP type connections.
Broadcast	UInt32	Broadcast address for interface. Doesn't need to be specified for PPP or SLIP type connections.
Username	Char[32]	User name. Only required if the login script uses the user name substitution escape sequence in it. Call `NetLibIFSettingSet` with a `valueLen` of 0 to remove this setting.
Password	Char[32]	Password. Only required if the login script uses the password substitution escape sequence in it. Call `NetLibIFSettingSet` with a `valueLen` of 0 to remove this setting. If the login script uses password substitution and no password setting is set, the user will be prompted for a password at connect time.
AuthUsername	Char[32]	Authentication user name. Only required if the authentication protocol uses a different user name than the what's in the `netIFSettingUsername` setting. If this setting is empty (`valueLen` of 0), the Username setting will be used instead. Call `NetLibIFSettingSet` with a `valueLen` of 0 to remove this setting.

Table 54.1 Network Interface Settings *(continued)*

netIFSetting...	Type	Description
AuthPassword	Char[32]	Authentication password. If "$" then the user will be prompted for the authentication password at connect time. Else, if 0 length, then the `netIFSettingPassword` setting or the result of its prompt will be used instead. Call `NetLibIFSettingSet` with a `valueLen` of 0 to remove this setting.
ServiceName	Char[]	Service name. Used for display purposes while showing the connection progress dialog box. Call `NetLibIFSettingSet` with a `valueLen` of 0 to remove this setting.
LoginScript	Char[]	Login script. Only required if the particular service requires a login sequence. Call `NetLibIFSettingSet` with a `valueLen` of 0 to remove this setting. See below for a description of the login script format.
ConnectLog	Char[]	Connect log. Generally, this setting is just retrieved, not set. It contains a log of events from the most recent login. To clear this setting, call `NetLibIFSettingSet` with a `valueLen` of 0.
InactivityTimeout	UInt16	Maximum number of seconds of inactivity allowed. Set to 0 to ignore.
EstablishmentTimeout	UInt16	Maximum delay, in seconds, allowed between each stage of connection establishment or login script line. Must be non-zero.
DynamicIP	UInt8	If non-zero, negotiate for an IP address. If `false`, the IP address specified in the `netIFSettingReqIPAddr` setting will be used. Default is `false`.
VJCompEnable	UInt8	If non-zero, enable VJ header compression. Default is `true` for PPP, `false` for SLIP, and `true` for CSLIP.

Table 54.1 Network Interface Settings *(continued)*

netIFSetting...	Type	Description
VJCompSlots	UInt8	Number of slots to use for VJ compression. Default is 4 for PPP and 16 for SLIP and CSLIP. More slots require more memory so it is best to keep this number to a minimum.
MTU	UInt16	Maximum transmission unit in octets. Currently not implemented in SLIP or PPP.
AsyncCtlMap	UInt32	Bit mask of characters to escape for PPP. Default is 0.
PortNum	UInt16	Which serial communication port to use. Port 0 is the only port available on the device.
BaudRate	UInt32	Serial port baud rate to use in bits per second.
FlowControl	UInt8	If bit 0 is 1, use hardware handshaking on the serial port. Default is no hardware handshaking.
StopBits	UInt8	Number of stop bits. Default is 1.
ParityOn	UInt8	`true` if parity detection enabled. Default is `false`.
ParityEven	UInt8	`true` for even parity detection. Default is `true`.
UseModem	UInt8	If `true`, dial-up through modem. If `false`, go direct over serial port
PulseDial	UInt8	If `true`, pulse dial modem. Else, tone dial. Default is tone dial.
ModemInit	Char[]	Zero-terminated modem initialization string, not including the "AT". If not specified (`valueLen` of 0), the modem initialization string from system preferences are used.
ModemPhone	Char[]	Zero-terminated modem phone number string. Only required if `netIFSettingUseModem` is true.

Table 54.1 Network Interface Settings *(continued)*

netIFSetting...	Type	Description
RedialCount	UInt16	Number of times to redial modem when trying to establish a connection. Only required if `netIFSettingUseModem` is true.
DNSQuery	UInt8	`true` if PPP queries for DNS address. The default is `true`.
TraceBits	UInt32	A bitfield of various trace bits. See "Tracing Bits." An application can get a list of events in the trace buffer using the `NetLibMaster` call. Each interface has its own trace bits setting so that trace event recording in each interface can be selectively enabled or disabled.
ActualIPAddr	UInt32	Read-only. The actual IP address that the interface ends up using. The login script execution engine stores the result of the "g" (get IP address) command here as does the PPP negotiation logic.
ServerIPAddr	UInt32	Read-only. The IP address of the PPP server we're connected to.
BringDownOnPower Down	UInt8	`true` if the interface is brought down when the Palm OS® device is turned off.
RawMode	UInt32	Specifies if the interface is in raw mode. The net library places an interface in raw mode when it is bound to a raw socket in the raw domain. Raw sockets are available in Palm OS version 3.0 and higher.

See Also `NetLibIFSettingSet`, `NetLibSettingGet`, `NetLibSettingSet`, "Interface Specific Settings" in the *Palm OS Programmer's Companion*

NetLibIFSettingSet

Purpose Sets a network interface specific setting.

Prototype

```
Err NetLibIFSettingSet (UInt16 libRefnum,
UInt32 ifCreator, UInt16 ifInstance,
UInt16 setting, void* valueP, UInt16 valueLen)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `ifCreator`	Creator of the network interface.
-> `ifInstance`	Instance number of the network interface.
-> `setting`	The setting to set, one of the `NetIFSettingEnum` constants. See Table 54.1.
-> `valueP`	Space new value of setting.
-> `valueLen`	Size of new setting.

Result Returns one of the following values:

0	Success.
`netErrUnknownSetting`	Invalid setting constant.
`netErrPrefNotFound`	No current value for setting.
`netErrBufTooSmall`	`valueP` was too small to hold entire setting. Setting value was truncated to fit in `valueP`.
`netErrUnimplemented`	
`netErrInterfaceNotFound`	
`netErrBufWrongSize`	
`netErrReadOnlySetting`	

Sockets Equivalent None

Comments This call can be used to set the current value of any network interface setting. The caller must pass a pointer to a buffer which holds the new value (`valueP`), the size of the buffer (`valueLen`), and the setting ID (`setting`).

See `NetLibIFSettingGet` for an explanation of each of the settings.

Of particular interest is the `netIFSettingResetAll` setting, which, if used, resets all settings for the interface to their default values. When using this setting, `valueP` and `valueLen` are ignored.

See Also `NetLibIFSettingGet`, `NetLibSettingGet`, `NetLibSettingSet`, "Interface Specific Settings" in the *Palm OS Programmer's Companion*

NetLibIFUp

Purpose Bring an interface up and establish a connection.

Prototype

```
Err NetLibIFUp (UInt16 libRefnum,
UInt32 ifCreator, UInt16 ifInstance)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `ifCreator`	Creator of interface to attach.
-> `ifInstance`	Instance number of interface to attach.

Result Returns one of the following values:

0	Success.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrInterfaceNotFound`	
`netErrUserCancel`	
`netErrBadScript`	
`netErrPPPTimeout`	

`netErrAuthFailure`

`netErrPPPAddressRefused`

Sockets Equivalent

None

Comments

The net library must be open before this call can be made. For dial-up interfaces, this call will dial up the modem if necessary and run through the connect script to establish the connection.

If the interface is already up, this routine returns immediately with no error. This call doesn't take a timeout parameter because it relies on each interface to have its own established timeout setting.

See Also

`NetLibIFGet`, `NetLibIFAttach`, `NetLibIFDetach`, `NetLibIFDown`

NetLibMaster

Purpose

Retrieves the network statistics, interface statistics, and the contents of the trace buffer.

Prototype

```
Err NetLibMaster (UInt16 libRefnum, UInt16 cmd,
NetMasterPBPtr pbP, Int32 timeout)
```

Parameters

-> `libRefNum`	Reference number of the net library.

-> `cmd`	Function to perform (`NetMasterEnum` type). The following commands are supported:
	`netMasterInterfaceInfo`
	`netMasterInterfaceStats`
	`netMasterIPStats`
	`netMasterICMPStats`
	`netMasterUDPStats`
	`netMasterTCPStats`
	`netMasterTraceEventGet`

<-> `pbP`	Command parameter block.

-> `timeout`	Timeout in ticks; -1 means wait forever.

Result

Returns one of the following values:

0	No error
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrUnimplemented`	

Sockets Equivalent

None

Comments

This call allows applications to get detailed information about the net library. This information is usually helpful in debugging network configuration problems.

This function takes a command word (`cmd`) and parameter block pointer (`pbP`) as arguments and returns its results in the parameter block on exit. Which values you must specify in the parameter block and which values are returned are specific to the command you specify.

netMasterInterfaceInfo

The `pbP->interfaceInfo` struct specifies interface information.

->	`index`	Index of interface to fetch info about.
<-	`creator`	Creator of interface.
<-	`instance`	Instance of interface.
<-	`netIFP`	Private interface info pointer.
<-	`drvrName`	Driver type that interface uses (“PPP”, “SLIP”, etc.).
<-	`hwName`	Hardware driver name (“Serial Library”, etc.).

<-	`localNetHdrLen`	Number of bytes in local net header.
<-	`localNetTrailerLen`	Number of bytes in local net trailer.
<-	`localNetMaxFrame`	Local net maximum frame size.
<-	`ifName`	Interface name with instance number concatenated.
<-	`driverUp`	`true` if interface driver is up.
<-	`ifUp`	`true` if interface media layer is up.
<-	`hwAddrLen`	Length of interface's hardware address.
<-	`hwAddr`	Interface's hardware address.
<-	`mtu`	Maximum transfer unit of interface.
<-	`speed`	Speed in bits per second.
<-	`lastStateChange`	Time in milliseconds of last state change.
<-	`ipAddr`	IP address of interface.
<-	`subnetMask`	Subnet mask of local network.
<-	`broadcast`	Broadcast address of local network.

netMasterInterfaceStats

The `pbP->interfaceStats` structure specifies interface statistics.

->	`index`	Index of interface to fetch info about.
<-	`inOctets`	Number of octets received.
<-	`inUcastPkts`	Number of packets received.
<-	`inNUcastPkts`	Number of broadcast packets received.
<-	`inDiscards`	Number of incoming packets that were discarded.

<-	inErrors	Number of packet errors encountered.
<-	inUnknownProtos	Number of unknown protocols encountered.
<-	outOctets	Number octets sent.
<-	outUcastPkts	Number of packets sent.
<-	outNUcastPkts	Number of broadcast packets sent.
<-	outDiscards	Number of packets discarded.
<-	outErrors	Number of outbound packet errors.

netMasterIPStats

The `pbP->ipStats` structure contains statistics about the IP protocol. See `NetMgr.h` for a complete list of statistics returned.

netMasterICMPStats

The `pbP->icmpStats` structure contains statistics about the ICMP protocol. See `NetMgr.h` for a complete list of statistics returned.

netMasterUDPStats

The `pbP->udpStats` structure contains statistics about the UDP protocol. See `NetMgr.h` for a complete list of statistics returned.

netMasterTCPStats

The `pbP->tcpStats` structure contains statistics about the TCP protocol. See `NetMgr.h` for a complete list of statistics returned.

netMasterTraceEventGet

The `pbP->traceEventGet` structure contains a trace event.

->	index	Index of event to fetch.
<-	textP	Pointer to text string to return event in. Should be at least 256 bytes long.

See Also NetLibSettingSet

NetLibOpen

Purpose Opens and initializes the net library.

Prototype

```
Err NetLibOpen (UInt16 libRefnum,
UInt16 *netIFErrP)
```

Parameters

-> `libRefnum` Reference number of the net library.

<- `netIFErrP` First error encountered when bringing up network interfaces. (See `NetLibIFUp` for a list of possible values.)

Result Returns one of the following values:

0 No error.

`netErrAlreadyOpen`
Not really an error; returned if library was already open and the open count was simply incremented.

`netErrOutOfMemory`
Not enough memory available to open the library.

`netErrNoInterfaces`
Incorrect setup.

`netErrPrefNotFound`
Incorrect setup.

Comments Applications must call this function before using the net library. If the net library was already open, `NetLibOpen` increments its open count. Otherwise, it opens the library, initializes it, starts up the net protocol stack component of the library as a separate task, and brings up all attached network interfaces.

`NetLibOpen` uses settings saved in the net library's preferences database during initialization. These settings include the interfaces to attach, the IP addresses, etc. It's assumed that these settings have been previously set up by a preference panel or equivalent so an

application doesn't normally have to set them up before calling `NetLibOpen`.

If any of the attached interfaces fails to come up, `*netIFErrP` will contain the error number of the first interface that encountered a problem.

See Also `SysLibFind`, `NetLibClose`, `NetLibOpenCount`

NetLibOpenCount

Purpose Retrieves the open count of the net library.

Prototype

```
Err NetLibOpenCount (UInt16 libRefnum,
UInt16 *countP)
```

Parameters

-> `libRefnum`	Reference number of the net library.
<- `countP`	Contains the open count of the net library upon return.

Result Always returns 0.

Sockets Equivalent None.

Comments This call will most likely only be used by the Network preferences panel. Most applications will simply call `NetLibOpen` unconditionally during startup and `NetLibClose` when they exit.

NetLibReceive

Purpose Receive data from a socket into a single buffer.

Prototype

```
Int16 NetLibReceive (UInt16 libRefNum,
NetSocketRef socket, void* bufP, UInt16 bufLen,
UInt16 flags, void* fromAddrP, UInt16 * fromLenP,
Int32 timeout, Err* errP);
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
<- `bufP`	Pointer to buffer to hold received data.
-> `bufLen`	Length of `bufP` buffer.
-> `flags`	One or more `netIOFlagxxx` flags. See "I/O Flags."
<- `fromAddrP`	Pointer to buffer to hold address of sender (a `NetSocketAddrType`).
<-> `fromLenP`	On entry, size of `fromAddrP` buffer. On exit, actual size of returned address in `fromAddrP`.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns the number of bytes successfully received. If the return value is 0, the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	

`netErrWouldBlock`

`netErrUserCancel`

Sockets Equivalent

```
int recvfrom (int socket, const void* bufP,
int bufLen, int flags, const void* fromAddrP,
int* fromLenP);
```

```
int recv(int socket, const void* bufP, int bufLen,
int flags);
```

```
int read(int socket, const void* bufP,
int bufLen);
```

Comments

For stream-based sockets, this call reads whatever bytes are available and returns the number of bytes actually read into the caller's buffer. If there is no data available, this call will block until at least one byte arrives, until the socket is shut down by the remote host, or until a timeout occurs.

For datagram-based sockets, this call reads a complete datagram and returns the number of bytes in the datagram. If the caller's buffer is not large enough to hold the entire datagram, the end of the datagram is discarded. If a datagram is not available, this call will block until one arrives, or until the call times out.

The data is read into a single buffer pointed to by `bufP`.

See Also

`NetLibReceive`, `NetLibDmReceive`, `NetUReadN`, `NetLibSend`, `NetLibSendPB`

NetLibReceivePB

Purpose

Receive data from a socket into a multi-buffer gather-read array.

Prototype

```
Int16 NetLibReceivePB (UInt16 libRefnum,
NetSocketRef socket, NetIOParamType* pbP,
UInt16 flags, Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.

-> `pbP`	Pointer to parameter block containing buffer info.
-> `flags`	One or more `netIOFlagxxx` flags. See "I/O Flags."
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result

Returns the number of bytes successfully received. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrWouldBlock`	

Sockets Equivalent

```
int recvmsg (int socket, const struct msghdr* pbP,
int flags);
```

Comments

The `pbP` parameter is a pointer to a `NetIOParamType` structure. `NetIOParamType` is defined as follows:

```
typedef struct {
  UInt8 *      addrP;
  UInt16       addrLen;
  NetIOVecPtr  iov;
  UInt16       iovLen;
  UInt8 *      accessRights;
  UInt16       accessRightsLen;
} NetIOParamType, *NetIOParamPtr;
```

You provide the following information in this struct:

`addrP`	Address of sender, set by `NetLibReceivePB`. Set to 0 if you don't require this field.
`addrLen`	Length of `*addrP`.
`iov`	Array of buffers into which the data should be received. `NetIOVecPtr` is a pointer to a `NetIOVecType` structure, which has two fields:
	`bufP` — Pointer to a buffer.
	`bufLen` — Length of `bufP`.
`iovLen`	Length of the `iov` array.
`accessRights`	Access rights. This field currently isn't used and should be set to 0.
`accessRightsLen`	Length of the `*accessRights`. This field currently isn't used and should be set to 0.

For stream-based sockets, this call reads whatever bytes are available and returns the number of bytes actually read into the caller's buffer. If no data is available, this call will block until at least one byte arrives, until the socket is shut down by the remote host, or until a timeout occurs.

For datagram-based sockets, this call reads a complete datagram and returns the number of bytes in the datagram. If the caller's buffer is not large enough to hold the entire datagram, the end of the datagram is discarded. If a datagram is not available, this call will block until one arrives, or until the call times out.

The data is read into the gather-read array specified by the `pbP->iov` array.

See Also `NetLibReceive`, `NetLibDmReceive`, `NetLibSend`, `NetLibSendPB`

NetLibSelect

Purpose

Blocks until I/O is ready on one or more descriptors, where a descriptor can represent socket input, socket output, or a user input event like a pen tap or key press.

Prototype

```
Int16 NetLibSelect (UInt16 libRefnum,
UInt16 width, NetFDSetType* readFDs,
NetFDSetType* writeFDs, NetFDSetType* exceptFDs,
Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `width`	Number of descriptor bits to check in the `readFDs`, `writeFDs`, and `exceptFDs` descriptor sets.
<-> `readFDs`	Pointer to 32-bit `NetFDSetType` containing set of bits representing descriptors to check for input.
<-> `writeFDs`	Pointer to 32-bit `NetFDSetType` containing set of bits representing descriptors to check for output.
<-> `exceptFDs`	Pointer to 32-bit `NetFDSetType` containing set of bits representing descriptors to check for exception conditions. This parameter is ignored. Upon return, its bits are always cleared.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result

Returns the sum total number of ready file descriptors in `*readFDs`, `*writeFDs`, and `*exceptFDs`. Returns 0 upon timeout. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0	No error
`netErrTimeout`	Call timed out.

`netErrNotOpen`	The referenced net library has not been opened yet.

Sockets Equivalent

```
int select (int width, fd_set* readfds,
fd_set* writefds, fd_set* exceptfds,
struct timeval* timeout);
```

Comments

This call blocks until one or more descriptors are ready for I/O. In the Palm OS environment, a descriptor is either a `NetSocketRef` or the "stdin" descriptor, `sysFileDescStdIn`. The `sysFileDescStdIn` descriptor will be ready for input whenever a user event is available like a pen tap or key press.

The caller should set which bits in each descriptor set need to be checked by using the `netFDZero` and `netFDSet` macros. After this call returns, the macro `netFDIsSet` can be used to determine which descriptors in each set are actually ready.

On exit, the total number of ready descriptors is returned and each descriptor set is updated with the appropriate bits set for each ready descriptor in that set.

The following example illustrates how to use this call to check for input on a socket or a user event:

```
Err     err;
NetSocketRef   socket;
NetFDSetType   readFDs,writeFDs,exceptFDs;
Int16       numFDs;
UInt16       width;

// Create the descriptor sets
netFDZero(&readFDs);
netFDZero(&writeFDs);
netFDZero(&exceptFDs);
netFDSet(sysFileDescStdIn, &readFDs);
netFDSet(socket, &readFDs);

// Calculate the max descriptor number and
// use that +1 as the max width.
// Alternatively, we could simply use the
// constant netFDSetSize as the width which
```

```
// is simpler but makes the NetLibSelect call
// slightly slower.
width = sysFileDescStdIn;
if (socket > width) width = socket;

// Wait for any one of the descriptors to be
// ready.
numFDs = NetLibSelect(AppNetRefnum, width+1,
  &readFDs, &writeFDs, &exceptFDs,
  AppNetTimeout, &err);
```

Also see the `NetSample` example application in the `Palm OS Examples` folder. The function `CmdTelnet` in the file `CmdTelnet.c` shows how to use the Berkeley sockets `select` function and how to interpret the results.

See Also `NetLibSocketOptionSet`

NetLibSend

Purpose Send data to a socket from a single buffer.

Prototype

```
Int16 NetLibSend (UInt16 libRefNum,
NetSocketRef socket, void* bufP, UInt16 bufLen,
UInt16 flags, void* toAddrP, UInt16 toLen,
Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
-> `bufP`	Pointer to data to write.
-> `bufLen`	Length of data to write
-> `flags`	One or more `netIOFlagxxx` flags. See "I/O Flags."
-> `toAddrP`	Address to send to (a pointer to a `NetSocketAddrType`), or 0.
-> `toLen`	Size of `toAddrP` buffer.

-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns the number of bytes successfully sent. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrMessageTooBig`	
`netErrSocketNotConnected`	
`netErrSocketClosedByRemote`	
`netErrIPCantFragment`	
`netErrIPNoRoute`	
`netErrIPNoSrc`	
`netErrIPNoDst`	
`netErrIPktOverflow`	
`netErrOutOfCmdBlocks`	
`netErrOutOfPackets`	
`netErrInterfaceNotFound`	
`netErrInterfaceDown`	
`netErrUnreachableDest`	
`netErrNoMultiPktAddr`	
`netErrWouldBlock`	

Sockets Equivalent

```
int sendto(int socket, const void* bufP,
int bufLen, int flags, const void* toAddrP,
int toLen);
```

```
int send(int socket, const void* bufP, int bufLen,
int flags);
```

```
int write(int socket, const void* bufP,
int bufLen,);
```

Comments

This call attempts to write data to the specified socket and returns the number of bytes actually sent, which may be less than or equal to the requested number of bytes. The data is passed in a single buffer that `bufP` points to.

For datagram sockets, you must only send a single packet at a time. If the data is too large to fit in a single UDP packet (1536 bytes), no data is sent and -1 is returned.

The `toAddrP` field applies only to datagram sockets without an existing connection. An error is returned if the datagram socket was previously connected and `toAddrP` is specified. Stream-based sockets, by definition, must have a connection established with a remote host before data can be written. Raw sockets (supported in Palm OS version 3.0 and higher) must construct the entire IP header, including the destination address, before data can be sent; thus, the address is taken from the data to be sent.

If there isn't enough buffer space to send any data, this call will block until there is enough buffer space, or until a timeout.

NOTE: For stream-based sockets, this call may write only a portion of the desired data. It always returns the number of bytes actually written. Consequently, the caller should be prepared to call this routine repeatedly until the desired number of bytes have been written, or until it returns 0 or -1.

See Also

`NetLibSendPB`, `NetUWriteN`, `NetLibReceive`, `NetLibReceivePB`, `NetLibDmReceive`

NetLibSendPB

Purpose Send data to a socket from a scatter-write array.

Prototype

```
Int16 NetLibSendPB (UInt16 libRefnum,
NetSocketRef socket, NetIOParamType* pbP,
UInt16 flags, Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
-> `pbP`	Pointer to parameter block containing buffer info. See the description in `NetLibReceivePB`.
-> `flags`	One or more `netIOFlagxxx` flags. See "I/O Flags."
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns the number of bytes successfully sent. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrMessageTooBig`	
`netErrSocketNotConnected`	
`netErrSocketClosedByRemote`	
`netErrIPCantFragment`	

```
netErrIPNoRoute
netErrIPNoSrc
netErrIPNoDst
netErrIPktOverflow
netErrOutOfCmdBlocks
netErrOutOfPackets
netErrInterfaceNotFound
netErrInterfaceDown
netErrUnreachableDest
netErrNoMultiPktAddr
netErrWouldBlock
```

Sockets Equivalent

```
int sendmsg(int socket, const struct msghdr* pbP,
int flags);
```

Comments

This call attempts to write data to the given socket and returns the number of bytes actually sent, which may be less than or equal to the requested number of bytes. The data is passed in the scatter-write array specified in the `pbP` parameter block.

For datagram sockets, you must only send a single packet at a time. If the data is too large to fit in a single UDP packet, no data is sent and -1 is returned.

The `toAddrP` field applies only to datagram sockets without an existing connection. An error is returned if the datagram socket was previously connected and `toAddrP` is specified. Stream-based sockets, by definition, must have a connection established with a remote host before data can be written. Raw sockets (supported in Palm OS version 3.0 and higher) must construct the entire IP header, including the destination address, before data can be sent; thus, the address is taken from the data to be sent.

If there isn't enough buffer space to send any data, this call will block until there is space, or until a timeout.

NOTE: For stream-based sockets, this call may write only a portion of the desired data. It always returns the number of bytes actually written. Consequently, the caller should be prepared to call this routine repeatedly until the desired number of bytes have been written, or until it returns 0 or -1.

See Also `NetLibSend`, `NetLibReceive`, `NetLibReceivePB`, `NetLibDmReceive`

NetLibSettingGet

Purpose Retrieves a general setting.

Prototype

```
Err NetLibSettingGet (UInt16 libRefnum,
UInt16 setting, void* valueP, UInt16* valueLenP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `setting`	Setting to retrieve, one of the `NetSettingEnum` constants.
<- `valueP`	Space for return value of setting.
<-> `valueLenP`	On entry, size of `valueP`. On exit, actual size of setting.

Result Returns one of the following values:

0	Success.
`netErrUnknownSetting`	Invalid setting constant
`netErrPrefNotFound`	No current value for setting
`netErrBufTooSmall`	`valueP` was too small to hold entire setting. Setting value was truncated to fit in `valueP`.
`netErrBufWrongSize`	

Sockets Equivalent None

Comments This call retrieves the current value of any general setting. The caller must pass a pointer to a buffer to hold the return value (`valueP`), the size of the buffer (`*valueLenP`), and the setting ID (`setting`). The setting ID is one of the `NetSettingEnum` constants in the `netSettingEnum` type.

Some settings, such as the host table, are variable size. For these types of settings, you can obtain the actual size required for the buffer by passing 0 for `*valueLenP`. The required size is returned in `valueLenP`.

Table 54.2 lists the general settings and the type of each setting.

Table 54.2 Net Library General Settings

netSetting...	Type	Description
ResetAll	void	Used for `NetLibSettingSet` only. This will clear all other settings to their default values.
PrimaryDNS	UInt32	IP address of primary DNS server. This setting must be set to a non-zero IP address in order to support any of the name lookup calls.
SecondaryDNS	UInt32	IP address of secondary DNS server. Set to 0 to have stack ignore this setting.
DefaultRouter	UInt32	IP address of default router. Default value is 0 which is appropriate for most implementations with only one attached interface (besides loopback). Packets with destination IP addresses that don't lie in the subnet of an attached interface will be sent to this router through the default interface specified by the `netSettingDefaultIFCreator/ netSettingDefaultIFInstance` pair.
DefaultIFCreator	UInt32	Creator of the default network interface. Default value is 0, which is appropriate for most implementations. Packets with destination IP addresses that don't lie in the subnet of a directly attached interface are sent through this interface. If this setting is 0, the stack automatically makes the first non-loopback interface the default interface.
DefaultIFInstance	UInt16	Instance number of the default network interface. Packets with destination IP addresses that don't lie in the subnet of an attached interface are sent through the default interface. Default value is 0.
HostName	Char[]	A zero-terminated character string of 64 bytes or less containing the host name of this machine. This setting is not actually used by the stack. It's present mainly for informative purposes and to support the `gethostname/sethostname` sockets API calls. To clear the host name, call `NetLibIFSettingSet` with a valueLen of 0.

Table 54.2 Net Library General Settings *(continued)*

netSetting...	Type	Description
DomainName	Char[]	A zero-terminated character string of 256 bytes or less containing the default domain. This default domain name is appended to all host names before name lookups are performed. If the name is not found, the host name is looked up again without appending the domain name to it. To have the stack not use the domain name, call `NetLibIFSettingSet` with a `valueLen` of 0.
HostTbl	Char[]	A null-terminated character string containing the host table. This table is consulted first before sending a DNS query to the DNS server(s). To have the stack not use a host table, call `NetLibIFSettingSet` with a `valueLen` of 0. The format of a host table is a series of lines separated by '\n' in the following format: `host.company.com A 111.222.333.444`
CloseWaitTime	UInt32	The close-wait time in milliseconds. This setting must be specified. See the discussion of the `NetLibClose` call for an explanation of the close-wait time.
TraceBits	UInt32	A bitfield of various trace bits. See "Tracing Bits." Default value is (`netTracingErrors | netTracingAppMsgs`). An application can get a list of events in the trace buffer using the `NetLibMaster` call.
TraceSize	UInt32	Maximum trace buffer size in bytes. Setting this setting always clears the existing trace buffer. Default is 2 KB.
TraceRoll	UInt8	Boolean value, default is `true` (non-zero). If `true`, trace buffer will roll over when it fills. If `false`, tracing will stop as soon as trace buffer fills.

See Also `NetLibSettingSet`, `NetLibIFSettingSet`, `NetLibIFSettingGet`, `NetLibMaster`

NetLibSettingSet

Purpose Sets a general setting.

Prototype

```
Err NetLibSettingSet (UInt16 libRefnum,
UInt16 setting, void* valueP, UInt16 valueLen)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `setting`	Setting to set; one of the `NetSettingEnum` constants. See Table 54.2.
-> `valueP`	New value for the setting.
-> `valueLen`	Size of new setting.

Result Returns one of the following values:

0	Success.
`netErrUnknownSetting`	Invalid setting constant.
`netErrInvalidSettingSize`	`valueLen` was invalid for the given setting.
`netErrBufTooSmall`	`valueP` was too small to hold entire setting. Setting value was truncated to fit in `valueP`.
`netErrBufWrongSize`	
`netErrReadOnlySetting`	

Sockets Equivalent None

Comments This call can be used to set the current value of any general setting. The caller must pass a pointer to a buffer which holds the new value (`valueP`), the size of the buffer (`valueLen`), and the setting ID (`setting`). The setting ID is one of the `netSettingXXX` constants in the `NetSettingEnum` type.

See `NetLibSettingGet` for an explanation of each of the settings.

Of particular interest is the `netSettingResetAll` setting, which, if used, will reset all general settings to their default values. When using this setting, `valueP` and `valueLen` are ignored.

See Also `NetLibSettingGet`, `NetLibSettingSet`, `NetLibIFSettingSet`, `NetLibMaster`

NetLibSocketAccept

Purpose Accept a connection from a stream-based socket.

Prototype

```
Int16 NetLibSocketAccept (UInt16 libRefnum,
NetSocketRef socket, NetSocketAddrType* sockAddrP,
Int16* addrLenP, Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
<- `sockAddrP`	Address of remote host is returned here.
<->`addrLenP`	On entry, length of `sockAddrP` buffer in bytes. On exit, length of returned address stored in `*sockAddrP`.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns the `NetSocketRef` of the new socket. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	

`netErrSocketNotConnected`

`netErrSocketClosedByRemote`

`netErrWrongSocketType`

`netErrSocketNotListening`

`netErrUnimplemented`

Sockets Equivalent

```
int accept (int socket, void* sockAddrP,
int* addrLenP);
```

Comments

Accepts the next connection request from a remote client. This call is only applicable to stream-based sockets. Before calling `NetLibSocketAccept` on a socket, a server application needs to:

- Open the socket (`NetLibSocketOpen`).
- Bind the socket to a local address (`NetLibSocketBind`).
- Set the maximum pending connection-request queue length (`NetLibSocketListen`).

`NetLibSocketAccept` will block until a successful connection request is obtained from a remote client. After a successful connection is made, this call returns with the address of the remote host in `*sockAddrP` and the socket descriptor of a **new** socket as the return value. You then use the new socket to send and receive data.

See Also

`NetLibSocketBind`, `NetLibSocketListen`

NetLibSocketAddr

Purpose Returns the local and remote addresses currently associated with a socket.

Prototype

```
Int16 NetLibSocketAddr (UInt16 libRefnum,
NetSocketRef socketRef,
NetSocketAddrType* locAddrP, Int16* locAddrLenP,
NetSocketAddrType* remAddrP, Int16* remAddrLenP,
Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socketRef`	Descriptor for the open socket.
<- `locAddrP`	Local address of socket is returned here.
<->`locAddrLenP`	On entry, length of `locAddrP` buffer in bytes. On exit, length of returned address stored in `*locAddrP`.
<- `remAddrP`	Address of remote host is returned here.
<->`remAddrLenP`	On entry, length of `remAddrP` buffer in bytes. On exit, length of returned address stored in `*remAddrP`.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns 0 upon success and -1 if an error occurred. If the return value is -1, `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrSocketClosedByRemote`	

netErrOutOfCmdBlocks

Sockets Equivalent

```
int getpeername (int s, struct sockaddr* name,
int* namelen);
```

```
int getsockname (int s, struct sockaddr* name,
int* namelen);
```

Comments

This call is mainly useful for stream-based sockets. It allows the caller to find out what address was bound to a connected socket and the address of the remote host that it's connected to.

In Palm OS version 3.0 and higher, if you pass a raw socket to this function, it returns the instance number and creator of the interface to which the socket is bound.

See Also

NetLibSocketBind, NetLibSocketConnect, NetLibSocketAccept

NetLibSocketBind

Purpose

Assign a local address to a socket.

Prototype

```
Int16 NetLibSocketBind (UInt16 libRefnum,
NetSocketRef socket, NetSocketAddrType* sockAddrP,
Int16 addrLen, Int32 timeout, Err* errP)
```

Parameters

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
-> sockAddrP	Pointer to the address to give to the socket. This can be a NetSocketAddrINType or a NetSocketAddrRawType.
-> addrLen	Length of address in *sockAddrP.
-> timeout	Maximum timeout in system ticks; -1 means wait forever.

<- `errP`	Contains an error code if the return value is -1.

Result

Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrSocketAlreadyConnected`	
`netErrSocketClosedByRemote`	
`netErrOutOfCmdBlocks`	

Sockets Equivalent

```
int bind (int socket, const void* sockAddrP,
int addrLen);
```

Comments

Applications that want to wait for an incoming connection request from a remote host must call this function. After calling `NetLibSocketBind`, applications can call `NetLibSocketListen` and then `NetLibSocketAccept` to make the socket ready to accept connection requests.

Compatibility

Raw sockets are only supported in Palm OS version 3.0 and higher. See `NetLibSocketOpen` for instructions on how to bind raw sockets.

See Also

`NetLibSocketConnect`, `NetLibSocketListen`, `NetLibSocketAccept`

NetLibSocketClose

Purpose Close a socket.

Prototype

```
Int16 NetLibSocketClose (UInt16 libRefnum,
NetSocketRef socket, Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrOutOfCmdBlocks`	

Sockets Equivalent

```
int close(int socket);
```

Comments Closes down a socket and frees all memory associated with it.

See Also `NetLibSocketOpen`, `NetLibSocketShutdown`

NetLibSocketConnect

Purpose Assign a destination address to a socket and initiate three-way handshake if it's stream based.

Prototype

```
Int16 NetLibSocketConnect (UInt16 libRefnum,
NetSocketRef socket, NetSocketAddrType* sockAddrP,
Int16 addrLen, Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
-> `sockAddrP`	Pointer to address to connect to.
-> `addrLen`	Length of address in `*sockAddrP`.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrSocketBusy`	
`netErrNoInterfaces`	Incorrect setup.
`netErrPortInUse`	
`netErrQuietTimeNotElapsed`	
`netErrInternal`	
`netErrSocketAlreadyConnected`	

`netErrSocketClosedByRemote`

`netErrTooManyTCPConnections`

`netErrWouldBlock`

`netErrWrongSocketType`

`netErrOutOfCmdBlocks`

Sockets Equivalent

```
int connect (int socket, const void* sockAddrP,
int addrLen);
```

See Also NetLibSocketBind, NetUTCPOpen

NetLibSocketListen

Purpose Put a stream-based socket into passive listen mode.

Prototype

```
Int16 NetLibSocketListen (UInt16 libRefnum,
NetSocketRef socket, UInt16 queueLen,
Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
-> `queueLen`	Maximum number of pending connections allowed.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	

`netErrOutOfResources`

`netErrSocketNotOpen`

`netErrSocketBusy`

`netErrNoInterfaces`
Incorrect setup.

`netErrPortInUse`

`netErrInternal`

`netErrSocketAlreadyConnected`

`netErrSocketClosedByRemote`

`netErrWrongSocketType`

`netErrQuietTimeNotElapsed`

`netErrOutOfCmdBlocks`

Sockets Equivalent

```
int listen (int socket, int queueLen);
```

Comments

Sets the maximum allowable length of the queue for pending connections. This call is only applicable to stream-based (TCP/IP) sockets.

After a socket is created and bound to a local address using `NetLibSocketBind`, a server application can call `NetLibSocketListen` and then `NetLibSocketAccept` to accept connections from remote clients.

The `queueLen` is currently quietly limited to 1 (higher values are ignored).

See Also

`NetLibSocketBind`, `NetLibSocketAccept`

NetLibSocketOpen

Purpose Open a new socket.

Prototype

```
NetSocketRef NetLibSocketOpen (UInt16 libRefnum,
NetSocketAddrEnum domain, NetSocketTypeEnum type,
Int16 protocol, Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `domain`	Address domain. See `NetSocketAddrEnum`.
-> `type`	Desired type of connection. See `NetSocketTypeEnum`.

-> `protocol`	Protocol to use. This parameter is currently ignored.
	For raw sockets in the `netSocketAddrINET` domain, specify one of the following:
	`netSocketProtoIPTCP`
	`netSocketProtoIPUDP`
	`netSocketProtoIPRAW`
	For all other socket types or for raw sockets in the raw domain, this parameter is ignored.

-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns the `NetSocketRef` of the opened socket or -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	

`netErrNoMoreSockets`

`netErrOutOfCmdBlocks`

`netErrOutOfMemory`

Sockets Equivalent

```
int socket(int domain, int type, int protocol);
```

Comments

Allocates memory for a new socket and opens it.

Raw sockets are supported in Palm OS version 3.0 and higher. Two types of raw sockets are supported:

- Raw sockets in the `netSocketAddrINET` domain

  In this case, you must bind the socket to an IP address using `NetLibSocketBind`, passing a `NetSocketAddrINType` structure for the socket address. The port field is ignored.

  For applications that use raw sockets in the INET domain, the net library checks the destination IP address of all incoming packets to see if it matches any of those raw sockets. If it does, the packet is enqueued directly into the matching socket and is **not** passed to the protocol stack.

  When an application sends data through raw sockets in the IP domain, the net library packages the data into a packet and passes it directly to the interface's send routine. You are responsible for forming the entire IP header, including any necessary checksums, source and destination IP address, and so on.

- Raw sockets in the `netSocketAddrRaw` domain with no protocol

  In this case, you must bind the socket to an interface using `NetLibSocketBind`, passing a `NetSocketAddrRawType` structure for the socket address. The instance and creator specify which interface the caller wants to receive raw packets from.

  When an interface is bound to a raw socket with no protocol, the net library places that interface into raw mode. In raw

mode, the interface passes all incoming packets, no matter what the link layer protocol, to its raw receive function.

When an application sends data through a raw socket with no protocol, the net library packages the data into a packet and passes it directly to the interface's send routine.

The interface remains in raw mode until the raw socket is closed.

Compatibility Raw sockets supported only in Palm OS version 3.0 and higher.

See Also NetLibSocketClose, NetUTCPOpen

NetLibSocketOptionGet

Purpose Retrieves the current value of a socket option.

Prototype

```
Int16 NetLibSocketOptionGet (UInt16 libRefnum,
NetSocketRef socket, UInt16 level, UInt16 option,
void* optValueP, UInt16 * optValueLenP,
Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
-> `level`	Level of the option, one of the `NetSocketOptLevelEnum` constants. See NetLibSocketOptionSet.
-> `option`	One of the `NetSocketOptEnum` constants. See NetLibSocketOptionSet.
<- `optValueP`	Pointer to variable holding new value of option.
<-> `optValueLenP`	Size of variable pointed to by `optValueP` on entry. Actual size of return value on exit.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.

<- `errP`	Contains an error code if the return value is -1.

Result

Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrUnimplemented`	
`netErrWrongSocketType`	
`netErrInvalidSettingSize`	

Sockets Equivalent

```
int getsockopt (int socket, int level, int option,
const void* optValueP, int* optValueLenP);
```

Comments

Returns the current value of a socket option. The caller passes a pointer to a variable to hold the returned value (in `optValueP`) and the size of this variable (in `*optValueLenP`). On exit, `*optValueP` is updated with the actual size of the return value.

For all of the fixed size options (every option except `netSocketOptIPOptions`), `*optValueLenP` is unmodified on exit and this call does its best to return the value in the caller's desired type size.

For compatibility with existing Internet applications, this call is quite flexible on the `*optValueLenP` parameter. If the desired type for an option is `FLAG`, this call supports an `*optValueLenP` of 1, 2, or 4. If the desired type for an option is `int`, it supports an `*optValueLenP` of 2 or 4.

See `NetLibSocketOptionSet` for a list of available options.

See Also

`NetLibSocketOptionSet`

NetLibSocketOptionSet

Purpose Set a socket option.

Prototype

```
Int16 NetLibSocketOptionSet (UInt16 libRefnum,
NetSocketRef socket, UInt16 level, UInt16 option,
void* optValueP, UInt16 optValueLen,
Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.
-> `level`	Level of the option, one of the `NetSocketOptLevelEnum` constants. See the comments section.
-> `option`	One of the `NetSocketOptEnum` constants. See the comments section.
-> `optValueP`	Pointer to the variable holding the new value of the option.
-> `optValueLen`	Size of variable pointed to by `optValueP`.
-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrUnimplemented`	
`netErrWrongSocketType`	

```
netErrInvalidSettingSize
```

Sockets Equivalent

```
int setsockopt (int socketRef, int level,
int option, const void* optValueP,
int optValueLen);
```

Comments

Sets various options associated with a socket. The caller passes a pointer to the new option value in `optValueP` and the size of the option in `optValueLen`.

Table 54.3 lists the available options.

- The Level column specifies the option level, which is one of the `netSocketOptLevelXXX` constants.
- The Option column lists the option, which is one of the `netSocketOptXXX` constants.
- The G/S column lists whether this option can be fetched with the `NetLibSocketOptionGet` call (G) and/or set (S) with this call.
- The type column lists the data type of the option.
- The I column specifies whether or not this option is currently implemented.

Table 54.3 Net Library Socket Options

netSocket OptLevel...	netSocketOpt...	G/S	Type	I	Description
IP	IPOptions	GS	UInt8[]	N	Options in IP Header
TCP	TCPNoDelay	GS	FLAG	Y	Don't delay send to coalesce packets
TCP	TCPMaxSeg	G	int	Y	Get TCP maximum segment size
Socket	SockDebug	GS	FLAG	N	Turn on recording of debug info
Socket	SockAcceptConn	G	FLAG	N	Socket has had listen
Socket	SockReuseAddr	GS	FLAG	N	Allow local address reuse

Table 54.3 Net Library Socket Options *(continued)*

netSocket OptLevel...	netSocketOpt...	G/S	Type	I	Description
Socket	SockKeepAlive	GS	FLAG	Y	Keep connections alive
Socket	SockDontRoute	GS	FLAG	N	Just use interface addresses
Socket	SockBroadcast	GS	FLAG	N	Permit sending of broadcast messages
Socket	SockUseLoopback	GS	FLAG	N	Bypass hardware when possible
Socket	SockLinger	GS	NetSocket LingerType	Y	Linger on close if data present `NetSocketLingerType` is a structure with two fields: `onOff` (`true` or `false`) and `time` (linger time in seconds).
Socket	SockOOBInLine	GS	FLAG	N	Leave received OOB data in-line
Socket	SockSndBufSize	GS	int	N	Send buffer size
Socket	SockRcvBufSize	GS	int	N	Receive buffer size
Socket	SockSndLowWater	GS	int	N	Send low-water mark
Socket	SockRcvLowWater	GS	int	N	Receive low-water mark
Socket	SockSndTimeout	GS	int	N	Send timeout
Socket	SockRcvTimeout	GS	int	N	Receive timeout
Socket	SockErrorStatus	G	int	Y	Get error status and clear
Socket	SockSocketType	G	int	Y	Get socket type
Socket	SockNonBlocking	GS	FLAG	Y	Set non-blocking mode on/off

For compatibility with existing Internet applications, this call is quite flexible on the `optValueLen` parameter. If the desired type for an option is `FLAG`, this call accepts an `optValueLen` of 1, 2, or 4. If the desired type for an option is `int`, it accepts an `optValueLen` of 2 or 4.

Except for the `netSocketOptSockNonBlocking` option, all options listed above have equivalents in the sockets API. The `netSocketOptSockNonBlocking` option was added to this call in the net library in order to implement the functionality of the UNIX `fcntl()` control call, which can be used to turn nonblocking mode on and off for sockets.

See Also `NetLibSocketOptionGet`

NetLibSocketShutdown

Purpose Shut down a socket in one or both directions.

Prototype

```
Int16 NetLibSocketShutdown (UInt16 libRefnum,
NetSocketRef socket, Int16 direction,
Int32 timeout, Err* errP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `socket`	Descriptor for the open socket.

-> `direction`	Direction to shut down. One of the `NetSocketDirEnum` constants. Specifically:
	`netSocketDirInput`
	`netSocketDirOutput`
	`netSocketDirBoth`

-> `timeout`	Maximum timeout in system ticks; -1 means wait forever.
<- `errP`	Contains an error code if the return value is -1.

Result Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.

`netErrTimeout`	Call timed out.
`netErrNotOpen`	The referenced net library has not been opened yet.
`netErrParamErr`	
`netErrSocketNotOpen`	
`netErrNoMultiPktAddr`	
`netErrOutOfCmdBlocks`	

Sockets Equivalent

```
int shutdown (int socket, int direction);
```

Comments

Shuts down communication in one or both directions on a socket.

If direction is `netSocketDirInput`, the socket is marked as down in the receive direction and further read operations from it return a `netErrSocketInputShutdown` error.

NetLibTracePrintF

Purpose

Store debugging information in the net library's trace buffer.

Prototype

```
Err NetLibTracePrintF (UInt16 libRefnum,
Char *formatStr, ...)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `formatStr`	A `printf` style format string.
-> ...	Arguments to the format string.

Result

Returns 0 upon success or `netErrNotOpen` if the net library has not been opened.

Sockets Equivalent

None

Comments

This call is a convenient debugging tool for developing Internet applications. It stores a message into the net library's trace buffer, which can later be dumped using the `NetLibMaster` call. The net

library's trace buffer is used to store run-time errors that the net library encounters as well as errors and messages from network interfaces and from applications that use this call.

The `formatStr` parameter is a `printf` style format string which supports the following format specifiers:

%d, %i, %u, %x, %s, %c

but it does **not** support field widths, leading 0's etc.

Note that the `netTracingAppMsgs` bit of the `netSettingTraceBits` setting must be set using the call `NetLibSettingSet(...netSettingTraceBits...)`. Otherwise, this routine will do nothing.

See Also `NetLibTracePutS`, `NetLibMaster`, `NetLibSettingSet`

NetLibTracePutS

Purpose Store debugging information in the net library's trace buffer.

Prototype

```
Err NetLibTracePutS (UInt16 libRefnum, Char *strP)
```

Parameters

-> `libRefNum`	Reference number of the net library.
-> `strP`	String to store in the trace buffer.

Result Returns 0 upon success or `netErrNotOpen` if the net library has not been opened.

Sockets Equivalent None

Comments This call is a convenient debugging tool for developing Internet applications. It will store a message into the net library's trace buffer which can later be dumped using the `NetLibMaster` call. The net library's trace buffer is used to store run-time errors that the net library encounters as well as errors and messages from network interfaces and from applications that use this call.

Note the `netTracingAppMsgs` bit of the `netSettingTraceBits` setting must be set using the `NetLibSettingSet(...netSettingTraceBits...)` call or this routine will do nothing.

See Also `NetLibTracePrintF`, `NetLibMaster`, `NetLibSettingSet`.

NetNToHL

Purpose Macro that converts a 32-bit value from network to host byte order.

Prototype

```
NetNToHL (x)
```

Parameters

-> `x`	32-bit value to convert.

Result Returns `x` in host byte order.

Errors none

Sockets Equivalent `ntohl()`

See Also `NetNToHS`, `NetHToNL`, `NetHToNS`

NetNToHS

Purpose Macro that converts a 16-bit value from network to host byte order.

Prototype `NetNToHS (x)`

Parameters -> `x` 16-bit value to convert.

Result Returns `x` in host byte order.

Errors None

Sockets Equivalent `ntohs()`

See Also `NetHToNL`, `NetNToHL`, `NetHToNS`

55

Network Utilities

This chapter describes network utilities provided in the module `NetSocket.c`. These utilities are convenience functions that you can use in place of net library functions in applications that use the net library. You can find `NetSocket.c` in the folder `Libraries\Net\Src`. (On Palm OS® 3.5, `NetSocket.c` is in the folder `CodeWarrior Libraries\Comms\NetSocket\Src`.)

The include file for the functions described in this chapter is `<unix/sys_socket.h>`. This header file is not included by any other Palm header file; you must explicitly include it in your code.

For more information on `NetSocket.c` and `sys_socket.h`, see the chapter "Network Communication" in the *Palm OS Programmer's Companion*.

Network Utility Functions

NetUReadN

Purpose Reads a specified number of bytes from a socket.

Prototype

```
Int32 NetUReadN (NetSocketRef fd, UInt8* bufP,
UInt32 numBytes)
```

Parameters

`-> fd`	Descriptor for the open socket.
`<- bufP`	Pointer to buffer to hold received data.
`-> numBytes`	Number of bytes to read.

Result Returns the number of bytes actually read. If the return value is less than 0, an error occurred.

Comments This function repeatedly calls NetLibReceive until numBytes have been read or until NetLibReceive returns an error.

See Also NetUWriteN

NetUTCPOpen

Purpose Opens a TCP (streams-based) socket and connects it to a server.

Prototype

```
NetSocketRef NetUTCPOpen (Char* hostName,
Char* serviceName, Int16 port)
```

Parameters

-> hostName	Remote host, given either by name or by dotted decimal address.
-> serviceName	The name of a network service or NULL if the port parameter is used. Possible services are "echo", "discard", "daytime", "qotd", "chargen", "ftp-data", "ftp", "telnet", "smtp", "time", "name", "finger", "pop2", "pop3", "nntp", "imap2".
-> port	The number of the port to connect to on the remote host. Ignored if serviceName is not NULL.

Result Returns the socket descriptor of the socket that was connected, or -1 if an error occurred.

Comments If serviceName is given, this function looks up the port number for that service on the remote host and uses it for the connection.

This function is the equivalent of calling NetLibSocketOpen and NetLibSocketConnect (or socket and connect).

NOTE: This function does not return specific reasons for failure if there is a failure. This function is not production-quality code. It is provided as a quick and dirty way of creating a connection and as sample code that can be used as a reference.

NetUWriteN

Purpose Writes the specified number of bytes to a socket.

Prototype

```
Int32 NetUWriteN (NetSocketRef fd, UInt8* bufP,
UInt32 numBytes)
```

Parameters

`-> fd`	Descriptor for the open socket.
`-> bufP`	Pointer to buffer to write.
`-> numBytes`	Number of bytes to write.

Result Returns the number of bytes actually sent. If the return value is less than 0, an error occurred.

Comments This function repeatedly calls `NetLibSend` until `numBytes` have been written or until `NetLibSend` returns an error.

See Also `NetUReadN`

56

New Serial Manager

This chapter provides reference material for the new serial manager API:

- New Serial Manager Data Structures
- New Serial Manager Constants
- New Serial Manager Functions
- New Serial Manager Application-Defined Function

The header file `SerialMgr.h` declares the serial manager API. For more information on the new serial manager, see the chapter "Serial Communication" in the *Palm OS Programmer's Companion*.

New Serial Manager Data Structures

DeviceInfoType

The `DeviceInfoType` structure defines information about a serial device. This structure is returned by the `SrmGetDeviceInfo` function.

```
typedef struct DeviceInfoType {
UInt32 serDevCreator;
UInt32 serDevFtrInfo;
UInt32 serDevMaxBaudRate;
UInt32 serDevHandshakeBaud;
Char *serDevPortInfoStr;
UInt8 reserved[8]; // Reserved
} DeviceInfoType;
typedef DeviceInfoType *DeviceInfoPtr;
```

Value Descriptions

`serDevCreator`	Four-character creator type for serial driver ('sdrv').
`serDevFtrInfo`	Flags defining features of this serial hardware. Specify one of the flags described in Serial Capabilities Constants.
`serDevMaxBaudRate`	Maximum baud rate for this device.
`serDevHandshakeBaud`	Hardware handshaking is recommended for baud rates over this rate.
`serDevPortInfoStr`	Description of serial hardware device or virtual device.

SrmCtlEnum

The `SrmCtlEnum` enumerated type specifies a serial control operation. Specify one of these enumerated types for the `op` parameter to the `SrmControl` call.

```
typedef enum SrmCtlEnum {
srmCtlFirstReserved = 0, // RESERVE 0
srmCtlSetBaudRate,
srmCtlGetBaudRate,
srmCtlSetFlags,
srmCtlGetFlags,
srmCtlSetCtsTimeout,
srmCtlGetCtsTimeout,
srmCtlStartBreak,
srmCtlStopBreak,
srmCtlStartLocalLoopback,
srmCtlStopLocalLoopback,
srmCtlIrDAEnable,
srmCtlIrDADisable,
srmCtlRxEnable,
srmCtlRxDisable,
srmCtlEmuSetBlockingHook,
srmCtlUserDef,
```

```
srmCtlGetOptimalTransmitSize,
srmCtlLAST
} SrmCtlEnum;
```

Value Descriptions

`srmCtlSetBaudRate`	Sets the current baud rate for the serial hardware.
`srmCtlGetBaudRate`	Gets the current baud rate for the serial hardware.
`srmCtlSetFlags`	Sets the current flag settings for the serial hardware. Specify flags from the set described in Serial Settings Constants.
`srmCtlGetFlags`	Gets the current flag settings for the serial hardware.
`srmCtlSetCtsTimeout`	Sets the current CTS timeout value for hardware handshaking.
`srmCtlGetCtsTimeout`	Gets the current CTS timeout value for hardware handshaking.
`srmCtlStartBreak`	Turn RS232 break signal on. Caller is responsible for turning this signal on and off and insuring it is on long enough to generate a viable break.
`srmCtlStopBreak`	Turn RS232 break signal off.
`srmCtlStartLocalLoopback`	Start local loopback test.
`srmCtlStopLocalLoopback`	Stop local loopback test.
`srmCtlIrDAEnable`	Enable IrDA connection on this serial port.
`srmCtlIrDADisable`	Disable IrDA connection on this serial port.
`srmCtlRxEnable`	Enable receiver (for IrDA).
`srmCtlRxDisable`	Disable receiver (for IrDA).
`srmCtlEmuSetBlockingHook`	Set a blocking hook routine for emulation mode only. Not supported on the Palm device.

`srmCtlUserDef`	This is a user-defined function that 3rd party hardware developers can use to set or retrieve hardware-specific information from the serial driver. This opCode invokes the `SdrvControl` (or `VdrvControl`) function with its user-defined opCode and the parameters are passed directly through to the serial driver. A serial driver that does not handle this function returns a `serErrBadParam` error.
`srmCtlGetOptimalTransmitSize`	Ask the port for the most efficient buffer size for transmitting data packets. This opCode returns an error (buffering not necessary), 0 (buffering requested, but application can choose buffer size), or a number > 0 (recommended buffer size).

SrmCallbackEntryType

The `SrmCallbackEntryType` structure defines a callback function for the `SrmControl` function's `srmCtlEmuSetBlockingHook` opCode.

```
typedef struct SrmCallbackEntryType {
BlockingHookProcPtr funcP;
UInt32 userRef; // ref value to pass to callback
} SrmCallbackEntryType;
```

Value Descriptions

`funcP`	Function pointer to the callback function. Pass `NULL` if you no longer want a callback function to be called.
`userRef`	User-defined reference value passed to the callback function.

New Serial Manager Constants

Serial Capabilities Constants

These constants describe serial hardware capabilities.

`serDevCradlePort`	Serial hardware controls RS-232 serial from cradle connector of Palm device.
`serDevRS232Serial`	Serial hardware has RS-232 line drivers
`serDevIRDACapable`	Serial hardware has IR line drivers and generates IrDA mode serial signals
`serDevModemPort`	Serial hardware drives modem connection
`serDevCncMgrVisible`	Serial device port name string is to be displayed in the Connection panel.

Serial Settings Constants

These constants identify bit flags that correspond to various serial hardware settings.

`srmSettingsFlagStopBitsM`	mask for stop bits field
`srmSettingsFlagStopBits1`	1 stop bit
`srmSettingsFlagStopBits2`	2 stop bits
`srmSettingsFlagParityOnM`	mask for parity on
`srmSettingsFlagParityEvenM`	mask for parity even
`srmSettingsFlagXonXoffM`	mask for Xon/Xoff flow control (not implemented)
`srmSettingsFlagRTSAutoM`	mask for RTS receive flow control
`srmSettingsFlagCTSAutoM`	mask for CTS transmit flow control

`srmSettingsFlagBitsPerCharM`	mask for bits per character
`srmSettingsFlagBitsPerChar5`	5 bits per character
`srmSettingsFlagBitsPerChar6`	6 bits per character
`srmSettingsFlagBitsPerChar7`	7 bits per character
`srmSettingsFlagBitsPerChar8`	8 bits per character
`srmSettingsFlagFlowControl`	Protect the receive buffer from software overruns. When this flag, and `srmSettingsFlagRTSAutoM` are set, it causes the new serial manager to assert RTS to prevent the transmitting device from continuing to send data when the receive buffer is full. Once the application receives data from the buffer, RTS is deasserted to allow data reception to resume. Note that this feature effectively prevents software overrun line errors but may also cause CTS timeouts on the transmitting device if the RTS line is asserted longer than the defined CTS timeout value.

Status Constants

These constants identify bit flags that correspond to the status of serial signals. They can be returned by the `SrmGetStatus` function.

`srmStatusCtsOn`	CTS line is active.
`srmStatusRtsOn`	RTS line is active.
`srmStatusDsrOn`	DSR line is active.
`srmStatusBreakSigOn`	Break signal is active.

New Serial Manager Functions

SrmClearErr

Purpose Clears the port of any line errors.

Prototype

```
Err SrmClearErr(UInt16 portId)
```

Parameters

-> `portID`	Port ID.

Result

0	No error.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SrmClose

Purpose Closes a serial port and makes it available to other applications, regardless of whether the port is a foreground or background port.

Prototype

```
Err SrmClose(UInt16 portID)
```

Parameters

-> `portID`	Port ID for port to be closed.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Comments If a foreground port is being closed and a background port exists, the background will have access to the port as long as another foreground port is not opened (via `SrmOpen`).

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmOpen`, `SrmOpenBackground`

SrmControl

Purpose Performs a serial control function.

Prototype

```
Err SrmControl(UInt16 portId, UInt16 op,
void *valueP, UInt16 *valueLenP)
```

Parameters

-> `portID`	Port ID.
-> `op`	Control operation to perform. Specify one of the `SrmCtlEnum` enumerated types.
<-> `valueP`	Pointer to a value to use for the operation. See Comments for details.
<-> `valueLenP`	Pointer to the size of `*valueP`. See Comments for details.

Comments Table 56.1 shows what to pass for the `valueP` and `valueLenP` parameters for each of the operation codes. Control codes not listed do not use these parameters.

Table 56.1 SrmControl Parameters

Operation Code	Parameters
`srmCtlSetBaudRate`	-> `valueP` = Pointer to `Int32` (baud rate) -> `valueLenP` = Pointer to `sizeof(Int32)`
`srmCtlGetBaud`	<- `valueP` = Pointer to `Int32` (baud rate) <- `valueLenP` = Pointer to `Int16`
`srmCtlSetFlags`	-> `valueP` = Pointer to `Uint32` (bitfield; see Serial Settings Constants) -> `valueLenP` = Pointer to `sizeof(UInt32)`
`srmCtlGetFlags`	<- `valueP` = Pointer to `UInt32` (bitfield) <- `valueLenP` = Pointer to `Int16`
`srmCtlSetCtsTimeout`	-> `valueP` = Pointer to `Int32` (timeout value) -> `valueLenP` = Pointer to `sizeof(Int32)`
`srmCtlGetCtsTimeout`	<- `valueP` = Pointer to `Int32` (timeout value) -> `valueLenP` = Pointer to `Int16`

Table 56.1 SrmControl Parameters *(continued)*

Operation Code	Parameters
`srmCtlEmuSetBlockingHook`	<-> `valueP` = Pointer to `SrmCallbackEntryType` struct <-> `valueLenP` = Pointer to `sizeof(SrmCallbackEntryType)` Returns the old settings in the first parameter.
`srmCtlUserDef`	<-> `valueP` = Pointer passed to the serial or virtual driver <-> `valueLenP` = Pointer to `sizeof(Int32)` For a serial driver, these pointers are passed to the `SdrvControl` function's `sdrvOpCodeUserDef` opCode. For a virtual driver, these pointers are passed to the `VdrvControl` function's `vdrvOpCodeUserDef` opCode.
`srmCtlGetOptimalTransmitSize`	<- `valueP` = Pointer to Int32 <- `valueLenP` = Pointer to `sizeof(Int32)` If an error is returned by `SrmControl`, no buffering should be done. If `valueP` points to zero, buffering is requested, but the transmitting application can determine the buffer size. If `valueP` points to a number > 0, then try to send data in blocks of this number of bytes, as this is the most efficient block size for this particular device.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SrmGetDeviceCount

Purpose Returns the number of available serial devices.

Prototype

```
Err SrmGetDeviceCount(UInt16* numOfDevicesP)
```

Parameters

<- `numOfDevicesP`	Pointer to address where the number of serial devices is returned.

Result

0	No error.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmGetDeviceInfo`

SrmGetDeviceInfo

Purpose Returns information about a serial device.

Prototype

```
Err SrmGetDeviceInfo(UInt32 deviceID,
DeviceInfoType* deviceInfoP)
```

Parameters

-> `deviceID`	ID of serial device to get information for. You can pass a zero-based index (0, 1, 2, ...), a valid port ID returned from `SrmOpen`, or a 4-character port name (such as 'u328', 'u650', or 'ircm').
<- `deviceInfoP`	Pointer to a `DeviceInfoType` structure where information about the device is returned.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmGetDeviceCount`

SrmGetStatus

Purpose Returns status information about the serial hardware.

Prototype

```
Err SrmGetStatus(UInt16 portId,
UInt32* statusFieldP), UInt16* lineErrsP)
```

Parameters

-> `portId`	Port ID.
<- `statusFieldP`	Pointer to address where hardware status information for the port is returned. This is a 32-bit field using the flags described in Status Constants.
<- `lineErrsP`	Pointer to address where the number of line errors for the port is returned.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Comments Typically, `SrmGetStatus` is called to retrieve the line errors for the port if some of the send and receive functions return a `serErrLineErr` error code.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SrmOpen

Purpose Opens a foreground port connection with the specified port name or logical port number.

Prototype

```
Err SrmOpen(UInt32 port, UInt32 baud,
UInt16* newPortIdP)
```

Parameters

-> `port`	Port name or logical port number to be opened. For information about how to identify a port, see "Specifying the portID Parameter" on page 236 in the *Palm OS Programmer's Companion*.
-> `baud`	Initial baud rate of port.
<- `newPortIdP`	Pointer to address where the port ID to be used with other new serial manager functions is returned.

Result

0	No error.
`serErrAlreadyOpen`	This port already has an installed foreground owner.
`serErrBadPort`	This port doesn't exist.
`memErrNotEnoughSpace`	There was not enough memory available to open the port.

Comments Only one application or task may have access to a particular serial port at any time.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmOpenBackground`

SrmOpenBackground

Purpose

Allows a task to open, initialize, and use the port, but always relinquishes control of the port when another task opens the port with the `SrmOpen` call.

Prototype

```
Err SrmOpenBackground(UInt32 port, UInt32 baud,
UInt16* newPortIdP)
```

Parameters

-> `port`	Physical or logical port number to be opened.
-> `baud`	Initial baud rate of port.
<- `newPortIdP`	Pointer to address where the port ID to be used with other new serial manager functions is returned.

Result

0	No error.
`serErrAlreadyOpen`	This port already has an installed background owner.
`serErrBadPort`	This port doesn't exist.
`memErrNotEnoughSpace`	There was not enough memory available to open the port.

Comments

This function is provided to support tasks that want to use a serial device to receive data only when no other task is using the port.

If a background port is forced to surrender control of the hardware as a result of another task opening a foreground connection, all buffers for the background port are flushed. After this active task closes the port, active control of the port is returned to the background task. Only one task can have background ownership of the port.

Note that background ports have limited functionality: they can only receive data and notify owning clients of what data has been received.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmOpen`

SrmPrimeWakeupHandler

Purpose Sets the number of received bytes that triggers a call to the wakeup handler function.

Prototype

```
Err SrmPrimeWakeupHandler(UInt16 portId,
UInt16 minBytes)
```

Parameters

-> `portId`	Port ID.
-> `minBytes`	Number of bytes that must be received before wakeup handler is called. Typically, this is set to 1.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Comments This function primes a wakeup handler installed by `SrmSetWakeupHandler`.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmSetWakeupHandler`, `WakeupHandlerProc`

SrmReceive

Purpose Receives a specified number of bytes.

Prototype

```
UInt32 SrmReceive(UInt16 portId, void *rcvBufP,
UInt32 count, Int32 timeout, Err* errP)
```

Parameters

-> `PortId`	Port ID.

<- `rcvBufP`	Pointer to buffer where received data is to be returned.
-> `count`	Length of data buffer (in bytes). This specifies the number of bytes to receive.
-> `timeout`	The amount of time (in ticks) that the new serial manager waits to receive the requested block of data. At the end of the timeout, data received up to that time is returned.
<- `errP`	Error code.

Result Number of bytes of data actually received.

Comments The following error codes can be returned in `*errP`:

0	No error.
`serErrBadPort`	This port doesn't exist.
`serErrTimeoutErr`	Unable to receive data within the specified `ctsTimeout` period.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmReceiveCheck`, `SrmReceiveFlush`, `SrmReceiveWait`

SrmReceiveCheck

Purpose Checks the receive FIFO and returns the number of bytes in the serial receive queue.

Prototype

```
Err SrmReceiveCheck(UInt16 portId,
UInt32* numBytesP)
```

Parameters

-> `portId`	Port ID.

<- `numBytesP`	Number of bytes in the receive queue.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmReceive`, `SrmReceiveFlush`, `SrmReceiveWait`

SrmReceiveFlush

Purpose Flushes the receive FIFOs.

Prototype

```
Err SrmReceiveFlush(UInt16 portId, Int32 timeout)
```

Parameters

-> `portId`	Port ID.
-> `timeout`	Timeout value, in ticks.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Comments The `timeout` value forces this function to wait a period of ticks after flushing the port to see if more data shows up to be flushed. If more data arrives within the timeout period, the port is flushed again and the timeout counter is reset and waits again. The function only exits after no more bytes are received by the port for the full timeout period since the last flush of the port. To avoid this waiting behavior, specify 0 for the timeout period.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmReceive`, `SrmReceiveCheck`, `SrmReceiveWait`

SrmReceiveWait

Purpose Waits until some number of bytes of data have arrived into the serial receive queue, then returns.

Prototype

```
Err SrmReceiveWait(UInt16 portId, UInt32 bytes,
Int32 timeout)
```

Parameters

-> `portId`	Port ID.
-> `bytes`	Number of bytes to wait for.
-> `timeout`	Timeout value, in ticks.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.
`serErrTimeoutErr`	Unable to receive data within the specified `timeout` period.

Comments If this function returns no error, the application can either check the number of bytes currently in the receive queue (using `SrmReceiveCheck`) or it can just specify a buffer and receive the data by calling `SrmReceive`.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmReceive`, `SrmReceiveCheck`, `SrmReceiveFlush`

SrmReceiveWindowClose

Purpose Closes direct access to the new serial manager's receive queue.

Prototype

```
Err SrmReceiveWindowClose(UInt16 portId,
UInt32 bytesPulled)
```

Parameters

-> `portId`	Port ID.

-> `bytesPulled`	Number of bytes the application read from the receive queue.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Comments Call this function when the application has read as many bytes as it needs out of the receive queue or it has read all the available bytes.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmReceiveWindowOpen`

SrmReceiveWindowOpen

Purpose Provides direct access to the new serial manager's receive queue.

Prototype

```
Err SrmReceiveWindowOpen(UInt16 portId,
UInt8 **bufPP, UInt32* sizeP)
```

Parameters

-> `portId`	Port ID.
<- `bufPP`	Pointer to a pointer to the receive buffer.
<- `sizeP`	Available bytes in buffer.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.
`serErrLineErr`	The data in the queue contains line errors.

Comments This function lets applications directly access the new serial manager's receive queue to eliminate buffer copying by the serial manager. This access is a "back door" route to the received data.

After retrieving data from the buffer, the application must call SrmReceiveWindowClose.

Applications that want to empty the receive buffer entirely should call the SrmReceiveWindowOpen and SrmReceiveWindowClose functions repeatedly until the buffer size returned is 0.

IMPORTANT: Once an application calls SrmReceiveWindowOpen, it should not attempt to receive data via the normal method of calling SrmReceive or SrmReceiveWait, as these functions interfere with direct access to the receive queue.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also SrmReceiveWindowClose

SrmSend

Purpose Sends a block of data out the specified port.

Prototype

```
UInt32 SrmSend(UInt16 portId, void *bufP,
UInt32 count, Err* errP)
```

Parameters

-> portId	Port ID.
-> bufp	Pointer to data to send.
-> count	Length of data buffer, in bytes.
<- errP	Error code. See Comments section for details.

Result Number of bytes of data actually sent.

Comments If *errP is NULL, the result value should be the same as the count parameter. If *errP is not NULL, then the result equals the number of bytes sent before the error occurred.

The following error codes can be returned in *errP:

0	No error.
`serErrBadPort`	This port doesn't exist.
`serErrTimeoutErr`	Unable to send data within the specified `ctsTimeout` period.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmSendCheck`, `SrmSendFlush`, `SrmSendWait`

SrmSendCheck

Purpose Checks the transmit FIFO and returns the number of bytes left to be sent.

Prototype

```
Err SrmSendCheck(UInt16 portId, UInt32* numBytesP)
```

Parameters

-> `portID`	Port ID.
<- `numBytesP`	Number of bytes left in the FIFO queue.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.
`serErrNotSupported`	This feature not supported by the hardware.

Comments Not all serial devices support this feature.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmSend`, `SrmSendFlush`, `SrmSendWait`

SrmSendFlush

Purpose Flushes the transmit FIFO.

Prototype

```
Err SrmSendFlush(UInt16 portId)
```

Parameters

-> `portId` Port ID.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmSend`, `SrmSendCheck`, `SrmSendWait`

SrmSendWait

Purpose Waits until all previous data has been sent from the transmit FIFO, then returns.

Prototype

```
Err SrmSendWait(UInt16 portId)
```

Parameters

-> `portId` Port ID.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.
`serErrTimeoutErr`	Unable to send data within the `ctsTimeout` period.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmSend`, `SrmSendCheck`, `SrmSendFlush`

SrmSetReceiveBuffer

Purpose Installs a new buffer into the new serial manager's receive queue.

Prototype

```
Err SrmSetReceiveBuffer(UInt16 portId, void *bufP,
UInt16 bufSize)
```

Parameters

-> `portId`	Port ID.
-> `bufP`	Pointer to new receive buffer. Ignored if `bufSize` is `NULL`.
-> `bufSize`	Size of new receive buffer in bytes. To remove this buffer and allocate a new default buffer (512 bytes), specify `NULL`.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.
`memErrNotEnoughSpace`	Not enough memory to allocate default buffer.

Comments

IMPORTANT: Applications must install the default buffer before closing the port (or disposing of the new receive queue.)

Compatibility Implemented only if New Serial Manager Feature Set is present.

SrmSetWakeupHandler

Purpose Installs a wakeup handler.

Prototype

```
Err SrmSetWakeupHandler(UInt16 portId,
WakeupHandlerProcPtr procP, UInt32 refCon)
```

Parameters

-> `portId`	Port ID.

-> `procP`	Pointer to a `WakeupHandlerProc` function. Specify `NULL` to remove a handler.
-> `refCon`	User-defined data that is passed to the wakeup handler function. This can be a pointer or not.

Result

0	No error.
`serErrBadPort`	This port doesn't exist.

Comments The wakeup handler function will not become active until it is primed with a number of bytes that is greater than 0, by the `SrmPrimeWakeupHandler` function. Every time a wakeup handler is called, it must be reprimed (via `SrmPrimeWakeupHandler`) in order to be called again.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmPrimeWakeupHandler`, `WakeupHandlerProc`

New Serial Manager Application-Defined Function

WakeupHandlerProc

Purpose Called after some number of bytes are received by the new serial manager's interrupt function.

Prototype

```
void WakeupHandlerProcPtr(UInt32 refCon)
```

Parameters

->`refCon`	User-defined data passed from the `SrmSetWakeupHandler` function.

Result Returns nothing.

Comments This handler function is installed by calling `SrmSetWakeupHandler`. The number of bytes after which it is called is specified by `SrmPrimeWakeupHandler`.

Because wakeup handlers are called during interrupt time, they cannot call ANY Palm OS® system functions that may block the system in any way. Wakeup handlers should also be very short so as to reduce interrupt latency.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SrmPrimeWakeupHandler`, `SrmSetWakeupHandler`

57

Script Plugin

This chapter describes the login script plugin support. You write a plugin to add to the list of available login script commands in the Network preferences panel. This chapter covers:

- Script Plugin Data Types
- Script Plugin Constants

The header file `ScriptPlugin.h` declares the API described in this chapter.

For more information on the script plugin, see the section "Extending the Network Login Script Support" on page 287 in the "Network Communication" chapter of the *Palm OS Programmer's Companion*.

Script Plugin Data Types

PluginCallbackProcType

The `PluginCallbackProcType` defines the `procP` field in `PluginExecCmdType`.

```
typedef struct {
  ScriptPluginSelectorProcPtr selectorProcP;
} PluginCallbackProcType,
*PluginCallbackProcPtr;
```

Field Descriptions

`selectorProcP`	The address of a selector-based callback function for accessing the functionality of the network interface. Each network interface provides it own `ScriptPluginSelectorProc` function. See `ScriptPluginSelectorProc`.

PluginCmdPtr

The `PluginCmdPtr` type defines a pointer to a `PluginCmdType` structure.

```
typedef PluginCmdType * PluginCmdPtr;
```

PluginCmdType

The `PluginCmdType` structure specifies the name of a command.

```
typedef struct {
  Char    commandName[pluginMaxCmdNameLen + 1];
  Boolean hasTxtStringArg;
  UInt8   reserved;
} PluginCmdType;
```

Field Descriptions

`commandName`	The name of the command. This string appears in the pull-down list in the Network preferences panel's script view. The pull-down list contains all available commands from all plugins. Make sure that your command name is unique and as short as possible.
`hasTxtStringArg`	`true` if the command takes an argument. In this case when the user selects this command, the Network preferences panel displays a field next to the command name where the user should enter the argument. This argument is passed in the `txtStringArg` field in `PluginExecCmdType` when the command is to be executed.
`reserved`	Reserved for future use.

PluginExecCmdType

The `PluginExecCmdType` structure defines the parameter block for the `scptLaunchCmdExecuteCmd` launch code. This structure

specifies which command is to be executed and provides any necessary arguments for the command. Your plugin should respond by executing the command.

```
typedef struct {
  Char    commandName[pluginMaxCmdNameLen + 1];
  Char    txtStringArg
                [pluginMaxLenTxtStringArg + 1];
  PluginCallbackProcPtr procP;
  void *  handle;
} PluginExecCmdType, *PluginExecCmdPtr;
```

Field Descriptions

`commandName`	The command's name. This is the string that appears in the pull-down list in the script view of the Network preferences panel.
`txtStringArg`	If the command takes an argument, this field provides the argument as a string. A `NULL` value means either that the user did not provide a value, or that you didn't specify that the command takes an argument.
`procP`	A pointer to a `PluginCallbackProcType` structure, which identifies the network interface function that the plugin can use to execute the command.
`handle`	Handle to information specific to a particular connection. You must pass this value when you call the function pointed to by `procP`.

PluginInfoPtr

The `PluginInfoPtr` type defines a pointer to a `PluginInfoType` structure.

```
typedef PluginInfoType * PluginInfoPtr;
```

PluginInfoType

The `PluginInfoType` structure is the parameter block for the `scptLaunchCmdListCmds` launch code. When your plugin receives the launch code, the `PluginInfoType` structure is empty. Your plugin should fill in the `PluginInfoType` and return it. The system uses the information returned to construct the pull-down list of available script commands and build a table of which plugin will execute which script command.

```
typedef struct {
  Char  pluginName[pluginMaxModuleNameLen + 1];
  UInt16  numOfCommands;
  PluginCmdType command[pluginMaxNumOfCmds];
} PluginInfoType;
```

Field Descriptions

`pluginName`	A name that the system can use to identify your plugin. This is typically the same name you give the PRC file.
`numOfCommands`	The number of commands that your plugin defines. The maximum allowed is `pluginMaxNumOfCmds`.
`command`	An array of `PluginCmdType` structures that provide information about the commands that your plugin defines.

ScriptPluginLaunchCodesEnum

The `ScriptPluginLaunchCodesEnum` defines the launch codes for the script plugin. Your script plugin's `PilotMain` function should respond to the launch codes defined in this enum.

```
typedef enum {
  scptLaunchCmdDoNothing =
      sysAppLaunchCmdCustomBase,
  scptLaunchCmdListCmds,
  scptLaunchCmdExecuteCmd
```

```
} ScriptPluginLaunchCodesEnum;
```

Value Descriptions

`scptLaunchCmdDoNothing`	This launch code is a no-op supplied only to provide a beginning value for the script plugin launch codes. It is not necessary to respond to this launch code.
`scptLaunchCmdListCmds`	Provide information about the commands that your plugin executes. See `PluginInfoType`.
`scptLaunchCmdExecuteCmd`	Execute the specified command. This launch code is received when the system is executing a user's login script during a network connection attempt. Your plugin should respond by executing the command provided in the `PluginExecCmdType` parameter block.

Script Plugin Constants

Command Constants

The following constants identify the available commands that the network interface can perform for you. These commands are building blocks that you use to create your own script commands. To perform one of these tasks, pass the constant value as an argument to the network interface's callback function (`ScriptPluginSelectorProc`).

Constant	Value	Description
`pluginNetLibDoNothing`	0	For debugging purposes.
`pluginNetLibReadBytes`	1	Read the specified number of bytes from the open connection.
`pluginNetLibWriteBytes`	2	Write the specified number of bytes to the open connection.
`pluginNetLibGetUserName`	3	Get the user name from the network service profile.
`pluginNetLibGetUserPwd`	4	Get the user's password from the network service profile.
`pluginNetLibCheckCancelStatus`	5	Check to see if the user canceled the connection.
`pluginNetLibPromptUser`	6	Prompt the user for input.
`pluginNetLibConnLog`	7	Write a string to the network service's connection log.
`pluginNetLibCallUIProc`	8	Have the network interface call a function in your plugin that displays UI. Use this command if you need to display a more complicated user interface than the simple user prompt that the network interface provides.
`pluginNetLibGetSerLibRefNum`	9	Obtain the serial library's reference number. You need the reference number to perform any serial library commands, which you might need to perform more complex work with the connection port.

Size Constants

The following table lists constants that control the size of strings in your plugin and the size of the plugin itself.

Constant	Value	Description
`pluginMaxCmdNameLen`	15	The maximum length for the command's name, not including the terminating `NULL` character. This is the string displayed to the user in the pull-down menu.
`pluginMaxModuleNameLen`	15	The maximum length for the plugin's name (not including the terminating `NULL` character), which is typically the name of the PRC file as well.
`pluginMaxNumOfCmds`	10	The maximum number of commands that your plugin can define.
`pluginMaxLenTxtStringArg`	63	The maximum length of the argument that each command can take, not including the terminating `NULL` character.

Script Plugin Functions

ScriptPluginSelectorProc

Purpose A function provided by the network interface for the purpose of performing script commands.

Prototype

```
Err (*ScriptPluginSelectorProcPtr) (void *handle,
UInt16 command, void *dataBufferP, UInt16 *sizeP,
UInt16 *dataTimeoutP, void *procAddrP);
```

Parameters

-> `handle` Handle to information specific to a particular connection.

-> `command`	The command to be executed. See "Command Constants" for a list of possible values. The rest of the parameters to this callback function are interpreted differently based on the value of the `command` parameter. See the table in the "Comments" section for specifics.
<-> `dataBufferP`	A pointer to arguments to pass to the command or a pointer to data returned by the command. See the "Comments" section below.
<-> `sizeP`	The size of `dataBufferP`.
-> `dataTimeoutP`	Number of seconds to wait for the command to execute. 0 means wait forever. Applies only to commands that request information from the network.
-> `procAddrP`	Pointer to a user interface callback function that the network interface should call to complete the function. Used only by `pluginNetLibCallUIProc`. This function should take one argument of the same type that you pass to `dataBufferP` and should return `void`.

Result

Returns 0 upon success, or an error condition upon failure. If an error condition is returned, your plugin should stop processing and return the error condition from its `PilotMain`.

Comments

When your plugin receives the `scptLaunchCmdExecuteCmd` launch code, the parameter block contains the command's name, its text string argument (if any), and a pointer to the network interface's callback function. You should use this callback function any time you need to communicate with the network library, the user, or the host computer during execution of your command.

The callback function takes as arguments the handle to information about this connection (which is also passed in the launch code's parameter block), and the command that the service should execute. The rest of the parameters are interpreted differently based on what the value the command argument is. See the table below.

pluginNetLib	dataBufferP	sizeP	dataTimeOutP	procAddrP
`DoNothing`	N/A	N/A	N/A	N/A
`ReadBytes`	On return, contains the bytes that were read.	On input, contains the number of bytes to read. On return, contains the number of bytes actually read.	Number of seconds to wait before timing out the operation.	N/A
`WriteBytes`	On input, contains the data to send.	On input, contains the number of bytes to send. On return, contains the number of bytes actually sent.	Number of seconds to wait for a response before canceling.	N/A
`UserName`	On return, contains the user's name	On return, contains the size of the string pointed to by `dataBufferP`.	N/A	N/A
`UserPwd`	On return, contains the user's password.	On return, contains the size of the string pointed to by `dataBufferP`.	N/A	N/A

pluginNetLib	dataBufferP	sizeP	dataTimeOutP	procAddrP
`CheckCancelStatus`	On return, the Boolean value `true` if the user canceled the command, `false` otherwise.	Size of Boolean.	N/A	N/A
`PromptUser`	On input, the prompt to display. On return, the text that the user entered.	On input and on return, the size of the string pointed to by `dataBufferP`.	N/A	N/A
`ConnLog`	The string that should be written to the log.	N/A	N/A	N/A
`CallUIProc`	A pointer to a structure to pass to your callback function as a parameter. This structure should contain a handle to the form to be displayed, plus any other necessary information.	N/A	N/A	A pointer to a function in your plugin that displays the form.
`GetSerLibRefNum`	On return, contains the serial library's reference number.	N/A	N/A	N/A

58

Serial Manager

This chapter provides reference material for the serial manager API:

- Serial Manager Data Structures
- Serial Manager Functions

The header file `SerialMgrOld.h` declares the serial manager API. For more information on the serial manager, see the chapter "Serial Communication" in the *Palm OS Programmer's Companion*.

Serial Manager Data Structures

SerCtlEnum

To perform a control function, applications call `SerControl`, which performs one of the control operations specified by `SerCtlEnum`, which has the following elements:

Element	Description
`serCtlFirstReserved` = 0	Reserve 0
`serCtlStartBreak`	Turn RS232 break signal on. Applications have to make sure that the break is set long enough to generate a value BREAK! `valueP = 0; valueLenP = 0`
`serCtlStopBreak`	Turn RS232 break signal off: `valueP = 0; valueLenP = 0`
`serCtlBreakStatus`	Get RS232 break signal status (on or off): `valueP` = ptr to Word for returning status (0 = off, !0 = on) `*valueLenP = sizeof(Word)`

Element	Description
`serCtlStartLocalLoopback`	Start local loopback test; `valueP = 0, valueLenP = 0`
`serCtlStopLocalLoopback`	Stop local loopback test `valueP = 0, valueLenP = 0`
`serCtlMaxBaud`	`valueP = ptr` to `DWord` for returned baud *valueLenP `= sizeof(DWord)`
`serCtlHandshakeThreshold`	Retrieve HW handshake threshold; this is the maximum baud rate that does not require hardware handshaking `valueP = ptr` to DWord for returned baud `*valueLenP = sizeof(DWord)`
`serCtlEmuSetBlockingHook`	Set a blocking hook routine. **WARNING!** WARNING: For use with the Simulator on Mac OS only: NOT SUPPORTED ON THE PALM DEVICE. `valueP = ` ptr to SerCallbackEntryType `*valueLenP=sizeof(SerCallbackEntryType)` Returns the old settings in the first argument.
`serCtlLAST`	Add new address entries before this one.

SerSettingsType

The `SerSettingsType` structure defines serial port attributes; it is used by the calls `SerGetSettings` and `SerSetSettings`. The `SerSettingsPtr` type points to a `SerSettingsType` structure.

```
typedef struct SerSettingsType {
  UInt32 baudRate;
  UInt32 flags;
  Int32 ctsTimeout;
  } SerSettingsType;

typedef SerSettingsType* SerSettingsPtr;
```

Field Descriptions

`baudRate`	Baud rate
`flags`	Miscellaneous settings
`ctsTimeout`	Maximum number of ticks to wait for CTS to become asserted before transmitting; used only when configured with the `serSettingsFlagCTSAutoM` flag.

Serial Manager Functions

SerClearErr

Purpose Reset the serial port's line error status.

Prototype

```
Err SerClearErr (UInt16 refNum)
```

Parameters

-> `refNum`	The serial library reference number.

Result

0	No error.

Comments Call `SerClearErr` only after a serial manager function (`SerReceive`, `SerReceiveCheck`, `SerSend`, etc.) returns with the error code `serErrLineErr`.

The reason for this is that `SerClearErr` resets the serial port. So, if `SerClearErr` is called unconditionally while a byte is coming into the serial port, that byte is guaranteed to become corrupted.

The right strategy is to always check the error code returned by a serial manager function. If it 's `serErrLineErr`, call `SerClearErr` immediately. However, don't make unsolicited calls to `SerClearErr`.

When you get `serErrLineErr`, consider flushing the receive queue for a fraction of a second by calling `SerReceiveFlush`. `SerReceiveFlush` calls `SerClearErr` for you.

SerClose

Purpose Release the serial port previously acquired by `SerOpen`.

Prototype

```
Err SerClose (UInt16 refNum)
```

Parameters

-> `refNum`	Serial library reference number.

Result

0	No error.
`serErrNotOpen`	Port wasn't open.
`serErrStillOpen`	Port still held open by another process.

Comments Releases the serial port and shuts down serial port hardware if the open count has reached 0. Open serial ports consume more energy from the device's batteries; it's therefore essential to keep a port open only as long as necessary.

Caveat Don't call `SerClose` unless the return value from `SerOpen` was 0 (zero) or `serErrAlreadyOpen`.

See Also SerOpen

SerControl

Purpose Perform a control function.

Prototype

```
Err SerControl (UInt16 refNum, UInt16 op,
void *valueP, UInt16 *valueLenP)
```

Parameters

-> `refNum`	Reference number of library.
-> `op`	Control operation to perform (`SerCtlEnum`).
<-> `valueP`	Pointer to value for operation.
<-> `valueLenP`	Pointer to size of value.

Result

0	No error.
`serErrBadParam`	Invalid parameter (unknown).

`serErrNotOpen`	Library not open.

Comments This function provides extensible control features for the serial manager. You can

- Turn on/off the RS232 break signal and check its status.
- Perform a local loopback test.
- Get the maximum supported baud rate.
- Get the hardware handshake threshold baud rate.

There is one emulator-only control, `serCtlEmuSetBlockingHook`. See Using the Serial Manager for more information.

Compatibility Implemented only if 2.0 New Feature Set is present.

SerGetSettings

Purpose Fill in the `SerSettingsType` structure with current serial port attributes.

Prototype

```
Err SerGetSettings (UInt16 refNum,
SerSettingsPtr settingsP)
```

Parameters

-> `refNum`	Serial library reference number.
<-> `settingsP`	Pointer to `SerSettingsType` structure to be filled in.

Result

0	No error.
`serErrNotOpen`	The port wasn't open.

Comments The information returned by this call includes the current baud rate, CTS timeout, handshaking options, and data format options.

See the `SerSettingsType` structure for more details.

See Also SerSend

SerGetStatus

Purpose Return the pending line error status for errors that have been detected since the last time `SerClearErr` was called.

Prototype

```
UInt16 SerGetStatus (UInt16 refNum,
Boolean *ctsOnP, Boolean *dsrOnP)
```

Parameters

-> `refNum`	Serial library reference number.
-> `ctsOnP`	Pointer to location for storing a Boolean value.
-> `dsrOnP`	Pointer to location for storing a Boolean value.

Result Returns any combination of the following constants, bitwise ORed together:

`serLineErrorParity`
Parity error.

`serLineErrorHWOverrun`
Hardware overrun.

`serLineErrorFraming`
Framing error.

`serLineErrorBreak`
Break signal detected.

`serLineErrorHShake`
Line handshake error.

`serLineErrorSWOverrun`
Software overrun.

Comments When another serial manager function returns an error code of `serErrLineErr`, `SerGetStatus` can be used to find out the specific nature of the line error(s).

The values returned via `ctsOnP` and `dsrOnP` are not meaningful in the present version of the software

See Also SerClearErr

SerOpen

Purpose

Acquire and open a serial port with given baud rate and default settings.

Prototype

```
Err SerOpen (UInt16 refNum, UInt16 port,
UInt32 baud)
```

Parameters

-> `refNum`	Serial library reference number.
-> `port`	Port number.
-> `baud`	Baud rate.

Result

0	No error.
`serErrAlreadyOpen`	Port was open. Enables port sharing by "friendly" clients (not recommended).
`serErrBadParam`	Invalid parameter.
`memErrNotEnoughSpace`	Insufficient memory.

Comments

Acquires the serial port, powers it up, and prepares it for operation. To obtain the serial library reference number, call `SysLibFind` with "Serial Library" as the library name. This reference number must be passed as a parameter to all serial manager functions. The device currently contains only one serial port with port number 0 (zero).

The baud rate is an integral baud value (for example - 300, 1200, 2400, 9600, 19200, 38400, 57600, etc.). The Palm OS® device has been tested at the standard baud rates in the range of 300 - 57600 baud. Baud rates through 1 Mbit are theoretically possible. Use CTS handshaking at baud rates above 19200 (see `SerSetSettings`).

An error code of 0 (zero) or `serErrAlreadyOpen` indicates that the port was successfully opened. If the port is already open when `SerOpen` is called, the port's open count is incremented and an error code of `serErrAlreadyOpen` is returned. This ability to open the serial port multiple times allows cooperating tasks to share the serial port. Other tasks must refrain from using the port if

`serErrAlreadyOpen` is returned and close it by calling `SerClose`.

SerReceive

Purpose Receives `size` bytes worth of data or returns with error if a line error or timeout is encountered.

Prototype

```
UInt32 SerReceive (UInt16 refNum, void *bufP,
UInt32 count, Int32 timeout, Err* errP)
```

Parameters

`refNum`	Serial library reference number.
<-> `bufP`	Buffer for receiving data.
-> `count`	Number of bytes to receive.
-> `timeout`	Interbyte timeout in ticks, 0 for none, -1 forever.
<-> `errP`	For returning error code.

Result Number of bytes received:

`*errP = 0`	No error.
`serErrLineErr`	RS232 line error.
`serErrTimeOut`	Interbyte timeout.

Compatibility Implemented only if 2.0 New Feature Set is present.

NOTE: The old versions of `SerSend` and `SerReceive` are still available as `SerSend10` and `SerReceive10` (not V10).

See Also SerReceive10

SerReceive10

Purpose Receive a stream of bytes.

Prototype

```
Err SerReceive10 (UInt16 refNum, void *bufP,
UInt32 bytes, Int32 timeout)
```

Parameters

-> `refNum`	The serial library reference number.
-> `bufP`	Pointer to the buffer for receiving data.
-> `bytes`	Number of bytes desired.
-> `timeout`	Interbyte time out in system ticks (-1 = forever).

Result

0	No error. Requested number of bytes was received.
`serErrTimeOut`	Interbyte time out exceeded while waiting for the next byte to arrive.
`serErrLineErr`	Line error occurred (see `SerClearErr` and `SerGetStatus`).

Comments `SerReceive` blocks until all the requested data has been received or an error occurs. Because this call returns immediately without any data if line errors are pending, it is important to acknowledge the detection of line errors by calling `SerClearErr`. If you just need to retrieve all or some of the bytes which are already in the receive queue, call `SerReceiveCheck` first to get the count of bytes presently in the receive queue.

Compatibility This function corresponds to the 1.0 version of `SerReceive`.

SerReceiveCheck

Purpose Return the count of bytes presently in the receive queue.

Prototype

```
Err SerReceiveCheck (UInt16 refNum,
UInt32 *numBytesP)
```

Parameters

-> `refNum`	Serial library reference number.
<-> `numBytesP`	Pointer to location for returning the byte count.

Result

0	No error.
`serErrLineErr`	Line error pending (see `SerClearErr` and `SerGetStatus`).

Comments Because this call does not return the byte count if line errors are pending, it is important to acknowledge the detection of line errors by calling `SerClearErr`.

See Also SerReceiveWait

SerReceiveFlush

Purpose Discard all data presently in the receive queue and flush bytes coming into the serial port. Clear the saved error status.

Prototype

```
void SerReceiveFlush (UInt16 refNum,
Int32 timeout)
```

Parameters

-> `refNum`	Serial library reference number.
-> `timeout`	Interbyte time out in system ticks (-1 = forever).

Result Returns nothing.

Comments `SerReceiveFlush` blocks until a timeout occurs while waiting for the next byte to arrive.

SerReceiveWait

Purpose

Wait for at least `bytes` bytes of data to accumulate in the receive queue.

Prototype

```
Err SerReceiveWait (UInt16 refNum, UInt32 bytes,
Int32 timeout)
```

Parameters

-> `refNum`	Serial library reference number.
-> `bytes`	Number of bytes desired.
-> `timeout`	Interbyte timeout in system ticks (-1 = forever).

Result

0	No error.
`serErrTimeOut`	Interbyte timeout exceeded while waiting for next byte to arrive.
`serErrLineErr`	Line error occurred (see `SerClearErr` and `SerGetStatus`).

Comments

This is the preferred method of waiting for serial input, since it blocks the current task and allows switching the processor into a more energy-efficient state.

`SerReceiveWait` blocks until the desired number of bytes accumulate in the receive queue or an error occurs. The desired number of bytes must be less than the current receive queue size. The default queue size is 512 bytes. Because this call returns immediately if line errors are pending, it is important to acknowledge the detection of line errors by calling `SerClearErr`.

See Also

`SerReceiveCheck`, `SerSetReceiveBuffer`

SerSend

Purpose Send one or more bytes of data over the serial port.

Prototype

```
UInt32 SerSend (UInt16 refNum, void *bufP,
UInt32 count, Err *errP
```

Parameters

-> `refNum`	Serial library reference number.
-> `bufP`	Pointer to data to send.
-> `count`	Number of bytes to send.
<-> `errP`	For returning error code.

Result Returns the number of bytes transferred.

Stores in `errP`:

0	No error.
`serErrTimeOut`	Handshake timeout.

The old calls worked, but they did not return enough info when they failed. The new calls (available in Palm OS v2.0 and greater) add more parameters to solve this problem and make serial communications programming simpler.

Don't call the new functions when running on Palm OS 1.0.

Compatibility Implemented only if 2.0 New Feature Set is present.

NOTE: The old versions of `SerSend` and `SerReceive` are still available as `SerSend10` and `SerReceive10` (not V10).

See Also `SerSend10`, `SerSendWait`

SerSend10

Purpose Send a stream of bytes to the serial port.

Prototype

```
Err SerSend10 (UInt16 refNum, void *bufP,
UInt32 size)
```

Parameters

-> `refNum`	Serial library reference number.
-> `bufP`	Pointer to the data to send.
-> `size`	Size (in number of bytes) of the data to send.

Result

0	No error.
`serErrTimeOut`	Handshake timeout (such as waiting for CTS to become asserted).

Comments In the present implementation, `SerSend10` blocks until all data is transferred to the UART or a timeout error (if CTS handshaking is enabled) occurs. Future implementations may queue up the request and return immediately, performing transmission in the background. If your software needs to detect when all data has been transmitted, see `SerSendWait`.

This routine observes the current CTS time out setting if CTS handshaking is enabled (see `SerGetSettings` and `SerSend`).

Compatibility This function corresponds to the 1.0 version of `SerSend`.

See Also `SerSend`, `SerSendWait`

SerSendFlush

Purpose Discard all data presently in the transmit queue.

Prototype

```
Err SerSendFlush (UInt16 refNum)
```

Parameters

-> `refNum`	Serial library reference number.

Result

0	No error.

See Also `SerSend`, `SerSendWait`

SerSendWait

Purpose Wait until the serial transmit buffer empties.

Prototype

```
Err SerSendWait (UInt16 refNum, Int32 timeout)
```

Parameters

-> `refNum`	Serial library reference number.
-> `timeout`	Reserved for future enhancements. Set to (-1) for compatibility.

Result

0	No error.
`serErrTimeOut`	Handshake timeout (such as waiting for CTS to become asserted).

Comments `SerSendWait` blocks until all data is transferred or a timeout error (if CTS handshaking is enabled) occurs. This routine observes the current CTS timeout setting if CTS handshaking is enabled (see `SerGetSettings` and `SerSend`).

See Also `SerSend`

SerSetReceiveBuffer

Purpose Replace the default receive queue. To restore the original buffer, pass `bufSize` = 0.

Prototype

```
Err SerSetReceiveBuffer (UInt16 refNum,
void *bufP, UInt16 bufSize)
```

Parameters

-> `refNum`	Serial library reference number.
-> `bufP`	Pointer to buffer to be used as the new receive queue.
-> `bufSize`	Size of buffer, or 0 to restore the default receive queue.

Result Returns 0 if successful.

Comments The specified buffer needs to contain 32 extra bytes for serial manager overhead (its size should be your application's requirement plus 32 bytes). The default receive queue must be restored before the serial port is closed. To restore the default receive queue, call `SerSetReceiveBuffer` passing 0 (zero) for the buffer size. The serial manager does not free the custom receive queue.

SerSetSettings

Purpose Set the serial port settings; that is, change its attributes.

Prototype

```
Err SerSetSettings (UInt16 refNum,
SerSettingsPtr settingsP)
```

Parameters

-> `refNum`	Serial library reference number.
<-> `settingsP`	Pointer to the filled in `SerSettingsType` structure.

Result

0	No error.
`serErrNotOpen`	The port wasn't open.

`serErrBadParam` Invalid parameter.

Comments The attributes set by this call include the current baud rate, CTS timeout, handshaking options, and data format options. See the definition of the `SerSettingsType` structure for more details.

To do 7E1 transmission, `OR` together:

```
serSettingsFlagBitsPerChar7 |
serSettingsFlagParityOnM |
serSettingsFlagParityEvenM |
serSettingsFlagStopBits1
```

If you're trying to communicate at speeds greater than 19.2 Kbps, you need to use hardware handshaking: `serSettingsFlagRTSAutoM | serSettingsFlagCTSAutoM`.

See Also `SerGetSettings`

59

Serial and Virtual Drivers

This chapter provides reference material for the new serial manager device driver API:

- Driver Data Structures
- Driver Constants
- Serial Driver-Defined Functions
- Virtual Driver-Defined Functions
- Serial Manager Queue Functions

The header file `SerialSdrv.h` declares the serial driver API and the file `SerialVdrv.h` declares the virtual driver API. Both types of drivers also use the `SerialDrvr.h` header file. For more information on writing device drivers for the new serial manager, see section Writing a Serial or Virtual Device Driver in the chapter Serial Communication in the *Palm OS Programmer's Companion*.

Driver Data Structures

DrvrInfoType

The `DrvrInfoType` structure defines information about the serial hardware. It is passed to and filled in by the `DrvEntryPoint` function for a serial driver and the `DrvEntryPoint` for a virtual driver.

```
typedef struct {
UInt32 drvrID;
UInt32 drvrVersion;
UInt32 maxBaudRate;
UInt32 handshakeThreshold;
UInt32 portFlags;
```

```
Char * portDesc;
DrvrIRQEnum irqType;
UInt8 reserved;
} DrvrInfoType;
```

Value Descriptions

`drvrID`	4-character creator type, such as 'u328'
`drvrVersion`	Version of code that works for this hardware. For this release, all serial drivers should return version `kDrvrVersion` here.
`maxBaudRate`	Maximum baud rate supported by this hardware
`handshakeThreshold`	Baud rate at which hardware handshaking is necessary to be used
`portFlags`	Bit flags denoting features of this hardware. The flags are described in Port Feature Constants.
`portDesc`	Pointer to null-terminated string describing this hardware. This string appears in the Connection panel to describe the port to the user (only if the `portCncMgrVisible` bit in `portFlags` is set). Can be `NULL` if the driver contains a resource (of type 'tSTR' and id `kPortDescStrID`) that supplies this string.
`irqType`	IRQ line being used for this hardware. Specify one of the `DrvrIRQEnum` values. For a virtual driver, specify `drvrIRQNone`.
`reserved`	Reserved for future use.

DrvrRcvQType

The `DrvrRcvQType` structure defines the virtual driver receive buffer and function pointers to functions that access and save data to the buffer. A pointer to this structure is passed to the VdrvOpen function. The `DrvrHWRcvQPtr` type defines a pointer to a `DrvrRcvQType` structure.

```
typedef struct DrvrRcvQType {
void *rcvQ;
WriteByteProcPtr qWriteByte;
WriteBlockProcPtr qWriteBlock;
GetSizeProcPtr qGetSize;
GetSpaceProcPtr qGetSpace;
} DrvrRcvQType;

typedef DrvrRcvQType *DrvrHWRcvQPtr;
```

Value Descriptions

`rcvQ`	Pointer to the receive buffer.
`qWriteByte`	Function pointer to a function that the virtual driver can use to write one byte to the new serial manager's receive queue. See the `WriteByte` function.
`qWriteBlock`	Function pointer to a function that the virtual driver can use to write a block of bytes to the new serial manager's receive queue. See the `WriteBlock` function.
`qGetSize`	Function pointer to a function that the virtual driver can use to get the total size of the new serial manager's receive queue. See the `GetSize` function.
`qGetSpace`	Function pointer to a function that the virtual driver can use to get the available space in the new serial manager's receive queue. See the `GetSpace` function.

DrvrStatusEnum

The `DdrvStatusEnum` enumerated type specifies serial status bit flags. Return these enumerated types from the `SdrvStatus` and `VdrvStatus` calls.

```
typedef enum DrvrStatusEnum {
drvrStatusCtsOn = 0x0001,
drvrStatusRtsOn = 0x0002,
drvrStatusDsrOn = 0x0004,
drvrStatusTxFifoFull = 0x0008,
drvrStatusTxFifoEmpty = 0x0010,
drvrStatusBreakAsserted = 0x0020,
drvrStatusDataReady = 0x0040, // For polling mode
debugger only
drvrStatusLineErr = 0x0080 // For polling mode
debugger only
} DrvrStatusEnum;
```

Value Descriptions

`drvrStatusCtsOn`	Set if CTS line is active.
`drvrStatusRtsOn`	Set if RTS line is active.
`drvrStatusDsrOn`	Set if DSR is on.
`drvrStatusTxFifoFull`	Set if transmit FIFO is full; cleared if FIFO has space.
`drvrStatusTxFifoEmpty`	Set if transmit FIFO is empty.
`drvrStatusBreakAsserted`	Set if sending break characters is enabled.
`drvrStatusDataReady`	Used by debugger only.
`drvrStatusLineErr`	Used by debugger only.

SdrvAPIType

The `SdrvAPIType` structure defines the function pointers to the required serial driver functions. When passed a pointer to this

structure in the DrvEntryPoint function, that function must fill in the pointers to the serial driver functions appropriately.

```
typedef struct {
SdrvOpenProcPtr drvOpen;
SdrvCloseProcPtr drvClose;
SdrvControlProcPtr drvControl;
SdrvStatusProcPtr drvStatus;
SdrvReadCharProcPtr drvReadChar;
SdrvWriteCharProcPtr drvWriteChar;
} SdrvAPIType;
```

Value Descriptions

`drvOpen`	Pointer to the driver open function.
`drvClose`	Pointer to the driver close function.
`drvControl`	Pointer to the driver control function.
`drvStatus`	Pointer to the driver status function.
`drvReadChar`	Pointer to the driver read character function.
`drvWriteChar`	Pointer to the driver write character function.

SdrvCtlOpCodeEnum

The `SdrvCtlOpCodeEnum` enumerated type specifies a serial control operation. You should handle each of these enumerated types when passed for the `controlCode` parameter to the SdrvControl call.

```
typedef enum SdrvCtlOpCodeEnum {
  sdrvOpCodeNoOp = 0,
  sdrvOpCodeSetBaudRate = 0x1000,
  sdrvOpCodeSetSettingsFlags,
  sdrvOpCodeClearErr,
  sdrvOpCodeEnableUART,
  sdrvOpCodeDisableUART,
  sdrvOpCodeEnableUARTInterrupts,
  sdrvOpCodeDisableUARTInterrupts,
  sdrvOpCodeSetSleepMode,
```

```
  sdrvOpCodeSetWakeupMode,
  sdrvOpCodeRxEnable,
  sdrvOpCodeRxDisable,
  sdrvOpCodeLineEnable,
  sdrvOpCodeFIFOCount,
  sdrvOpCodeEnableIRDA,
  sdrvOpCodeDisableIRDA,
  sdrvOpCodeStartBreak,
  sdrvOpCodeStopBreak,
  sdrvOpCodeStartLoopback,
  sdrvOpCodeStopLoopback,
  sdrvOpCodeFlushTxFIFO,
  sdrvOpCodeFlushRxFIFO,
  sdrvOpCodeGetOptTransmitSize,
  sdrvOpCodeEnableRTS,
  sdrvOpCodeDisableRTS,
  sdrvOpCodeUserDef = 0x2000
} SdrvCtlOpCodeEnum;
```

Value Descriptions

`sdvrOpCodeSetBaudRate`	Sets the baud rate for the UART.
`sdvrOpCodeSetSettingsFlags`	Sets the data transmission options. The bit flags are described in Serial Settings Constants.
`sdvrOpCodeClearError`	Clears the hardware error state.
`sdvrOpCodeEnableUart`	Powers-up the UART and the line-drivers.
`sdvrOpCodeDisableUART`	Powers-down the UART and the line drivers.
`sdvrOpCodeEnableUARTInterrupts`	Enables the appropriate UART receive interrupts.
`sdvrOpCodeDisableUARTInterrupt s`	Disables all UART interrupts.
`sdvrOpCodeSetSleepMode`	Puts the UART in sleep mode.
`sdvrOpCodeSetWakeupMode`	Wakes up the UART from sleep mode.

`sdvrOpCodeRxEnable`	Enables the receive FIFO, enables UART interrupts, and does whatever else is necessary to allow the UART to receive data.
`sdvrOpCodeRxDisable`	Disables the receive FIFO and UART interrupts and does whatever is needed to prevent the UART from receiving data.
`sdvrOpCodeLineEnable`	Enables the main serial line driver for the UART.
`sdvrOpCodeFIFOCount`	Returns the number of bytes currently in the FIFO (or best estimate).
`sdvrOpCodeEnableIRDA`	Enable the IRDA mode and power up the IR line drivers.
`sdvrOpCodeDisableIRDA`	Disable the IRDA mode and disable the IR line drivers.
`sdvrOpCodeStartBreak`	Sends a break character or enables the sending of break characters.
`sdvrOpCodeStopBreak`	Stops sending break characters.
`sdvrOpCodeStartLoopback`	Places the UART in loopback mode.
`sdvrOpCodeStopLoopback`	Stops loopback mode.
`SdrvOpCodeFlushTxFIFO`	Flushes the contents of the transmit FIFO.
`sdrvOpCodeFlushRxFIFO`	Flushes the contents of the receive FIFO.
`sdrvOpCodeGetOptTransmitSize`	Returns the optimum buffer size for sending data or returns 0 to specify any buffer size is acceptable.
`sdrvOpCodeEnableRTS`	Asserts the RTS line.
`sdrvOpCodeDisableRTS`	Deasserts the RTS line.
`sdvrOpCodeUserDef`	User defined function invoked via `SrmControl`.

VdrvAPIType

The `VdrvAPIType` structure defines function pointers to the required virtual driver functions. When passed a pointer to this structure in the `DrvEntryPoint` function, that function must fill in the pointers to the virtual driver functions appropriately.

```
typedef struct {
VdrvOpenProcPtr drvOpen;
VdrvCloseProcPtr drvClose;
VdrvControlProcPtr drvControl;
VdrvStatusProcPtr drvStatus;
VdrvReadProcPtr drvRead;
VdrvWriteProcPtr drvWrite;
} VdrvAPIType;
```

Value Descriptions

`drvOpen`	Pointer to the driver open function.
`drvClose`	Pointer to the driver close function.
`drvControl`	Pointer to the driver control function.
`drvStatus`	Pointer to the driver status function.
`drvRead`	Pointer to the driver read function.
`drvWrite`	Pointer to the driver write function.

VdrvCtlOpCodeEnum

The `VdrvCtlOpCodeEnum` enumerated type specifies a serial control operation. You should handle each of these enumerated types when passed for the `controlCode` parameter to the `VdrvControl` call.

```
typedef enum VdrvCtlOpCodeEnum {
  vdrvOpCodeNoOp = 0,
  vdrvOpCodeSetBaudRate = 0x1000,
  vdrvOpCodeSetSettingsFlags,
  vdrvOpCodeSetCtsTimeout,
  vdrvOpCodeClearErr,
```

```
  vdrvOpCodeSetSleepMode,
  vdrvOpCodeSetWakeupMode,
  vdrvOpCodeFIFOCount,
  vdrvOpCodeStartBreak,
  vdrvOpCodeStopBreak,
  vdrvOpCodeStartLoopback,
  vdrvOpCodeStopLoopback,
  vdrvOpCodeFlushTxFIFO,
  vdrvOpCodeFlushRxFIFO,
  vdrvOpCodeSendBufferedData,
  vdrvOpCodeRcvCheckIdle,
  vdrvOpCodeEmuSetBlockingHook,
  vdrvOpCodeGetOptTransmitSize,
  vdrvOpCodeGetMaxRcvBlockSize,
  vdrvOpCodeNotifyBytesReadFromQ,
  vdrvOpCodeUserDef = 0x2000
} VdrvCtlOpCodeEnum;
```

Value Descriptions

`vdvrOpCodeSetBaudRate`	Sets the baud rate.
`vdvrOpCodeSetSettingsFlags`	Sets the data transmission options. The bit flags are described in Serial Settings Constants.
`vdrvOpCodeSetCtsTimeout`	Hardware handshake timeout.
`vdvrOpCodeClearError`	Clears the hardware error state.
`vdvrOpCodeSetSleepMode`	Puts the port in sleep mode (not typically used for virtual drivers).
`vdvrOpCodeSetWakeupMode`	Wakes up the port from sleep mode (not typically used for virtual drivers).
`vdvrOpCodeFIFOCount`	Returns the number of bytes currently in the FIFO (or best estimate).
`vdvrOpCodeStartBreak`	Sends a break character or enables the sending of break characters.
`vdvrOpCodeStopBreak`	Stops sending break characters.

`vdvrOpCodeStartLoopback`	Starts loopback mode (not typically used for virtual drivers).
`vdvrOpCodeStopLoopback`	Stops loopback mode (not typically used for virtual drivers).
`vdrvOpCodeFlushTxFIFO`	Flushes the contents of the transmit FIFO.
`vdrvOpCodeFlushRxFIFO`	Flushes the contents of the receive FIFO.
`vdrvOpCodeSendBufferedData`	Notifies virtual device to send any buffered data it has not emptied from its internal buffers.
`vdrvOpCodeRcvCheckIdle`	Called periodically to allow virtual device time to check if there is data to be received. Because virtual devices execute in the same thread as applications, they can be prevented from handling notifications of received data.
`vdrvOpCodeEmuSetBlockingHook`	Special opCode for the Simulator.
`vdrvOpCodeGetOptTransmitSize`	Returns the optimum buffer size for sending data or returns 0 to specify any buffer size is acceptable.
`vdrvOpCodeGetMaxRcvBlockSize`	Returns the maximum receive block size that the serial manager should request from the virtual device. Can be used to implement flow control.
`vdrvOpCodeNotifyBytesReadFromQ`	Tells the virtual device that some number of bytes have been read from the receive queue by the client application. Can be used to implement flow control.
`vdvrOpCodeUserDef`	User defined function invoked via `SrmControl`.

Driver Constants

Port Feature Constants

These flag constants describe serial hardware capabilities.

`portPhysicalPort`	Should be set for a physical port, unset for a virtual port
`portRS232Capable`	Set if this hardware has a RS-232 port
`portIRDACapable`	Set if this hardware has an IR port and supports IRDA mode
`portCradlePort`	Set if this hardware controls the cradle port
`portExternalPort`	Set if this hardware port is external or on a memory card
`portModemPort`	Set if this hardware communicates with a modem
`portCncMgrVisible`	Set if this serial port's name is to be displayed in the Connection panel
`portPrivateUse`	Set if this driver is for special software and not general applications.

Serial Driver-Defined Functions

The functions in this section must be defined by your serial driver.

DrvEntryPoint

Purpose Entry point for the serial driver.

Prototype

```
Err DrvEntryPoint(DrvrEntryOpCodeEnum opCode,
void * uartData)
```

Parameters

-> `opCode`	Entry function code.

<-> `uartData`	Pointer to data specific to the opCode.

Result

0	No error.
-1	The opCode is invalid or the hardware could not be found.

Comments

This functions serves a dual purpose based on the value of the `opCode` parameter. The two possible codes are `drvrEntryGetUartFeatures` and `drvrEntryGetDrvrFuncts`.

`DrvEntryPoint` is called with the `drvrEntryGetUartFeatures` code when the new serial manager is installed into the system at boot time and is looking for all UART hardware currently connected to the device. When this opCode is set, the `uartData` pointer points to a `DrvrInfoType` structure. This function does not allocate the structure, it just fills in the fields with information.

This function should check to make sure the hardware exists in the current system. If the hardware cannot be found, the function should leave the `DrvrInfoType` struct untouched and return a -1 error.

The driver needs to supply a string that describes the port it manages. This string is displayed to the user in the Connection panel and is returned by the `SrmGetDeviceInfo` function. To set this string, copy it into the `portDesc` field of the `DrvrInfoType` structure. Alternatively, you can supply this string in a driver resource of type 'tSTR' and id `kPortDescStrID`.

`DrvEntryPoint` is called with the `drvrEntryGetDrvrFuncts` code when a serial port is opened. The `uartData` pointer points to a `SdrvAPIType` structure and `DrvEntryPoint` must fill in the fields of this structure with appropriate function pointers.

Compatibility

Implemented only if New Serial Manager Feature Set is present.

SdrvClose

Purpose Handles all activities needed to power-down the UART.

Prototype

```
Err SdrvClose(SdrvDataPtr drvrDataP)
```

Parameters

-> `drvrDataP`	Pointer to the driver's private global area.

Result

0	No error.

Comments This function should disable all UART interrupts for the Dragonball processor as well as for the UART, place the UART in sleep mode, power down the transceiver, and do whatever other necessary tasks there may be. Additionally, this function should remove the interrupt handler installed by `SdrvOpen`.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SdrvControl

Purpose Extends the `SrmControl` function to the level of the hardware.

Prototype

```
Err *SdrvControl(SdrvDataPtr drvrDataP,
SdrvCtlOpCodeEnum controlCode,
void * controlDataP, UInt16 * controlDataLenP)
```

Parameters

-> `drvrDataP`	Pointer to the driver's private global area.
-> `controlCode`	Control function opCode. One of the opCodes listed in the `SdrvCtlOpCodeEnum` type.
<-> `controlDataP`	Pointer to data for the specified control function.

<-> `controlDataLenP`
Pointer to length of control data being passed in or out.

Result

0	No error.
`serErrNotSupported`	`controlCode` not supported.
`serErrBadParam`	`controlDataP` or `controlDataLenP` is bad.

Comments

This function should support the opCodes listed in the `SdrvCtlOpCodeEnum` type. If this function does not support an opCode, it must return the `serErrNotSupported` error code for that opCode.

Table 59.1 shows what is passed for the `controlDataP` and `controlDataLenP` parameters for each of the control codes that use them. Control codes not listed do not use these parameters.

Table 59.1 SdrvControl Parameters

`sdvrOpCodeSetBaudRate`	-> `controlDataP` = Pointer to `Int32` (baud rate), -> `controlDataLenP` = Pointer to `sizeof(Int32)`.
`sdvrOpCodeSetSettingsFlags`	-> controlDataP = Pointer to `UInt32` (bitfield; see Serial Settings Constants) -> controlDataLenP = Pointer to `sizeof(UInt32)`
`sdvrOpCodeFIFOCount`	-> controlDataP = Pointer to `Int16`, which contains the number of bytes in the FIFO. -> controlDataLenP = Pointer to `sizeof(Int16)`.

Table 59.1 SdrvControl Parameters *(continued)*

`sdrvOpCodeGetOptTransmitSize`	<- controlDataP = Pointer to `Int32`, <- controlDataLenP = Pointer to `sizeof(Int32)`. Return the optimum buffer size for sending data, or 0 to specify any buffer size is acceptable.
`sdvrOpCodeUserDef`	<-> `controlDataP` = Pointer from `SrmControl` (user-defined data), <-> `controlDataLenP` = Pointer to `sizeof(Int32)`.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SdrvISP

Purpose An interrupt service routine called when a hardware interrupt is generated on the IRQ line associated with the serial hardware.

Prototype

```
asm Boolean SdrvISP(UInt32 param: __A0):__D0
```

Parameters A0 = `param` Pointer to the driver's private global area.

Result D0 returns a Boolean value. Return `true` if this UART has data that needs to be read; return `false` if no other interrupt service is needed.

Comments This function can retrieve its globals from the low-memory global they were saved in (via the pointer in A0) and then must determine if the interrupt is for this particular serial hardware. If so, it must call the `saveDataProc` function (passed into `SdrvOpen`) with the portP pointer as the parameter. The `saveDataProc` function, supplied by the new serial manager, handles reading the data from the UART by calling the `SdrvReadChar` function.

The `SdrvISP` function must be installed in the appropriate IRQ handler by the `SdrvOpen` routine.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also SdrvOpen

SdrvOpen

Purpose Initializes the serial hardware to send and receive data.

Prototype

```
Err SdrvOpen(SdrvDataPtr* drvrDataP,
UInt32 baudRate, void * portP,
SerialMgrISPProcPtr saveDataProc)
```

Parameters

<-> drvrDataP	Pointer to a pointer to the driver's private global area (allocated by this function).
-> baudRate	Initial baud rate setting.
-> portP	Pointer to the open port data.
-> saveDataProc	Pointer to the function where data received by interrupt is to be saved. The typedef for this function is shown in the Comments section.

Result

0	No error.

Comments Here is the typedef for the saveDataProc function:

```
typedef void (*SerialMgrISPProcPtr)(void *
portP:__A0)
```

To accomplish serial hardware initialization, SdrvOpen must perform the following tasks:

- Allocate global data needed by the driver. There is a low memory global (GIrq*N*GlobalsP) for every IRQ line in the system. At open time, a serial driver must save its global data in this low memory global because when the interrupt is called there is no way to get the globals through the driver data parameter that the serial manager normally supplies.
- Save the portP and saveDataProc parameters passed to SdrvOpen in the global variable structure, because they are

needed when the `SdrvISP` function is called. When the interrupt routine subsequently gets called, the driver gets access to the low memory globals which contain the `saveDataProc` function and the `portP` pointer. This pointer is passed into the new serial manager, which then calls the driver `SdrvReadChar` function in order to read all the bytes and fill its queue.

- Save the pointer to the globals in the appropriate low memory global variable for the IRQ line the device is using (for example, a device which uses IRQ3 would use the `GIrq3GlobalsP`). You can find the IRQ global variables defined in the header file `globals.h`.
- Save the pointer to the globals in the `drvrDataP` parameter passed into the `SdrvOpen` function. This private global data pointer is passed to every serial driver function so they all have access to the global data.
- Patch out the appropriate interrupt handler trap and replace it with the serial driver's ISP function (`SdrvISP`). For example, the system trap to be patched for IRQ3 is called `sysTrapHwrIRQ3Handler` (see `SysTraps.h`). Be sure to save the old interrupt handler to be re-installed when `DrvClose` is called. Here is an example of how to do this:

  ```
  oldIntHandler =
  SysGetTrapAddress(sysTrapHwrIRQ3Handler);
  SysSetTrapAddress(sysTrapHwrIRQ3Handler,
  SdrvISP);
  ```

  If there is another serial device sharing the same IRQ line, you must tail-patch the IRQ handler rather than replace it. In other words, you must call the previously installed handler after your own handler executes.
- Set up and open the hardware to its default state.

Compatibility Implemented only if New Serial Manager Feature Set is present.

See Also `SdrvISP`

SdrvReadChar

Purpose Reads a byte (if available) from the receive FIFO of the UART.

Prototype

```
asm UInt16 SdrvReadChar(SdrvDataPtr
drvrDataP:__A0):__D0
```

Parameters A1 = `drvrDataP` Pointer to the driver's private global area.

Result D0 returns an `Int16` value. The returned 16-bit word contains the data byte read from the hardware in the low-order byte. If there is an error, the error code is returned in the low-order byte and the error flag (`$80`) is set in the high-order byte.

Comments This function should be written in 68K assembly language for speed, but can be written in a higher-level language as long as the register usage for the parameters and return values is obeyed. If this function is too slow, hardware overruns may occur.

This function is responsible for translating break, framing, parity, and overrun errors back to the calling function. If an error is received by the hardware, the high-order byte of the return value should be set to `$80` to mark the low-order byte as an error code and not a readable byte. The error code returned in the low-order byte of D0 should be translated into one of the following four serial manager error codes: `serLineErrorBreak`, `serLineErrorFraming`, `serLineErrorParity`, or `serLineErrorHWOverrun`.

`SdrvReadChar` executes during interrupt time, and cannot call any OS functions that are normally not allowed to be called during this time. All registers needed for this function should be saved onto the stack (except for D0). The A1 register must not be changed on exit.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SdrvStatus

Purpose Returns UART status.

Prototype

```
UInt16 SdrvStatus(SdrvDataPtr drvrDataP)
```

Parameters

-> `drvrDataP`	Pointer to the driver's private global area.

Result An unsigned long bitfield denoting the status of the UART. The individual bit flags are described in the `DrvrStatusEnum` type.

Comments The `drvrStatusCtsOn` flag should be set if the UART's CTS line is active. The `drvrStatusRtsOn` flag should be set if the RTS line for the UART is currently high. The `drvrStatusDsrOn` flag should be set if DSR is turned on. Again, this may not be supported on all UARTs and should be set or cleared based on the type of hardware used. The `drvrStatusTxFifoFull` flag is set if the transmit FIFO for the hardware has no available space to receive more data and the flag should be cleared if the transmit FIFO does have available space. And the `drvrStatusBreakAsserted` flag should be set if the UART currently has sending break characters enabled.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SdrvWriteChar

Purpose Writes a byte to the appropriate UART register for transmission.

Prototype

```
Err SdrvWriteChar(SdrvDataPtr drvrDataP,
UInt8 aChar)
```

Parameters

-> `drvrDataP`	Pointer to the driver's private global area.
-> `aChar`	Byte of data to be written to the UART.

Result

0	No error.

Compatibility Implemented only if New Serial Manager Feature Set is present.

Virtual Driver-Defined Functions

The functions in this section must be defined by your virtual driver.

DrvEntryPoint

Purpose Entry point for the virtual driver.

Prototype

```
Err DrvEntryPoint(DrvrEntryOpCodeEnum opCode,
void * uartData)
```

Parameters

-> `opCode`	Entry function code.
<-> `uartData`	Pointer to data specific to the opCode.

Result

0	No error.
-1	The opCode is invalid or the hardware could not be found.

Comments This functions serves a dual purpose based on the value of the `opCode` parameter. The two possible codes are `drvrEntryGetUartFeatures` and `drvrEntryGetDrvrFuncts`.

`DrvEntryPoint` is called with the `drvrEntryGetUartFeatures` code when the new serial manager is installed into the system at boot time and is looking for all installed drivers. When this opCode is set, the `uartData` pointer points to a `DrvrInfoType` structure. This function does not allocate the structure, it just fills in the fields with information.

This function should check to make sure the associated serial device can operate under the current OS and system settings. If the hardware cannot be found, the function should leave the `DrvrInfoType` struct untouched and return a -1 error.

The driver needs to supply a string that describes the port it manages. This string is displayed to the user in the Connection

panel and is returned by the `SrmGetDeviceInfo` function. To set this string, copy it into the `portDesc` field of the `DrvrInfoType` structure. Alternatively, you can supply this string in a driver resource of type 'tSTR' and id `kPortDescStrID`.

`DrvEntryPoint` is called with the `drvrEntryGetDrvrFuncts` code when a virtual port is opened. The `uartData` pointer points to a `VdrvAPIType` structure and `DrvEntryPoint` must fill in the fields of this structure with appropriate function pointers.

Compatibility Implemented only if New Serial Manager Feature Set is present.

VdrvClose

Purpose Handles all activities needed to close the virtual device.

Prototype

```
Err VdrvClose(VdrvDataPtr drvrDataP)
```

Parameters

-> `drvrDataP`	Pointer to the driver's private global area.

Result

0	No error.

Compatibility Implemented only if New Serial Manager Feature Set is present.

VdrvControl

Purpose Extends the `SrmControl` function to the level of the virtual device.

Prototype

```
Err *VdrvControl(VdrvDataPtr drvrDataP,
VdrvCtlOpCodeEnum controlCode,
void * controlDataP, UInt16 * controlDataLenP)
```

Parameters

-> `drvrDataP`	Pointer to the driver's private global area.
-> `controlCode`	Control function opCode. One of the opCodes listed in the `VdrvCtlOpCodeEnum` type.

<-> `controlDataP` Pointer to data for the specified control function.

<-> `controlDataLenP` Pointer to length of control data being passed in or out.

Result

0	No error.
`serErrNotSupported`	`controlCode` not supported.
`serErrBadParam`	`controlDataP` or `ControlDataLenP` is bad.

Comments

This function should support the opCodes listed in the `VdrvCtlOpCodeEnum` type. If this function does not support an opCode, it must return the `serErrNotSupported` error code for that opCode.

Table 59.2 shows what is passed for the `controlDataP` and `controlDataLenP` parameters for each of the control codes that use them. Control codes not listed do not use these parameters.

Table 59.2 VdrvControl Parameters

`vdvrOpCodeSetBaudRate`	-> `controlDataP` = Pointer to `Int32` (baud rate), -> `controlDataLenP` = Pointer to `sizeof(Int32)`.
`vdvrOpCodeSetSettingsFlags`	-> controlDataP = Pointer to `UInt32` (bitfield; see Serial Settings Constants) -> controlDataLenP = Pointer to `sizeof(UInt32)`
`vdvrOpCodeFIFOCount`	-> controlDataP = Pointer to `Int16`, which contains the number of bytes in the FIFO. -> controlDataLenP = Pointer to `sizeof(Int16)`.

`vdrvOpCodeGetOptTransmitSize`	<- controlDataP = Pointer to `Int32` (buffer size), <- controlDataLenP = Pointer to `sizeof(Int32)`. Return the optimum buffer size for sending data, or 0 to specify any buffer size is acceptable.
`vdrvOpCodeGetMaxRcvBlockSize`	<- controlDataP = Pointer to `Int32` (block size), <- controlDataLenP = Pointer to `sizeof(Int32)`. Return the maximum block size that the serial manager should request from the virtual device.
`vdrvOpCodeNotifyBytesReadFromQ`	-> controlDataP = Pointer to `Int32` (number of bytes read), -> controlDataLenP = Pointer to `sizeof(Int32)`.
`vdvrOpCodeUserDef`	<-> `controlDataP` = Pointer from `SrmControl` (user-defined data), <-> `controlDataLenP` = Pointer to `sizeof(Int32)`.

Compatibility Implemented only if New Serial Manager Feature Set is present.

VdrvOpen

Purpose Initializes the virtual device to begin communication.

Prototype

```
Err VdrvOpen(VdrvDataPtr* drvrDataP,
UInt32 baudRate, DrvrHWRcvQPtr rcvQP)
```

Parameters

<-> `drvrDataP`	Pointer to a pointer to the driver's private global area (allocated by this function). A pointer to this private global area is passed to the other virtual driver functions.

-> `baudRate`	Initial baud rate setting.
-> `rcvQP`	Pointer to the driver's receive queue buffer structure. For details on the fields, see `DrvrRcvQType`.

Result

0	No error.

Comments This function must allocate and initialize any global variables (and pass back a pointer to a pointer to them in `drvrDataP`), do any set-up necessary for communicating with other software, and save the `rcvQP` pointer since it will need the functions and pointers to structures enclosed within to be able to save received data into the new serial manager's receive queue.

Compatibility Implemented only if New Serial Manager Feature Set is present.

VdrvStatus

Purpose Returns virtual device status.

Prototype

```
UInt16 VdrvStatus(VdrvDataPtr drvrDataP)
```

Parameters

-> `drvrDataP`	Pointer to the driver's private global area.

Result An unsigned long bitfield denoting the status of the virtual device, but only if the virtual device is emulating hardware. The individual bit flags are described in the `DrvrStatusEnum` type.

Comments Generally, status is returned only to the client application using the virtual device. The new serial manager does not use status information from virtual devices.

Compatibility Implemented only if New Serial Manager Feature Set is present.

VdrvWrite

Purpose Writes a block of bytes.

Prototype

```
UInt32 VdrvWrite(VdrvDataPtr drvrDataP,
void * bufP, UInt32 size, Err* errP)
```

Parameters

-> `drvrDataP`	Pointer to the driver's private global area.
-> `bufP`	Pointer to buffer containing the data to be written to the virtual device.
-> `size`	Number of bytes in the buffer `bufP`.
<- `errP`	Pointer to an error code resulting from the operation. Zero is returned if there is no error.

Result Returns the actual number of bytes written.

Compatibility Implemented only if New Serial Manager Feature Set is present.

Serial Manager Queue Functions

The functions in this section are supplied by the new serial manager to the virtual driver via the `DrvrRcvQType` passed to the `VdrvOpen` function.

GetSize

Purpose Returns the total size of the new serial manager's receive queue.

Prototype

```
typedef UInt32 (*GetSizeProcPtr)(void * theQ)
```

Parameters

-> `theQ`	Pointer to the receive queue.

Result Size in bytes of the new serial manager's receive queue.

Comments This function is useful for implementing flow control.

Compatibility Implemented only if New Serial Manager Feature Set is present.

GetSpace

Purpose Returns the available space in the new serial manager's receive queue.

Prototype

```
typedef UInt32 (*GetSpaceProcPtr)(void * theQ)
```

Parameters

-> `theQ`	Pointer to the receive queue.

Result Size in bytes of the available space in the new serial manager's receive queue.

Comments This function is useful for implementing flow control.

Compatibility Implemented only if New Serial Manager Feature Set is present.

WriteBlock

Purpose Writes a block of bytes to the new serial manager's receive queue.

Prototype

```
typedef Err (*WriteBlockProcPtr)(void * theQ,
UInt8 * bufP, UInt16 size, UInt16 lineErrs)
```

Parameters

-> `theQ`	Pointer to the receive queue.
-> `bufP`	Pointer to the buffer holding bytes for the WriteBlock function.
-> `size`	Size of `bufP`.

-> `lineErrs`	Any serial line errors received should be reported here.

Result

0	No error.
`serErrLineErr`	There was a software overrun line error.

Compatibility Implemented only if New Serial Manager Feature Set is present.

WriteByte

Purpose Writes one byte to the new serial manager's receive queue.

Prototype

```
typedef Err (*WriteByteProcPtr)(void * theQ,
UInt8 theByte, UInt16 lineErrs)
```

Parameters

-> `theQ`	Pointer to the receive queue.
-> `theByte`	The byte to be written to the queue.
-> `lineErrs`	Any serial line errors received should be reported here.

Result

0	No error.
`serErrLineErr`	There was a software overrun line error.

Compatibility Implemented only if New Serial Manager Feature Set is present.

60

Serial Link Manager

This chapter provides reference material for the serial link manager API. The header file `SerialLinkMgr.h` declares the serial link manager API. For more information on the serial link manager, see the chapter "Serial Communication" in the *Palm OS Programmer's Companion*.

Serial Link Manager Functions

SlkClose

Purpose Close down the serial link manager.

Prototype

```
Err SlkClose (void)
```

Parameters None.

Result

0	No error.
`slkErrNotOpen`	The serial link manager was not open.

Comments When the open count reaches zero, this routine frees resources allocated by serial link manager.

SlkCloseSocket

Purpose Closes a socket previously opened with `SlkOpenSocket`.

The caller is responsible for closing the communications library used by this socket, if necessary.

Prototype

```
Err SlkCloseSocket (UInt16 socket)
```

Parameters

`socket`	The socket ID to close.

Result

0	No error.
`slkErrSocketNotOpen`	The socket was not open.

Comments `SlkCloseSocket` frees system resources the serial link manager allocated for the socket. It does not free resources allocated and passed by the client, such as the buffers passed to `SlkSetSocketListener`; this is the client's responsibility. The caller is also responsible for closing the communications library used by this socket.

See Also `SlkOpenSocket`

SlkFlushSocket

Purpose Flush the receive queue of the communications library associated with the given socket.

Prototype

```
Err SlkFlushSocket (UInt16 socket, Int32 timeout)
```

Parameters

-> `socket`	Socket ID.
-> `timeout`	Interbyte timeout in system ticks.

Result

0	No error.
`slkErrSocketNotOpen`	The socket wasn't open.

SlkOpen

Purpose Initialize the serial link manager.

Prototype

```
Err SlkOpen (void)
```

Parameters None.

Result

0	No error.
`slkErrAlreadyOpen`	No error.

Comments Initializes the serial link manager, allocating necessary resources. Return codes of 0 (zero) and `slkErrAlreadyOpen` both indicate success. Any other return code indicates failure. The `slkErrAlreadyOpen` function informs the client that someone else is also using the serial link manager. If the serial link manager was successfully opened by the client, the client needs to call `SlkClose` when it finishes using the serial link manager.

SlkOpenSocket

Purpose Open a serial link socket and associate it with a communications library. The socket may be a known static socket or a dynamically assigned socket.

Prototype

```
Err SlkOpenSocket (UInt16 portID, UInt16 *socketP,
Boolean staticSocket)
```

Parameters

`portID`	Comm library reference number for socket.
`socketP`	Pointer to location for returning the socket ID.
`staticSocket`	If `TRUE`, `*socketP` contains the desired static socket number to open. If `FALSE`, any free socket number is assigned dynamically and opened.

Result

0	No error.

`slkErrOutOfSockets`
No more sockets can be opened.

Comments The communications library must already be initialized and opened (see `SerOpen`). When finished using the socket, the caller must call `SlkCloseSocket` to free system resources allocated for the socket. For information about well-known static socket IDs, see The Serial Link Protocol.

SlkReceivePacket

Purpose Receive and validate a packet for a particular socket or for any socket. Check for format and checksum errors.

Prototype

```
Err SlkReceivePacket (UInt16 socket,
Boolean andOtherSockets, SlkPktHeaderPtr headerP,
void* bodyP, UInt16 bodySize, Int32 timeout)
```

Parameters

-> `socket` The socket ID.

-> `andOtherSockets`
If `TRUE`, ignore destination in packet header.

<-> `headerP` Pointer to the packet header buffer (size of `SlkPktHeaderType`).

<-> `bodyP` Pointer to the packet client data buffer.

-> `bodySize` Size of the client data buffer (maximum client data size which can be accommodated).

-> `timeout` Maximum number of system ticks to wait for beginning of a packet; -1 means wait forever.

Result

0 No error.

`slkErrSocketNotOpen`
The socket was not open.

`slkErrTimeOut` Timed out waiting for a packet.

`slkErrWrongDestSocket`
The packet being received had an unexpected destination.

`slkErrChecksum`	Invalid header checksum or packet CRC-16.
`slkErrBuffer`	Client data buffer was too small for packet's client data.

If `andOtherSockets` is `FALSE`, this routine returns with an error code unless it gets a packet for the specific socket.

If `andOtherSockets` is `TRUE`, this routine returns successfully if it sees any incoming packet from the communications library used by `socket`.

Comments You may request to receive a packet for the passed socket ID only, or for any open socket which does not have a socket listener. The parameters also specify buffers for the packet header and client data, and a timeout. The timeout indicates how long the receiver should wait for a packet to begin arriving before timing out. If a packet is received for a socket with a registered socket listener, it will be dispatched via its socket listener procedure. On success, the packet header buffer and packet client data buffer is filled in with the actual size of the packet's client data in the packet header's `bodySize` field.

SlkSendPacket

Purpose Send a serial link packet via the serial output driver.

Prototype

```
Err SlkSendPacket (SlkPktHeaderPtr headerP,
SlkWriteDataPtr writeList)
```

Parameters

<-> `headerP`	Pointer to the packet header structure with client information filled in (see Comments).
-> `writeList`	List of packet client data blocks (see Comments).

Result

0	No error.
`slkErrSocketNotOpen`	The socket was not open.
`slkErrTimeOut`	Handshake timeout.

Comments `SlkSendPacket` stuffs the signature, client data size, and the checksum fields of the packet header. The caller must fill in all other packet header fields. If the transaction ID field is set to 0 (zero), the serial link manager automatically generates and stuffs a new non-zero transaction ID. The array of `SlkWriteDataType` structures enables the caller to specify the client data part of the packet as a list of noncontiguous blocks. The end of list is indicated by an array element with the `size` field set to 0 (zero). This call blocks until the entire packet is sent out or until an error occurs.

SlkSetSocketListener

Purpose Register a socket listener for a particular socket.

Prototype

```
Err SlkSetSocketListener (UInt16 socket,
SlkSocketListenPtr socketP)
```

Parameters

-> `socket`	Socket ID.
-> `socketP`	Pointer to a `SlkSocketListenType` structure.

Result

0	No error.
`slkErrBadParam`	Invalid parameter.
`slkErrSocketNotOpen`	The socket was not open.

Comments Called by applications to set up a socket listener.

Since the serial link manager does not make a copy of the `SlkSocketListenType` structure, but instead saves the passed pointer to it, the structure

- must **not** be an automatic variable (that is, local variable allocated on the stack)
- may be a global variable in an application
- may be a locked chunk allocated from the dynamic heap

The `SlkSocketListenType` structure specifies pointers to the socket listener procedure and the data buffers for dispatching

packets destined for this socket. Pointers to two buffers must be specified: the packet header buffer (size of `SlkPktHeaderType`), and the packet body (client data) buffer. The packet body buffer must be large enough for the largest expected client data size. Both buffers may be application global variables or locked chunks allocated from the dynamic heap.

The socket listener procedure is called when a valid packet is received for the socket. Pointers to the packet header buffer and the packet body buffer are passed as parameters to the socket listener procedure.

NOTE: The application is responsible for freeing the `SlkSocketListenType` structure or the allocated buffers when the socket is closed. The serial link manager doesn't do it.

SlkSocketPortID

Purpose Get the port ID associated with a particular socket; for use with the new serial manager.

Prototype

```
ErrSlkSocketPortID (UInt16 socket,
UInt16 * portIDP)
```

Parameters

-> `socket`	The socket ID.
<-> `portIDP`	Pointer to location for returning the port ID.

Result

0	No error.
`slkErrSocketNotOpen`	The socket was not open.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SlkSocketSetTimeout

Purpose Set the interbyte packet receive-timeout for a particular socket.

Prototype

```
Err SlkSocketSetTimeout (UInt16 socket,
Int32 timeout)
```

Parameters

-> `socket`	Socket ID.
-> `timeout`	Interbyte packet receive-timeout in system ticks.

Result

0	No error.
`slkErrSocketNotOpen`	The socket was not open.

Part IV: Libraries

61

Internet Library

This chapter provides reference material for the Internet library API:

- Internet Library Data Structures
- Internet Library Constants
- Internet Library Functions

The header file `INetMgr.h` declares the Internet library API. For more information on the Internet library, see the chapter "Network Communication" in the *Palm OS Programmer's Companion*.

NOTE: The information in this chapter applies only to version 3.2 or later of the Palm OS® on Palm VII® devices. These features are implemented only if the Wireless Internet Feature Set is present.

WARNING! In future OS versions, Palm Computing® does not intend to support or provide backward compatibility for the Internet library API documented in this chapter.

Internet Library Data Structures

INetCompressionTypeEnum

The `INetCompressionTypeEnum` enum indicates the type of compression used for data exchanged via a socket. One of these enumerated types is set as the value of the `inetSockSettingCompressionTypeID` socket setting (a read-only setting).

```
typedef enum {
inetCompressionTypeNone = 0,
inetCompressionTypeBitPacked,
inetCompressionTypeLZ77
} INetCompressionTypeEnum;
```

Value Descriptions

`inetCompressionTypeNone`	No compression.
`inetCompressionTypeBitPacked`	Custom 5-bit compression scheme. This is typically used for data sent from the Palm Web Clipping Proxy server.
`inetCompressionTypeLZ77`	Not used; reserved for future use.

INetConfigNameType

The `INetConfigNameType` structure holds the name of an Internet library network **configuration**. A configuration is a set of specific values for the Internet library settings. The Internet library defines a set of built-in configuration aliases for common network setups. These aliases point to configurations instead of holding the actual values themselves. You can use an alias anywhere in the API you would use a configuration. System-defined configuration aliases are listed in "Configuration Aliases" on page 1116.

This structure is used in the functions `INetLibConfigIndexFromName`, `INetLibConfigRename`, and `INetLibConfigSaveAs`.

```
#define inetConfigNameSize 32;

typedef struct {
Char name[inetConfigNameSize]; // name of
configuration
} INetConfigNameType, *INetConfigNamePtr;
```

Field Description

name	A configuration name (up to 32 characters).

INetContentTypeEnum

The `INetContentTypeEnum` enum specifies the type of content to be exchanged via a socket. One of these enumerated types is set as the value of the `inetSockSettingContentTypeID` socket setting (a read-only setting).

```
typedef enum {
inetContentTypeTextPlain = 0,
inetContentTypeTextHTML,
inetContentTypeImageGIF,
inetContentTypeImageJPEG,
inetContentTypeApplicationCML,
inetContentTypeImagePalmOS,
inetContentTypeOther
} INetContentTypeEnum;
```

Value Descriptions

`inetContentTypeTextPlain`	Not used
`inetContentTypeTextHTML`	Not used
`inetContentTypeImageGIF`	Not used
`inetContentTypeImageJPEG`	Not used
`inetContentTypeApplicationCML`	Compressed HTML content (format used by the Palm Web Clipping Proxy server and PQAs)

`inetContentTypeImagePalmOS`	Palm OS® bitmap
`inetContentTypeOther`	Some undefined content type

INetHTTPAttrEnum

The `INetHTTPAttrEnum` enum specifies HTTP request and response attributes that are set by `INetLibSockHTTPAttrSet` and returned by `INetLibSockHTTPAttrGet`.

```
typedef enum {

//-----------------------------------------------
-----------
// Request only attributes
//-----------------------------------------------
-----------
// The following are ignored unless going through
a CTP proxy
inetHTTPAttrWhichPart, // (W) UInt32 (0 -> N)
inetHTTPAttrIncHTTP, // (W) UInt32 (Boolean) only
applicable
        // when inetHTTPAttrConvAlgorithm set to
ctpConvNone
inetHTTPAttrCheckMailHi, // (W) UInt32
inetHTTPAttrCheckMailLo, // (W) UInt32
inetHTTPAttrReqContentVersion, // (W) UInt32,
desired content
        // version. Represented as 2 low bytes.
Lowest byte is
        // minor version, next higher byte is
major version.
//-----------------------------------------------
--------------
// Response only attributes
//-----------------------------------------------
--------------
// Server response info
```

```
inetHTTPAttrRspSize, // (R) UInt32, entire HTTP
Response size
        // including header and data
inetHTTPAttrResult, // (R) UInt32 (ctpErrXXX when
using CTP Proxy)
inetHTTPAttrErrDetail, // (R) UInt32 (server/proxy
err code when
        // using CTP Proxy)
inetHTTPAttrReason, // (R) Char[]
// Returned entity attributes
inetHTTPAttrContentLength, // (R) UInt32
inetHTTPAttrContentLengthUncompressed, // (R)
UInt32 (in bytes)
inetHTTPAttrContentLengthUntruncated, //(R) UInt32
inetHTTPAttrContentVersion, // (R) UInt32, actual
content version.
        // Represented as 2 low bytes. Lowest byte
is minor
        // version, next higher byte is major
version.
inetHTTPAttrContentCacheID, // (R) UInt32, cacheID
for this item
inetHTTPAttrReqSize // (R) UInt32 size of request
sent
} INetHTTPAttrEnum;
```

Value Descriptions

`inetHTTPAttrWhichPart`	An index to the part of the response data desired, if the response data is partitioned into chunks. Write-only.
`inetHTTPAttrIncHTTP`	A Boolean that, if set, causes HTTP header data to be included as part of the content when retrieving raw data. Applicable only when `inetSettingConvAlgorithm` is set to `ctpConvNone`. Write-only.
`inetHTTPAttrCheckMailHi`	High-order byte of ID for checking mail. Write-only.

`inetHTTPAttrCheckMailLo`	Low-order byte of ID for checking mail. Write-only.
`inetHTTPAttrReqContentVersion`	Desired content version. Represented as 2 low bytes. Lowest byte is minor version, next higher byte is major version. Write-only.
`inetHTTPAttrRspSize`	Size of entire HTTP (header and data). Read-only.
`inetHTTPAttrResult`	Transport protocol error code. Read-only.
`inetHTTPAttrErrDetail`	Server/proxy error code when using the Palm Web Clipping Proxy server. Read-only.
`inetHTTPAttrReason`	Transport protocol error message. Read-only.
`inetHTTPAttrContentLength`	Size of response data. Read-only.
`inetHTTPAttrContentLengthUncompressed`	Size of uncompressed response data. Read-only.
`inetHTTPAttrContentLengthUntruncated`	Total size of response data (it may have been truncated to less than this). Read-only.
`inetHTTPAttrContentVersion`	Actual content version. Represented as 2 low bytes. Lowest byte is minor version, next higher byte is major version. Read-only.
`inetHTTPAttrContentCacheID`	Cache ID for this item. Read-only.
`inetHTTPAttrReqSize`	Size of request sent. Read-only.

INetSchemeEnum

The `INetSchemeEnum` enum specifies a protocol (http, https, etc.) used by a socket. Specify one of these enumerated types for the `INetSockSettingScheme` socket setting and for the `scheme` parameter to the `INetLibSockOpen` call.

```
typedef enum {
inetSchemeUnknown = -1,
inetSchemeDefault = 0,

inetSchemeHTTP, // http:
inetSchemeHTTPS, // https:
inetSchemeFTP, // ftp:
inetSchemeGopher, // gopher:
inetSchemeFile, // file:
inetSchemeNews, // news:
inetSchemeMailTo, // mailto:
inetSchemePalm, // palm:
inetSchemePalmCall, // palmcall:
inetSchemeMail, // not applicable to URLs, but
used
                //  for the INetLibSockOpen call
when
                //  creating a socket for mail IO
inetSchemeMac, // mac: - Mac file system HTML

inetSchemeFirst = inetSchemeHTTP, // first one
inetSchemeLast = inetSchemeMail // last one
} INetSchemeEnum;
```

Value Descriptions

`inetSchemeHTTP`	Use the HTTP protocol.
`inetSchemeHTTPS`	Use the HTTPS protocol (for a secure connection).
`inetSchemeFTP`	Use the FTP protocol. Not implemented.
`inetSchemeGopher`	Use the Gopher protocol. Not implemented.
`inetSchemeFile`	Launch local PQA file
`inetSchemeNews`	Use the News protocol. Not implemented.
`inetSchemeMailTo`	Launch the local messaging application, passing a "to" address.
`inetSchemePalm`	Launches a local application database. The URL is expected to be in the form cccc.tttt, where cccc is a four character creator name and tttt is a four character database type. This pair of strings is used to identify an application database to receive the launch message via a call to `SysUIAppSwitch`.
`inetSchemePalmCall`	Launches a local application database. The URL is expected to be in the form cccc.tttt, where cccc is a four character creator name and tttt is a four character database type. This pair of strings is used to identify an application database to receive the launch message via a call to `SysAppLaunch`.
`inetSchemeMail`	Creates a socket for mail I/O.
`inetSchemeMac`	Handles opening Mac OS file system HTML URLs. For use by the Simulator only.

INetSettingEnum

The `INetSettingEnum` enum specifies a setting to be returned or set by the `INetLibSettingGet` or `INetLibSettingSet` calls.

```
typedef enum {
inetSettingCacheSize, // (RW) UInt32, max size of
cache
```

```
inetSettingCacheRef, // (R) DmOpenRef, ref of
cache DB
inetSettingNetLibConfig, // (RW) UInt32, NetLib
config to use
inetSettingRadioID, // (R) UInt32[2], the 64-bit
radio ID
inetSettingBaseStationID, // (R)  UInt32, the
radio base station Id
inetSettingMaxRspSize, // (W) UInt32 (in bytes)
inetSettingConvAlgorithm, // (W) UInt32
(CTPConvEnum)
inetSettingContentWidth, // (W) UInt32 (in pixels)
inetSettingContentVersion, // (W) UInt32, content
version (encoder
                            // version)
inetSettingNoPersonalInfo, // (RW) UInt32, send no
deviceID/zipcode
inetSettingUserName,
inetSettingLast
} INetSettingEnum;
```

Value Descriptions

`inetSettingCacheSize`	Maximum size of cache (in bytes).
`inetSettingCacheRef`	`DmOpenRef`, reference to cache database. Read-only.
`inetSettingNetLibConfig`	The index of the net library network configuration to use. This value is saved as part of the preferences for each Internet library configuration. A value of 0 means to use the current configuration.
`inetSettingRadioID`	64-bit radio ID. Read-only. Used for wireless connections only.
`inetSettingBaseStationID`	Radio base station ID. Read-only. Used for wireless connections only.
`inetSettingMaxRspSize`	Maximum response size, in bytes. The default is 1024 bytes. Write-only.

`inetSettingConvAlgorithm`	Content conversion desired. Write-only. Possible values include: `ctpConvCML` (use 5-bit compression scheme), `ctpConvCML8Bit` (use 5-bit compression scheme, but in 8-bit form for debugging), `ctpConvCMLLZ77` (use LZ77 compression scheme), `ctpConvNone` (no conversion; data is returned in native format)
`inetSettingContentWidth`	Width of the display for content. The default setting is 160 (pixels). Write-only.
`inetSettingContentVersion`	Content version (encoder version). Write-only. This setting is used to let the server know what encoder version it should use to encode content sent to the Palm client. Normally you don't need to set this value as it is initialized by `INetLibOpen`. The default encoder version is 0x8001.
`inetSettingNoPersonalInfo`	Send no device ID or zipcode information to the proxy server. This value is saved as part of the preferences for each Internet library configuration.
`inetSettingUserName`	Not applicable.

INetSockSettingEnum

The `INetSockSettingEnum` enum specifies a socket setting to be returned or set by the `INetLibSockSettingGet` or `INetLibSockSettingSet` calls.

```
typedef enum {
inetSockSettingScheme, // (R) UInt32,
INetSchemeEnum
inetSockSettingSockContext, // (RW) UInt32,
inetSockSettingCompressionType, // (R) Char[]
inetSockSettingCompressionTypeID, // (R) UInt32
                                  //
(INetCompressionTypeEnum)
inetSockSettingContentType, // (R) Char[]
```

```
inetSockSettingContentTypeID, // (R) UInt32
(INetContentTypeEnum)
inetSockSettingData, // (R) UInt32, pointer to
data
inetSockSettingDataHandle,// (R) UInt32, handle to
data
inetSockSettingDataOffset,// (R) UInt32, offset to
data from handle
inetSockSettingTitle, // (W) Char[]
inetSockSettingURL, // (R) Char[]
inetSockSettingIndexURL, // (RW) Char[]
inetSockSettingFlags, // (RW) UInt16, one or more
of
                      // inetOpenURLFlagXXX flags
inetSockSettingReadTimeout, // (RW) UInt32, read
timeout in ticks
inetSockSettingContentVersion,// (R) UInt32,
content version number
inetSockSettingLast
} INetSockSettingEnum;
```

Value Descriptions

`inetSockSettingScheme`	Requested scheme; one of the `INetSchemeEnum` values. Read-only.
`inetSockSettingSockContext`	Not used.
`inetSockSettingCompressionType`	Name of requested compression type. Read-only.
`inetSockSettingCompressionTypeID`	Requested compression type; one of the `INetCompressionTypeEnum` values. Read-only.
`inetSockSettingContentType`	String containing the MIME type of the content. Used only on received raw data. Read-only.
`inetSockSettingContentTypeID`	Content type of socket data; one of the `INetContentTypeEnum` values. Read-only.

`inetSockSettingData`	Pointer to socket data. Read-only.
`inetSockSettingDataHandle`	Handle to socket data. Read-only.
`inetSockSettingDataOffset`	Offset to socket data from handle. Read-only.
`inetSockSettingTitle`	Web page title. This value is written to the cache (and Clipper uses it later in a history list of cache entries). Write-only.
`inetSockSettingURL`	URL of requested data. Read-only.
`inetSockSettingIndexURL`	Index (or master) URL of requested data (for cache indexing). This is the topmost web page in a group of hierarchical pages; it serves to group the pages together and to filter cache list results. Clipper sets this to the URL of the active PQA, for all pages linked from the PQA.
`inetSockSettingFlags`	URL request flags; one or more of `inetOpenURLFlag`... flags (see URL Open Constants).
`inetSockSettingReadTimeout`	The default timeout value for reads when the application uses the event mechanism. The time since last receiving data from a socket is monitored and a timeout error status event is returned from `INetLibGetEvent` if the timeout is exceeded.
`inetSockSettingContentVersion`	Content version number. Read-only.

INetStatusEnum

The `INetStatusEnum` enum specifies the status of the socket. The status is returned in the `inetSockStatusChangeEvent` event structure and by the call `INetLibSockStatus`.

```
typedef enum {
inetStatusNew, // just opened
```

```
inetStatusResolvingName, // looking up host
address
inetStatusNameResolved, // found host address
inetStatusConnecting, // connecting to host
inetStatusConnected, // connected to host
inetStatusSendingRequest, // sending request
inetStatusWaitingForResponse, // waiting for
response
inetStatusReceivingResponse, // receiving response
inetStatusResponseReceived, // response received
inetStatusClosingConnection, // closing connection
inetStatusClosed, // closed
inetStatusAcquiringNetwork, // network temporarily
                            // unreachable; socket
on hold
inetStatusPrvInvalid = 30 // internal value, not
returned by
                            // INetMgr. Should be
last.
} INetStatusEnum;
```

Value Descriptions

`inetStatusNew`	Just opened
`inetStatusResolvingName`	Looking up host address
`inetStatusNameResolved`	Found host address
`inetStatusConnecting`	Connecting to host
`inetStatusConnected`	Connected to host
`inetStatusSendingRequest`	Sending request
`inetStatusWaitingForResponse`	Waiting for response
`inetStatusReceivingResponse`	Receiving response
`inetStatusResponseReceived`	Response received
`inetStatusClosingConnection`	Closing connection
`inetStatusClosed`	Connection closed

`inetStatusAcquiringNetwork`	Network temporarily unreachable; socket on hold
`inetStatusPrvInvalid`	Not used

Internet Library Constants

Configuration Aliases

The constants listed here specify Internet library network configuration alias names. Most of the Internet library API requires a configuration index rather than a name. Use `INetLibConfigIndexFromName` to obtain the alias's index from the name. For more information, see `INetConfigNameType`.

The following aliases are defined for configuration names:

Alias	Name string	Description
`inetCfgNameDefault`	".Default"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.
`inetCfgNameDefWireline`	".DefWireline"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.
`inetCfgNameDefWireless`	".DefWireless"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.

Alias	Name string	Description
`inetCfgNameCTPDefault`	".CTPDefault"	Initially points to either ".CTPWireless" (on Palm VII® units) or ".CTPWireline" (on all other units). On the Palm VII unit, the Clipper application uses this configuration.
`inetCfgNameCTPWireline`	".CTPWireline"	Initially points to a wireline configuration that uses the Palm Web Clipping Proxy server.
`inetCfgNameCTPWireless`	".CTPWireless"	Initially points to a wireless configuration that uses the Palm.Net™ wireless system and the Palm Web Clipping Proxy server.

URL Info Constants

The `inetURLInfoFlag`... constants convey information about a URL and are returned by the function `INetLibURLGetInfo`.

Constant	Value	Description
inetURLInfoFlagIsSecure	0x0001	URL was encrypted.
inetURLInfoFlagIsRemote	0x0002	URL was retrieved from the net.
inetURLInfoFlagIsInCache	0x0004	URL is stored in the cache.

URL Open Constants

The `inetOpenURLFlag`... constants control how the `INetLibURLOpen` call operates with respect to caching and encryption. These flags are also used for the `inetSockSettingFlags` socket setting.

Constant	Value	Description
`inetOpenURLFlagLookInCache`	0x0001	Read data from the cache, if available.
`inetOpenURLFlagKeepInCache`	0x0002	Store the item in the cache, overwriting any other entries with an equivalent URL.
`inetOpenURLFlagForceEncOn`	0x0008	Use encryption even if scheme does not desire it.
`inetOpenURLFlagForceEncOff`	0x0010	Do not use encryption even if scheme desires it.

Internet Library Functions

INetLibCacheGetObject

Purpose Returns information about an entry in the cache database, including a handle to the record. Either the URL or the unique ID can be used to find the cache entry.

Prototype

```
Err INetLibCacheGetObject (UInt16 libRefnum,
MemHandle clientParamH, UInt8 * urlTextP,
UInt32 uniqueID, INetCacheInfoPtr cacheInfoP)
```

Parameters

`-> libRefnum` Refnum of the Internet library.

`-> clientParamH` Inet handle allocated by `INetLibOpen`.

`-> urlTextP` Pointer to URL text string to find. If this parameter is `NULL`, then `uniqueID` is used to find the entry.

`-> uniqueID` Unique ID of the cache entry to find. This value can be obtained by calling `INetLibCacheList`. This parameter is ignored if `urlTextP` is specified.

`<- INetCacheInfoPtr`
Pointer to a structure where information about the cache entry is returned. See the Comments section for details.

Result

0	No error
`inetErrParamsInvalid`	One or more of the parameters are invalid.

Comments

The `INetCacheInfoPtr` type returned from this function is defined as a pointer to an `INetCacheInfoType` structure, which has the following definition:

```
typedef struct {
MemHandle recordH; // handle to the cache
record
INetContentTypeEnum contentType;
INetCompressionTypeEnum encodingType;
UInt32 uncompressedDataSize;
UInt8 flags; // unused
UInt8 reserved;
UInt16 dataOffset; // offset to content
UInt16 dataLength; // size of content
UInt16 urlOffset; // offset to URL
UInt32 viewTime; // time last viewed
UInt32 createTime; // time entry was created
UInt16 murlOffset; // offset to master URL
} INetCacheInfoType, *INetCacheInfoPtr;
```

Compatibility

Implemented only if Wireless Internet Feature Set is present.

INetLibCacheList

Purpose

Returns an item from the cache list, based on its URL and index in the list.

Prototype

```
Err INetLibCacheList (UInt16 libRefnum,
MemHandle inetH, UInt8 * cacheIndexURLP,
UInt16 * indexP, UInt32 * uidP,
INetCacheEntryP cacheP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.
`-> cacheIndexURLP`	Pointer to a master URL string. Cache entries indexed with this master URL are returned. Clipper sets the master URL of a cache page to the URL of the active PQA, so all pages linked from the PQA have the same master URL.
`<-> indexP`	Pointer to the index of the entry. Specify an index to find entries at this index or higher in the list. Specify `NULL` to search from the beginning, the first time you call this function. The index of the entry following the one found is returned on exit.
`<- uidP`	Pointer to a long value where the unique ID of the found cache entry is returned.
`<- cacheP`	Pointer to a structure where information about the found cache entry is returned. See the Comments section for details.

Result

0	No error
`inetErrTypeNotCached`	Cache entry under requested index not found

`inetErrParamsInvalid`	The `cacheIndexURLP` parameter is `NULL`
`inetErrCacheInvalid`	The cache database doesn't exist

Comments

This function first sorts the list of cache entries by URL. Then it returns in `uidP` the unique ID of the first cache entry with an index equal to or greater than `indexP`. The `indexP` value is updated to point to the next entry upon return.

To generate a complete list of cache entries having the same master URL (as for a history list), call this function repeatedly, always specifying the updated `index`, until it returns the error `inetErrTypeNotCached`.

Note that a URL can exist multiple times in the Clipper cache database, thus the need for the `uidP` value.

The `INetCacheEntryP` type returned from this function is defined as a pointer to an `INetCacheEntryType` structure, which has the following definition:

```
typedef struct {
UInt8 * urlP; // ptr to URL string
UInt16 urlLen; // length of URL string
UInt8 * titleP; // ptr to title string
UInt16 titleLen; // length of title string
UInt32 lastViewed; // time last viewed
                   // seconds since 1/1/1904
UInt32 firstViewed; // time first viewed
                   // seconds since 1/1/1904
} INetCacheEntryType, *INetCacheEntryP;
```

Compatibility

Implemented only if Wireless Internet Feature Set is present.

INetLibCheckAntennaState

Purpose Checks the antenna state and displays a dialog asking the user to raise it if it is down.

Prototype

```
Err INetLibCheckAntennaState(UInt16 Refnum)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.

Result

0	The user raised the antenna.
`netErrUserCancel`	The user closed the dialog by tapping Cancel.

This call can also return data manager errors if it fails internally.

Comments Applications don't need to check the antenna state by using this call. If an application opens the Internet library, the Internet library checks the antenna state when needed and displays the dialog to prompt the user to raise the antenna.

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibClose

Purpose Closes up and frees an inet handle. Closes or decrements the open count of the net library.

Prototype

```
Err INetLibClose (UInt16 libRefnum,
MemHandle inetH)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.

Result

0	No error

Comments This call must be made by an application when it's done with the Internet library. It closes any Internet sockets open by the application, disposes the memory referenced by the given inet handle, and calls `NetLibClose`, if necessary, to close the net Library or decrement its open count.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibOpen`

INetLibConfigAliasGet

Purpose Determines to which configuration a built-in alias points.

Prototype

```
Err INetLibConfigAliasGet (UInt16 refNum,
UInt16 aliasIndex, UInt16 * indexP,
Boolean * isAnotherAliasP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> aliasIndex`	Index of alias configuration to query. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by `INetLibConfigList`, or the index returned by `INetLibConfigIndexFromName`.
`<- indexP`	Pointer where the index of the configuration pointed to by `aliasIndex` is returned. 0 is returned if `aliasIndex` does not point to another configuration.

`<- isAnotherAliasP`
If `*indexP` is the index of another alias configuration, this Boolean is set to `true`.

Result

0	No error
`inetErrParamsInvalid`	`aliasIndex` is not valid
`inetErrConfigNotAlias`	`aliasIndex` is not an alias configuration

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibConfigAliasSet`

INetLibConfigAliasSet

Purpose Points any of the built-in aliases (".DefWireline", ".DefWireless", etc.) to a given defined configuration.

Prototype

```
Err INetLibConfigAliasSet (UInt16 refNum,
UInt16 configIndex, UInt16 aliasToIndex)
```

Parameters

`-> libRefnum` Refnum of the Internet library.

`-> configIndex` Index of configuration to set. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by `INetLibConfigList`, or the index returned by `INetLibConfigIndexFromName`.

`-> aliasToIndex`
Index of configuration that the alias identified by `configIndex` is to point to. Specify 0 to remove an existing alias assignment.

Result

0	No error
`inetErrConfigNotAlias`	`configIndex` is not an alias configuration
`inetErrParamsInvalid`	`configIndex` or `aliasToIndex` is not valid
`inetErrConfigCantPointToAlias`	Alias doesn't point to a real entry

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibConfigAliasGet`

INetLibConfigDelete

Purpose Deletes a configuration.

Prototype

```
Err INetLibConfigDelete (UInt16 refNum,
UInt16 index)
```

Parameters

`-> refnum` Refnum of the Internet library.

`-> index`	Index of configuration to delete. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by `INetLibConfigList`, or the index returned by `INetLibConfigIndexFromName`.

Result

0	No error
`inetErrParamsInvalid`	Index not valid
`inetErrConfigCantDelete`	Attempted to delete an alias configuration

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibConfigIndexFromName`, `INetLibConfigList`

INetLibConfigIndexFromName

Purpose Returns the index of a configuration given it's name.

Prototype

```
Err INetLibConfigIndexFromName (UInt16 refNum,
INetConfigNamePtr nameP, UInt16 * indexP)
```

Parameters

`-> refnum`	Refnum of the Internet library.
`-> nameP`	Pointer to an `INetConfigNameType` structure that names the configuration whose index you want to get.

`<- indexP`	Pointer where the index of the configuration identified in `nameP` is returned.

Result

0	No error
`inetErrConfigNotFound`	Could not find requested configuration name

Comments If you name an alias, this routine returns the index of the alias entry, not the configuration the alias points to. This way the alias can be pointed to a different configuration.

Applications should store the index of the configuration they're using, rather than the name, so that they won't be confused if the user edits the name of the configuration from the Preferences panel.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibConfigList`

INetLibConfigList

Purpose Returns an array containing a list of the available Internet library network configurations.

Prototype

```
Err INetLibConfigList (UInt16 refNum,
INetConfigNameType nameArray[],
UInt16 * arrayEntriesP)
```

Parameters

`-> refnum`	Refnum of the Internet library.
`-> nameArray`	Pointer to an array of `INetConfigNameType` structs that is to be filled in by this routine.

`<-> arrayEntriesP`
On entry, a pointer to the number of entries available in `nameArray`; on exit, a pointer to the total number of entries in the system (which could exceed the size of the array you pass in).

Result

0	No error

Comments This routine can be used to obtain a list of available configurations for selection by the user.

Note that the built-in alias configurations, which start with a period, should not be displayed to the user as selectable choices. They are designed for internal use by applications that need a predetermined type of service (like ".CTPWireless" for PQA applications.) Also, any configurations that start with an underscore, like "_CTPRAM", should not be displayed. These typically are configurations created by the Internet library for internal use and cannot be edited using the Network preferences panel.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibConfigMakeActive`

INetLibConfigMakeActive

Purpose Makes the given configuration active without having to close and reopen the Internet library by using `INetLibOpen`.

Prototype

```
Err INetLibConfigMakeActive (UInt16 refNum,
MemHandle inetH, UInt16 configIndex)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.

`-> configIndex`	Index of configuration to activate. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by `INetLibConfigList`, or the index returned by `INetLibConfigIndexFromName`.

Result

0	No error

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibConfigSaveAs`, `INetLibConfigList`, `INetLibConfigIndexFromName`

INetLibConfigRename

Purpose Renames a configuration.

Prototype

```
Err INetLibConfigRename (UInt16 refNum,
UInt16 index, INetConfigNamePtr newNameP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> index`	Index of configuration to rename. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by `INetLibConfigList`, or the index returned by `INetLibConfigIndexFromName`.

`-> newNameP`	Pointer to an `INetConfigNameType` structure holding the new name of the configuration. The name cannot start with a period or an underscore.

Result

0	No error
`inetErrConfigBadName`	Trying to save as an alias (beginning with ".") or as a built-in configuration (beginning with "_").
`inetErrParamsInvalid`	Invalid index
`inetErrConfigCantDelete`	Configuration to be renamed is either an alias or a built-in configuration

Comments After renaming, the configuration index stays the same so that applications that are set up to use that configuration will still work correctly. Note that built-in configuration aliases (ones that start with a period) cannot be renamed.

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibConfigSaveAs

Purpose Saves the current network configuration settings under the given name.

Prototype

```
Err INetLibConfigSaveAs (UInt16 refNum,
MemHandle inetH, INetConfigNamePtr nameP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.

`-> nameP`	Pointer to an `INetConfigNameType` structure holding the name of the configuration. The name cannot start with a period or an underscore.

Result

0	No error
`inetErrConfigBadName`	Trying to save as an alias (beginning with ".") or as a built-in configuration (beginning with "_").
`inetErrConfigTooMany`	The internal configurations table is full. No more entries can be stored.

Comments If the configuration name specified already exists, it is replaced with the new settings.

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibGetEvent

Purpose A replacement for `EvtGetEvent` that informs an application of status changes to Internet sockets as well as user interface events.

Prototype

```
void INetLibGetEvent (UInt16  libRefnum,
MemHandle inetH, INetEventType* eventP,
Int32 timeout)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`, or `NULL`.
`<-> eventP`	The event structure is returned via this pointer.

`-> timeout`	Timeout in ticks. Specify `evtWaitForever` to wait forever.

Result

0	No error

Comments This call is designed to replace `EvtGetEvent` in applications which use the Internet library. For convenience, if `inetH` is `NULL`, `INetLibGetEvent` is equivalent to `EvtGetEvent`.

`INetLibGetEvent` returns two additional events besides those returned by `EvtGetEvent`: `inetSockReadyEvent` and `inetSockStatusChangeEvent`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibSockStatus`, `INetLibURLOpen`, `INetLibSockOpen`, `INetLibSockRead`

INetLibOpen

Purpose Creates a new application inet handle structure. Opens or increments the open count of the net library.

Prototype

```
Err INetLibOpen (UInt16 libRefnum, UInt16 config,
UInt32 flags, DmOpenRef cacheRef,
UInt32 cacheSize, MemHandle* inetHP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library. Pass the value "INet.lib" to `SysLibFind` to return this refnum.
`-> config`	Indicates the type of network service desired by the application. Returned by `INetLibConfigIndexFromName`.
`-> flags`	Currently unused; set to 0.

`-> cacheRef`	Document cache database reference. Obtain this by using one of the `DmOpenDatabase`... calls. Pass `NULL` if you don't want to use a cache.
`-> cacheSize`	Maximum size of the document cache (in bytes). This is ignored if you pass `NULL` for `cacheRef`.
`<- inetHP`	Pointer to a handle variable.

Result

0	No error
`inetErrTooManyClients`	Too many clients opened already
`inetErrIncompatibleInterface`	The net library is already open with an incompatible interface

Comments

This call must be made by an application before it can use any other Internet library calls. This call opens the Internet library and returns a pointer to an inet handle, which is then passed to subsequent calls to the Internet library. Every application that opens the Internet library gets its own unique inet handle.

When an application is done using the Internet library, it must call `INetLibClose`, which closes both the Internet library and the net library, if necessary.

Compatibility

Implemented only if Wireless Internet Feature Set is present.

See Also

`INetLibClose`, `INetLibConfigIndexFromName`

INetLibSettingGet

Purpose Retrieves current settings for an inet handle.

Prototype

```
Err INetLibSettingGet (UInt16 libRefnum,
MemHandle inetH, UInt16 /*INetSettingEnum */
 setting, void * bufP, UInt16 * bufLenP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.
`-> setting`	The setting to get. Specify one of the `INetSettingEnum` enumerated types.
`<- bufP`	Pointer to buffer where the return value is to be put.
`<-> bufLenP`	Size of `bufP` on entry. Size of setting value on exit.

Result

0	No error
`inetErrParamsInvalid`	Invalid setting requested
`inetErrSettingSizeInvalid`	`*bufLenP` is the incorrect size for the requested setting

Comments This call can be used to retrieve the current settings of an inet handle. Some settings have default values that are stored in the system preferences database; see `INetSettingEnum` for details.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibOpen`, `INetLibSettingSet`, `INetSettingEnum`

INetLibSettingSet

Purpose Changes a setting for an inet handle.

Prototype

```
Err INetLibSettingSet (UInt16 libRefnum,
MemHandle inetH, UInt16 /*INetSettingEnum*/
 setting, void * bufP, UInt16 * bufLen)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.
`-> setting`	The setting to set. Specify one of the `INetSettingEnum` enumerated types.
`-> bufP`	Pointer to the new setting value.
`-> bufLen`	Size of the value in `bufP`.

Result

0	No error
`inetErrParamsInvalid`	Invalid setting specified
`inetErrSettingSizeInvalid`	`bufLen` is the incorrect size for the specified setting

Comments Any changes made to the settings last only as long as the `inetH` is around (until `INetLibClose` is called) and do not affect other applications that might be using the Internet library.

An important note is that settings made through this call essentially change the default values for any sockets subsequently created through `INetLibURLOpen` or `INetLibSockOpen`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibSettingGet`, `INetSettingEnum`

INetLibSockClose

Purpose Closes an inet socket handle.

Prototype

```
Err INetLibSockClose (UInt16 libRefnum,
MemHandle socketH)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> socketH`	Handle of the socket to close.

Result

0	No error

Comments This call closes down and releases all memory associated with a socket created by `INetLibSockOpen` or `INetLibURLOpen`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibOpen`, `INetLibSockOpen`, `INetLibURLOpen`

INetLibSockConnect

Purpose Establishes a connection with a remote host.

Prototype

```
Err INetLibSockConnect (UInt16 libRefnum,
MemHandle sockH, UInt8 * hostnameP, UInt16 port,
Int32 timeout)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> sockH`	Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket to connect.
`-> hostnameP`	Pointer to host name string; can be dotted decimal text string.
`-> port`	Port number, or 0 for default port.

`-> timeout`	Timeout in ticks; -1 means wait forever.

Result

0	No error

Comments

This call associates a remote host name and port number with a socket and, depending on the socket protocol, initiates a connection with that remote host.

This call may return immediately before actually finishing the connect. The application can simply go ahead and submit additional calls such as `INetLibSockRead`, or it may wait for the connect to complete by either polling `INetLibSockStatus` until the socket status is `inetStatusConnected` (not recommended), or by waiting for an `inetSockStatusChangeEvent` event from `INetLibGetEvent` and checking the status then (preferred).

Compatibility

Implemented only if Wireless Internet Feature Set is present.

See Also

`INetLibSockOpen`, `INetLibSockStatus`, `INetLibGetEvent`

INetLibSockHTTPAttrGet

Purpose

Queries HTTP request header formed by the local host, or the response header information returned by a remote host.

Prototype

```
Err INetLibSockHTTPAttrGet (UInt16 libRefnum,
MemHandle sockH, UInt16 /*inetHTTPAttrEnum*/ attr,
UInt16 attrIndex, void * bufP, UInt32 * bufLenP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> sockH`	Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket.
`-> attr`	The attribute to get. Specify one of the `INetHTTPAttrEnum` values.
`-> attrIndex`	The attribute index (if any). Currently unused.

`<- bufP`	Pointer to the address where the result is returned.
`<-> bufLenP`	Pointer to the size of `bufP` on entry; size of returned value on exit.

Result

0	No error
`inetErrSettingNotImplemented`	Invalid setting specified
`inetErrSettingSizeInvalid`	`bufLen` is the incorrect size for the specified setting

Comments This call queries either the request header formed by `INetLibSockHTTPReqCreate` and `INetLibSockHTTPAttrSet`, or the response header returned by the remote host.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibSockHTTPReqCreate`

INetLibSockHTTPAttrSet

Purpose Adds additional HTTP request headers to an HTTP request in a socket.

Prototype

```
Err INetLibSockHTTPAttrSet (UInt16 libRefnum,
MemHandle sockH, UInt16 /*inetHTTPAttrEnum*/ attr,
UInt16 attrIndex, UInt8 * bufP, UInt16 bufLen,
UInt16 flags)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> sockH`	Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket.
`-> attr`	The attribute to set. Specify one of the `INetHTTPAttrEnum` values.

`-> attrIndex`	The attribute index (if any). Currently unused.
`-> bufP`	Pointer to additional header text to add.
`-> bufLen`	Length of `bufP`.
`-> flags`	Flags that control the addition of new headers. Currently unused.

Result

0	No error
`inetErrSettingNotImplemented`	Invalid setting specified
`inetErrSettingSizeInvalid`	`bufLen` is the incorrect size for the specified setting

Comments This call modifies attributes of an HTTP request formed by `INetLibSockHTTPReqCreate`. Generally, attributes are set only before calling `INetLibSockHTTPReqSend`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibSockHTTPReqCreate`, `INetLibSockHTTPReqSend`

INetLibSockHTTPReqCreate

Purpose Forms an HTTP request for the socket.

Prototype

```
Err INetLibSockHTTPReqCreate (UInt16 libRefnum,
MemHandle sockH, UInt8 * verbP, UInt8 * resNameP,
UInt8 * refererP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> sockH`	Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket.
`-> verbP`	Pointer to a string holding an HTTP verb; most likely "GET".

`-> resNameP`	Pointer to a string holding the name of the resource to get or put.
`-> refererP`	Pointer to a string holding the name of the referring URL, or `NULL` if none.

Result

0	No error
`inetErrParamsInvalid`	Not an HTTP socket

Comments This call forms an HTTP request for the socket. The request is not actually sent to the remote host until `INetLibSockHTTPReqSend` is called. After this call, the application can add additional HTTP request headers using `INetLibSockHTTPAttrSet`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibSockHTTPAttrSet`, `INetLibSockHTTPReqSend`

INetLibSockHTTPReqSend

Purpose Sends an HTTP request to the remote host or looks for data in the cache.

Prototype

```
Err INetLibSockHTTPReqSend (UInt16 libRefnum,
MemHandle sockH, void * writeP, UInt32 writeLen,
Int32 timeout)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> sockH`	Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket.
`-> writeP`	Pointer to additional data to send after the request headers. Usually used for `POST` and `PUT` operations.
`-> writeLen`	Number of bytes in `writeP`.

`-> timeout` Timeout in ticks.

Result

0	No error
`inetErrRequestTooLong`	Request too big
`inetErrEncryptionNotAvail`	Encryption requested but not available

Comments

This call sends an HTTP request created by `INetLibSockHTTPReqCreate` and `INetLibSockHTTPAttrSet` to the remote host. If this is an `POST` or `PUT` operation, the data to write can be specified in `writeP`.

`INetLibSockHTTPReqSend` doesn't always do network I/O. If the proper socket flag is set, it checks first to see if the requested data is already in the cache. If it is, then a pointer to the cached data is stored in the socket and the socket status is updated to show that data is ready to be read. This will trigger an `inetSockReadyEvent` event.

The socket flag (`inetOpenURLFlagLookInCache`) that causes the cache to be checked first can be set via the `flags` parameter to `INetLibURLOpen` or by calling `INetLibSockSettingSet` with the appropriate setting (`inetSockSettingFlags`).

After sending the request, the application can wait for the response to arrive by either polling `INetLibSockStatus` until the `inputReady` boolean is set (not recommended), or by waiting for an `inetSockReadyEvent` event from `INetLibGetEvent` (preferred).

Compatibility

Implemented only if Wireless Internet Feature Set is present.

See Also

`INetLibSockHTTPReqCreate`, `INetLibSockHTTPAttrSet`, `INetLibGetEvent`

INetLibSockOpen

Purpose Creates and returns a new inet socket handle.

Prototype

```
Err INetLibSockOpen (UInt16 libRefnum,
MemHandle inetH, UInt16 /*INetSchemeEnum*/ scheme,
MemHandle* sockHP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.
`-> scheme`	The protocol scheme to use. Specify one of the `INetSchemeEnum` types.
`<- sockHP`	Pointer to the address where the socket handle is returned.

Result

0	No error
`inetErrTooManySockets`	Too many sockets open
`inetErrSchemeNotSupported`	Requested URL scheme not supported

Comments This call creates a new socket for the given scheme. No network I/O is performed. This is a relatively low level call that can be used in place of `INetLibURLOpen` when finer control over the socket settings is required.

Using `INetLibURLOpen`, an HTTP request can be handled with the simple sequence: `INetLibURLOpen`, `INetLibSockRead`, and `INetLibSockClose`. When using `INetLibSockOpen`, the same HTTP request would be handled by replacing the `INetLibURLOpen` call with the sequence: `INetLibSockOpen`, `INetLibSockSettingSet` (optional), `INetLibSockConnect`, `INetLibSockHTTPReqCreate`, `INetLibSockHTTPAttrSet` (optional), and `INetLibSockHTTPReqSend`.

The use of `INetLibSockOpen` allows an application finer control over the socket settings (by calling `INetLibSockSettingSet`)

and the HTTP request headers (by calling `INetLibSockHTTPAttrSet`).

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibOpen`, `INetLibURLOpen`, `INetLibSockRead`, `INetLibSockClose`, `INetLibSockSettingSet`, `INetLibSockHTTPAttrSet`

INetLibSockRead

Purpose Reads data from a socket.

Prototype

```
Err INetLibSockRead (UInt16 libRefnum,
MemHandle sockH, void * bufP, UInt32 reqBytes,
UInt32 * actBytesP, Int32 timeout)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> sockH`	Inet handle allocated by `INetLibOpen`.
`-> bufP`	Pointer to buffer where the data is placed.
`-> reqBytes`	Requested number of bytes.
`<- actBytesP`	Pointer to the actual number of bytes read.
`-> timeout`	Timeout in ticks; -1 means wait forever.

Result

0	No error

Comments This call attempts to read `reqBytes` bytes from the given socket. It returns the actual number of bytes read in `*actBytesP`. If the connection with the remote host has been closed, `*actBytesP` contains 0 on exit.

Note that it is normal for the actual bytes read to be less than the requested number of bytes. The application should be prepared to call this routine repeatedly until the desired number of bytes have been read or until `*actBytesP` contains 0, indicating the connection has been closed, or until an error is returned.

This call returns as much data as possible without blocking, however, if no data is available to be read, it does block until at least one byte is available.

Normally, applications will wait for an `inetSockReadyEvent` from `INetLibGetEvent` before calling `INetLibSockRead`. Alternatively, the application could call `INetLibSockStatus` to determine if the socket has any data ready (not recommended), or could simply rely on `INetLibSockRead` to block until at least one byte is available to read. If no data is available before the timeout expires, `inetErrReadTimeout` error is returned.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibURLOpen`, `INetLibSockOpen`, `INetLibSockStatus`, `INetLibSockClose`, `INetLibGetEvent`

INetLibSockSettingGet

Purpose Retrieves a socket setting.

Prototype

```
Err INetLibSockSettingGet (UInt16 libRefnum,
MemHandle socketH, UInt16 /*INetSockSettingEnum*/
 setting, void * bufP, UInt16 * bufLenP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> socketH`	Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket to get a setting from.
`-> setting`	The setting to get. Specify one of the `INetSockSettingEnum` values.
`<- bufP`	Pointer to buffer where the setting value is to be placed.

`<-> bufLenP`	Size of `bufP` on entry; size of returned value on exit.

Result

0	No error
`inetErrParamsInvalid`	Invalid setting requested
`inetErrSettingSizeInvalid`	`*bufLenP` is the incorrect size for the requested setting

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibSockSettingSet`

INetLibSockSettingSet

Purpose Changes a setting of a socket.

Prototype

```
Err INetLibSockSettingSet (UInt16 libRefnum,
MemHandle socketH, UInt16 /*INetSockSettingEnum*/
 setting, void * bufP, UInt16 bufLen)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> socketH`	Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket to set.
`-> setting`	The setting to set. Specify one of the `INetSockSettingEnum` values.
`-> bufP`	Pointer to buffer containing the new setting value.
`-> bufLen`	Size of new setting in `bufP`.

Result

0	No error

`inetErrSettingNotImplemented`	Invalid setting specified
`inetErrSettingSizeInvalid`	`bufLen` is the incorrect size for the setting

Comments This call can be use to override a general setting for a particular socket.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibSockSettingGet`

INetLibSockStatus

Purpose Retrieves the current status of a socket.

Prototype

```
Err INetLibSockStatus (UInt16 libRefnum,
MemHandle socketH, UInt16 * statusP,
Err* sockErrP, Boolean* inputReadyP,
Boolean* outputReadyP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> socketH`	Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket to get status on.
`<- statusP`	Pointer to the address where the status is returned. The status will be one of the `INetStatusEnum` values.
`<- sockErrP`	Currently unused.
`<- inputReadyP`	Pointer to a Boolean; `true` is returned if the socket has data available to read.

`<- outputReadyP`
Pointer to a Boolean; `true` is returned if the socket can accept data for writing.

Result

0	No error

Comments Most applications that use `INetLibGetEvent` will rarely need to use this call since socket status changes are returned in the event structure.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibURLOpen`, `INetLibSockOpen`, `INetLibSockRead`, `INetLibGetEvent`

INetLibURLCrack

Purpose Cracks a URL text string into its components.

Prototype

```
Err INetLibURLCrack (UInt16 libRefnum,
UInt8 * urlTextP, INetURLType* urlP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> urlTextP`	Pointer to URL text string.
`<-> urlP`	Pointer to address where the URL information block is to be returned.

Result

0	No error
`inetErrParamsInvalid`	`urlTextP` is `NULL` or empty, or `urlP` is `NULL`
`inetErrURLVersionInvalid`	`urlP` is wrong version

Comments If a pointer member of `urlP` is set to `NULL` on entry, then on exit it will point to the start of that component within the original `urlTextP` string; the associated member length is set to the length of that URL component. If a pointer member of `urlP` is not `NULL` on entry, then it must point to a buffer of sufficient size to hold the member data, and on exit the component string will be copied into this buffer and the associated member length will be updated with the actual size. Note that the returned strings are not `NULL` terminated, so the length values are important.

It's easiest to initialize the `InetURLType` block to zeros and let this function fill in all the information about the URL components.

The `InetURLType` block returned from this function has the following structure:

```
typedef struct {
UInt16 version; // 0, for future compatibility
UInt8 * schemeP; // ptr to scheme portion
UInt16 schemeLen; // size of scheme portion
UInt16 schemeEnum; // INetSchemeEnum; the
scheme
UInt8 * usernameP; // ptr to username portion
UInt16 usernameLen; // size of username
UInt8 * passwordP; // ptr to password portion
UInt16 passwordLen; // size of password
UInt8 * hostnameP; // ptr to host name portion
UInt16 hostnameLen; // size of host name
UInt16 port; // port number
UInt8 * pathP; // ptr to path portion
UInt16 pathLen; // size of path
UInt8 * paramP; // ptr to param (;param)
UInt16 paramLen; // size of param
UInt8 * queryP; // ptr to query (?query)
UInt16 queryLen; // size of query
UInt8 * fragP; // ptr to fragment (#frag)
UInt16 fragLen; // size of fragment
} INetURLType
```

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibURLGetInfo

Purpose

Returns information about a URL.

Prototype

```
Err INetLibURLGetInfo (UInt16 libRefnum,
MemHandle inetH, UInt8 * urlTextP,
INetURLInfoType* urlInfoP)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.
`-> urlTextP`	Pointer to URL text string.
`<-> urlInfoP`	Pointer to address where the URL information structure is to be returned.

Result

0	No error
`inetErrParamsInvalid`	`urlInfoP` is `NULL` or incorrect version

Comments

The `InetURLInfo` block returned from this function has the following structure:

```
typedef struct {
UInt16 version; // 0, for future compatibility
UInt16 flags; // flags word
UInt32 undefined; // reserved for future use
} INetURLInfo
```

The `flags` word can consist of some combination of these values:

```
inetURLInfoFlagIsSecure // URL was encrypted
inetURLInfoFlagIsRemote // URL was retrieved
from the net
inetURLInfoFlagIsInCache // URL is stored in
the cache
```

Compatibility

Implemented only if Wireless Internet Feature Set is present.

INetLibURLOpen

Purpose Accesses a URL on the Internet or in the cache.

Prototype

```
Err INetLibURLOpen (UInt16 libRefnum,
MemHandle inetH, UInt8 * urlP,
UInt8 * cacheIndexURLP, MemHandle* sockHP,
Int32 timeout, UInt16 flags)
```

Parameters

`-> libRefnum`	Refnum of the Internet library.
`-> inetH`	Inet handle allocated by `INetLibOpen`.
`-> urlP`	Pointer to string containing the URL to access.
`-> cacheIndexURLP`	Pointer to URL string under which the requested URL should be indexed in the cache. Specify `NULL` if you don't need to index the cache. If you are using the Clipper cache (not recommended), you must follow the Clipper convention, which is to pass the URL of the active PQA.
`<- sockHP`	Pointer to address where the socket handle is returned.
`-> timeout`	Timeout in ticks; -1 means wait forever.
`-> flags`	Flags indicating caching and encryption options desired. Specify zero, one, or more of the URL open flags (see URL Open Constants).

Result

0	No error
`inetErrParamsInvalid`	`urlP` is `NULL`

Comments This call sets up a connection to a resource on the Internet addressed by `urlP` and returns a socket handle. Note that if you specify that the cache should be searched first, and if the data is found in the cache, no network I/O occurs. The application can then read that socket resource through the `INetLibSockRead` call.

This call is a convenience routine that internally makes the following calls for http URLs: `INetLibSockOpen`, `INetLibSockConnect`, `INetLibSockHTTPReqCreate`, and `INetLibSockHTTPReqSend`.

This routine returns immediately before performing any required network I/O. It is then up to the caller to either block on `INetLibSockRead`, or to use `INetLibGetEvent` to model asynchronous operation. Using `INetLibGetEvent` is the preferred way of performing network I/O since it maximizes battery life and user-interface responsiveness.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also `INetLibSockOpen`, `INetLibSockConnect`, `INetLibSockRead`, `INetLibSockClose`

INetLibURLsAdd

Purpose Concatenates two URLs, resulting in one absolute URL.

Prototype

```
Err INetLibURLsAdd (UInt16 libRefnum,
Char * baseURLStr, Char * embeddedURLStr,
Char * resultURLStr, UInt16 * resultLenP)
```

Parameters

`-> libRefnum` Refnum of the Internet library.

`-> baseURLStr` Pointer to base URL string.

`-> embeddedURLStr` Pointer to URL text string to append.

`<-> resultURLStr` Pointer to resulting URL string.

`<-> resultLenP` Pointer to size of `resultURLStr` buffer on entry. On exit, pointer to resulting URL length (including `NULL` terminator).

Result

0	No error

Comments Used to append a URL fragment to a base URL, resulting in an absolute URL string that can be passed to `INetLibURLOpen` or other functions. This routine ensures that the resulting string conforms to the URL format.

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibWiCmd

Purpose Invokes a command that operates on the wireless indicator.

Prototype

```
Boolean INetLibWiCmd (UInt16 refNum, UInt16 /
*WiCmdEnum*/ cmd, int enableOrX, int y)
```

Parameters

`-> refNum`	Refnum of the Internet library.
`-> cmd`	The command to invoke. Specify one of the `WiCmdEnum` values (see Comments section).
`-> enableOrX`	If `cmd` is `wiCmdSetEnabled`, specify 1 to enable the wireless indicator or 0 to disable it. If `cmd` is `wiCmdSetLocation`, this specifies the x coordinate of the location.
`-> y`	The y coordinate of the location. Used only if `cmd` is `wiCmdSetLocation`.

Result If `cmd` is `wiCmdEnabled`, this function returns `true` if the wireless indicator is enabled or `false` if it is not. For other command types, the return value is undefined.

Comments The wireless indicator is a 19x13 pixel image on the screen to indicate the current wireless signal strength. This shows as 0 - 5 bars. If the application is in a non-modal window with a title bar, the preferred location for the indicator is at (140,1).

It automatically updates itself as long as you are calling `INetLibGetEvent`. It should be shown on screen while a wireless transaction is in progress. It may also be shown when the user has nothing useful to do next but initiate a wireless transaction, and there isn't much other useful information being displayed.

The `WiCmdEnum` enum specifies a command that operates on the wireless indicator in the user interface. The definition of this type is found in `WirelessIndicator.h` and is as follows:

```
typedef enum {
wiCmdInit =0,
wiCmdClear,
wiCmdSetEnabled,
wiCmdDraw,
wiCmdEnabled,
wiCmdSetLocation,
wiCmdErase
} WiCmdEnum;
```

Value Descriptions

`wiCmdInit`	Initializes the wireless indicator. You must invoke this command first, before using any of the others.
`wiCmdClear`	Applications shouldn't use this command. To erase the indicator, disable it by using `wiCmdSetEnabled` and passing 0 for `enableOrX`.
`wiCmdSetEnabled`	Enables or disables the wireless indicator.
`wiCmdDraw`	Redraws the wireless indicator using the latest data. Applications don't need to use this command since the indicator is redrawn automatically by `INetLibGetEvent`.
`wiCmdEnabled`	Returns a Boolean indicating if the wireless indicator is enabled.
`wiCmdSetLocation`	Sets the location for the wireless indicator on the screen.
`wiCmdErase`	Erases the wireless indicator. Applications shouldn't use this command. To erase the indicator, disable it by using `wiCmdSetEnabled` and passing 0 for `enableOrX`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

62

PalmOSGlue Library

This chapter describes the API provided in the link library PalmOSGlue (`PalmOSGlue.lib` or `libPalmOSGlue.a`).

You use PalmOSGlue if you want to use the international and text manager features described in the chapter "Localized Applications" on page 317 in the *Palm OS Programmer's Companion* and you want to maintain backward compatibility with earlier releases. If you link with PalmOSGlue, include the headers `DateGlue.h`, `FntGlue.h`, `TxtGlue.h`, `TsmGlue.h`, and `WinGlue.h` (in the `International` directory), and make calls as they are listed in this chapter, then your code will run regardless of whether the device's version of the operating system implements international support. The code in PalmOSGlue either uses the text manager or international manager on the ROM or, if the managers don't exist, executes a simple Latin equivalent of the function.

NOTE: PalmOSGlue is a link library, not a shared library. Linking with this library increases your application's code size. The amount by which your code size increases varies depending on the number of library functions you call. Use PalmOSGlue only on versions 2.0 and later of Palm OS®.

In addition to covering the text and international manager API, PalmOSGlue adds some functions that are not included in any version of the Palm OS. This chapter describes the functions that are unique to PalmOSGlue and provides a mapping of PalmOSGlue calls to calls that exist in later versions of Palm OS.

PalmOSGlue Functions

The following table shows the mapping between the functions declared in the glue headers and the international functions and

macros. To learn more about a glue function, click the link in the right column.

This table lists only those functions that map to a function that exists in newer versions of the OS. The functions that are exclusive to PalmOSGlue are not listed. They are described following this table.

Table 62.1 PalmOSGlue function mappings

This PalmOSGlue function...	...is identical to...
DateGlueToAscii	DateToAscii
DateGlueToDOWDMFormat	DateToDOWDMFormat
TsmGlueGetFepMode	TsmGetFepMode
TsmGlueSetFepMode	TsmSetFepMode
TxtGlueByteAttr	TxtByteAttr
TxtGlueCaselessCompare	TxtCaselessCompare
TxtGlueCharAttr	TxtCharAttr
TxtGlueCharBounds	TxtCharBounds
TxtGlueCharEncoding	TxtCharEncoding
TxtGlueCharIsAlNum	TxtCharIsAlNum
TxtGlueCharIsAlpha	TxtCharIsAlpha
TxtGlueCharIsCntrl	TxtCharIsCntrl
TxtGlueCharIsDelim	TxtCharIsDelim
TxtGlueCharIsDigit	TxtCharIsDigit
TxtGlueCharIsGraph	TxtCharIsGraph
TxtGlueCharIsHex	TxtCharIsHex
TxtGlueCharIsLower	TxtCharIsLower
TxtGlueCharIsPrint	TxtCharIsPrint
TxtGlueCharIsPunct	TxtCharIsPunct
TxtGlueCharIsSpace	TxtCharIsSpace

Table 62.1 PalmOSGlue function mappings *(continued)*

This PalmOSGlue function...	...is identical to...
TxtGlueCharIsUpper	TxtCharIsUpper
TxtGlueCharIsValid	TxtCharIsValid
TxtGlueCharSize	TxtCharSize
TxtGlueCharWidth	TxtCharWidth
TxtGlueCharXAttr	TxtCharXAttr
TxtGlueCompare	TxtCompare
TxtGlueEncodingName	TxtEncodingName
TxtGlueFindString	TxtFindString
TxtGlueGetChar	TxtGetChar
TxtGlueGetNextChar	TxtGetNextChar
TxtGlueGetPreviousChar	TxtGetPreviousChar
TxtGlueGetTruncation Offset	TxtGetTruncationOffset
TxtGlueMaxEncoding	TxtMaxEncoding
TxtGlueNextCharSize	TxtNextCharSize
TxtGlueParamString	TxtParamString
TxtGluePreviousCharSize	TxtPreviousCharSize
TxtGlueReplaceStr	TxtReplaceStr
TxtGlueSetNextChar	TxtSetNextChar
TxtGlueStrEncoding	TxtStrEncoding
TxtGlueTransliterate	TxtTransliterate
TxtGlueWordBounds	TxtWordBounds
WinGlueDrawChar	WinDrawChar
WinGlueDrawTruncChars	WinDrawTruncChars

FntGlueGetDefaultFontID

Purpose Return the font ID of a default font.

Prototype

```
FontID FntGlueGetDefaultFontID (
FontDefaultType inFontType)
```

Parameters

-> `inFontType` A `FontDefaultType` constant specifying one of the system default fonts. This value can be one of the following:

`defaultSystemFont`
The default font for the system.

`defaultLargeFont`
The default large font.

`defaultSmallFont`
The default small font.

`defaultBoldFont`
The default bold font.

Result Returns the ID of `inFontType`.

Comments Use this function whenever you need to obtain a font ID for one of the system default fonts. The default fonts (and thus, the IDs for the default fonts) vary depending on the system's locale. For example, Japanese systems have a different set of default fonts than systems using the Latin character encoding.

Use this function in place of the constants that specify the IDs of default fonts, as shown in the following table.

In place of this...	...use FntGlueGetDefaultFontID with this constant...
`stdFont`	`defaultSystemFont` (best for displaying text) or: `defaultSmallFont` (if you want a smaller font)
`largeFont`	`defaultLargeFont`

In place of this...	...use FntGlueGetDefaultFontID with this constant...
`largeBoldFont`	`defaultLargeFont`
`boldFont`	`defaultBoldFont`

Note that `defaultSystemFont` and `defaultSmallFont` might return the same font ID or different font IDs, depending on the system locale.

Compatibility Implemented only in the PalmOSGlue library.

See Also `FontSelect`, `FntGetFont`, `FntSetFont`

TxtGlueCharIsVirtual

Purpose Return whether a character is a virtual character or not.

Prototype

```
Boolean TxtGlueCharIsVirtual(UInt16 inModifiers,
WChar inChar)
```

Parameters

-> `inModifiers`	The value passed in the `modifiers` field of the `keyDownEvent`.
-> `inChar`	A character.

Result Returns `true` if the character `inChar` is a virtual character, `false` otherwise.

Comments Virtual characters are nondisplayable characters that trigger special events in the operating system, such as displaying low battery warnings or displaying the keyboard dialog. Virtual characters should never occur in any data and should never appear on the screen.

Starting in Palm OS 3.1, the command modifier bit is always set in the `keyDownEvent` for a virtual character because the range for virtual characters overlaps the range for “real” characters that

should appear on the screen. Earlier releases of the operating system did not always set the command modifier for virtual characters.

You can use this function to test whether a character is virtual or not. Pass the `chr` and `modifiers` fields exactly as you received them in the `keyDownEvent`, and this function performs the appropriate check based on the operating system version.

Compatibility Implemented only in the PalmOSGlue library.

TxtGlueGetHorizEllipsisChar

Purpose Return the horizontal ellipsis character.

Prototype

```
WChar TxtGlueGetHorizEllipsisChar (void)
```

Parameters none

Result Returns the character code for horizontal ellipsis.

Comments Version 3.1 of the Palm OS uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use `TxtGlueGetHorizEllipsisChar` to return the appropriate code for horizontal ellipsis regardless of which version of Palm OS your application is run on.

Compatibility Implemented only in the PalmOSGlue library.

See Also `ChrHorizEllipsis`, `TxtGlueGetNumericSpaceChar`

TxtGlueGetNumericSpaceChar

Purpose Return the numeric space character.

Prototype
```
WChar TxtGlueGetNumericSpaceChar (void)
```

Parameters none

Result Returns the character code for numeric space.

Comments Version 3.1 of the Palm OS uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use `TxtGlueGetNumericSpaceChar` to return the appropriate code for numeric space regardless of which version of Palm OS your application is run on.

Compatibility Implemented only in the PalmOSGlue library.

See Also `ChrNumericSpace`, `TxtGlueGetHorizEllipsisChar`

TxtGlueLowerChar

Purpose Convert a character to lowercase.

Prototype
```
WChar TxtGlueLowerChar (WChar inChar)
```

Parameters -> `inChar` A character.

Result Returns the character as a lowercase letter.

Comments This function can only handle characters in the ISO Latin 1 character encoding unless the International Feature Set is present.

Compatibility Implemented only in the PalmOSGlue library.

See Also `TxtGlueUpperChar`, `TxtGlueLowerStr`, `TxtGlueUpperStr`, `TxtGlueTransliterate`, `TxtTransliterate`, `StrToLower`

TxtGlueLowerStr

Purpose Convert a string to all lowercase letters.

Prototype

```
void TxtGlueLowerStr (Char* ioString,
UInt16 inMaxLength)
```

Parameters

<-> `ioString`	The string to be converted.
-> `inMaxLength`	The size of the buffer that contains the string, excluding the terminating `NULL` character.

Result Returns in `ioString` the input string with its letters converted to lowercase.

Comments On systems that use multi-byte character encodings, converting a string from uppercase to lowercase letters or vice versa may change the size of the string. For this reason, you should always check the size of the `ioString` after this call returns.

You must make sure that the parameter `ioString` points to the start of a a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

This function can only handle characters in the ISO Latin 1 character encoding unless the International Feature Set is present.

Compatibility Implemented only in the PalmOSGlue library.

See Also `TxtGlueUpperStr`, `TxtGlueLowerChar`, `TxtGlueUpperChar`, `StrToLower` `TxtGlueTransliterate`, `TxtTransliterate`

TxtGluePrepFindString

Purpose Set up for TxtFindString or FindStrInStr.

Prototype

```
void TxtGluePrepFindString (const Char* inSource,
CharPtr outDest, UInt16 inDstSize)
```

Parameters

`-> inSource`	Pointer to the text to be searched for. Must not be `NULL`.
`<- outDest`	The same text as in `inSource` but converted to a suitable format for searching. `outDest` must not be the same address as `inSource`.
`-> inDstSize`	The length in bytes of the area pointed to by `outDest`.

Result Returns in `outDest` an appropriately converted string.

Comments Use this function to normalize the string to search for before using `TxtGlueFindString`, `TxtFindString`, or `FindStrInStr` to perform a search that is internal to your application. If you use any of these three search routines in response to the sysAppLaunchCmdFind launch code, the string that the launch code passes in is already properly normalized for the search.

This function normalizes the string to be searched for. The method by which a search string is normalized varies depending on the version of Palm OS and the character encoding supported by the device.

Only `inDstSize` bytes of `inSource` are written to `outDest`. If necessary to prevent overflow of the destination buffer, not all of `inSource` is converted.

You must make sure that the parameter `inSource` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only in the PalmOSGlue library.

TxtGlueStripSpaces

Purpose Strip trailing and/or leading spaces from a string.

Prototype

```
Char* TxtGlueStripSpaces (Char* ioStr,
Boolean leading, Boolean trailing)
```

Parameters

<-> `ioStr`	Any string.
-> `leading`	If `true`, strip the leading spaces from the string.
-> `trailing`	If `true`, strip the trailing spaces from the string.

Result Returns `ioStr` with the specified spaces stripped from it. Note that this function both changes the `ioStr` buffer parameter and returns a pointer to it.

Comments You must make sure that the parameter `ioStr` points to the start of a a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only in the PalmOSGlue library.

TxtGlueUpperChar

Purpose Convert a character to uppercase.

Prototype

```
WChar TxtGlueUpperChar (WChar inChar)
```

Parameters

-> `inChar`	Any character.

Result Returns the character as an uppercase letter.

Comments This function can only handle characters in the ISO Latin 1 character encoding unless the International Feature Set is present.

Compatibility Implemented only in the PalmOSGlue library.

See Also `TxtGlueLowerChar`, `TxtGlueUpperStr` `TxtGlueLowerStr`, `TxtGlueTransliterate`, `TxtTransliterate` `StrToLower`

TxtGlueUpperStr

Purpose Convert a string to all uppercase letters.

Prototype

```
void TxtGlueUpperStr (Char* ioString,
UInt16 inMaxLength)
```

Parameters

<-> `ioString`	The string to be converted.
-> `inMaxLength`	The size of the buffer that contains the string, excluding the terminating `NULL` character.

Result Returns in `ioString` the input string with its letters converted to uppercase.

Comments On systems that use multi-byte character encodings, converting a string from uppercase to lowercase letters or vice versa may change the size of the string. For this reason, you should always check the size of the `ioString` after this call returns.

You must make sure that the parameter `ioString` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

This function can only handle characters in the ISO Latin 1 character encoding unless the International Feature Set is present.

Compatibility Implemented only in the PalmOSGlue library.

See Also `TxtGlueLowerStr`, `TxtGlueUpperChar`, `TxtGlueLowerChar`, `TxtGlueTransliterate`, `TxtTransliterate` `StrToLower`

A

System Use Only Functions

This appendix lists functions that are purposely undocumented because they are for system use only.

WARNING! System Use Only.

```
AbtShowAbout
AlmAlarmCallback
AlmCancelAll
AlmDisplayAlarm
AlmEnableNotification
AlmInit
AlmTimeChange
DmInit
DmResetRecordStates
ExgConnect
ExgGet
ExgInit
ExgNotifyReceive
EvtDequeueKeyEvent
EvtEnqueuePenPoint
EvtGetSysEvent
EvtInitialize
EvtSetKeyQueuePtr
EvtSetPenQueuePtr
EvtSysInit
FileReadLow
Find
FrmActiveState
FrmAddSpaceForObject
FtrInit
GrfFieldChange
```

```
GrfFree
GrfInit
INetLibSleep
INetLibSockMailAttrGet
INetLibSockMailAttrSet
INetLibSockMailQueryProgress
INetLibSockMailReqAdd
INetLibSockMailReqCreate
INetLibSockMailReqSend
INetLibWake
InsPtCheckBlink
InsPtInitialize
IntlInit
IrHandleEvent
IrWaitForEvent
MemCardFormat
MemChunkFree
MemChunkNew
MemHandleFlags
MemHandleLockCount
MemHandleOwner
MemHandleResetLock
MemHeapFreeByOwnerID
MemHeapInit
MemInit
MemInitHeapTable
MemKernelInit
MemPtrFlags
MemPtrOwner
MemPtrResetLock
MemSemaphoreRelease
MemSemaphoreReserve
MemStoreSetInfo
NetLibConfigAliasGet
NetLibConfigAliasSet
NetLibConfigDelete
NetLibConfigIndexFromName
NetLibConfigList
NetLibConfigMakeActive
NetLibConfigRename
```

```
NetLibConfigSaveAs
NetLibHandlePowerOff
NetLibOpenConfig
NetLibOpenIfCloseWait
NetLibSleep
NetLibWake
PenClose
PenGetRawPen
PenOpen
PenRawToScreen
PenScreenToRaw
PenSleep
PenWake
ScrCompressScanLine
ScrCopyRectangle
ScrDeCompressScanLine
ScrDrawChars
ScrDrawNotify
ScrInit
ScrLineRoutine
ScrRectangleRoutine
ScrScreenInfo
ScrSendUpdateArea
SerDbgAssureOpen
SerialMgrInstall
SerReceiveISP
SerReceiveWindowClose
SerReceiveWindowOpen
SerSetMapPort
SerSetWakeupHandler
SerSleep
SerWake
SlkProcessRPC
SlkSysPktDefaultResponse
SndInit
SndSetDefaultVolume
SrmSleep
SrmWake
SysAppStartup
SysAppExit
```

```
SysBatteryDialog
SysCardImageDeleted
SysCardImageInfo
SysColdBoot
SysDisableInts
SysDoze
SysEvGroupCreate
SysEvGroupRead
SysEvGroupSignal
SysEvGroupWait
SysInit
SysKernelInfo
SysLaunchConsole
SysLCDContrast
SysLibClose
SysLibInstall
SysLibOpen
SysLibSleep
SysLibTblEntry
SysLibWake
SysMailboxCreate
SysMailboxDelete
SysMailboxFlush
SysMailboxSend
SysMailboxWait
SysNewOwnerID
SysPowerOn
SysResSemaphoreCreate
SysResSemaphoreDelete
SysResSemaphoreRelease
SysResSemaphoreReserve
SysRestoreStatus
SysSemaphoreCreate
SysSemaphoreDelete
SysSemaphoreSet
SysSemaphoreSignal
SysSemaphoreWait
SysSetA5
SysSetPerformance
SysSleep
```

```
SysTaskCreate
SysTaskDelete
SysTaskID
SysTaskResume
SysTaskSuspend
SysTaskSwitching
SysTaskTrigger
SysTaskWait
SysTaskWaitClr
SysTaskWake
SysTimerCreate
SysTimerDelete
SysTimerRead
SysTimerWrite
SysTranslateKernelErr
SysUIBusy
SysUILaunch
SysUnimplemented
SysWantEvent
TimInit
TxtPrepFindString
UIInitialize
UIReset
UIPopTable
UIPushTable
WinAddWindow
WinDrawWindowFrame
WinDisableWindow
WinEnableWindow
WinGetWindowPointer
WinInitializeWindow
WinRemoveWindow
```

B

Compatibility Guide

This appendix lists groups of functions and other features (such as events and launch codes) that have been added to the Palm OS® after version 1.0.

Before you use any new functions or features in an application, you must check to ensure that they are implemented in the OS version your application is running on. Checking the OS version number is **not** a reliable indicator that a specific feature is present, since some later OS versions do not include features present in earlier versions. In order to ensure that your code is supported, you must check for the presence of individual features.

To make this check easier, this appendix lists new functions and features in groups such that all functions and features in a group are always implemented together in the ROM of a Palm device. This means that you can check for a single feature in that group and be assured that if that feature is present than all functions and features in that group are implemented.

Each group includes a recommended test to check if it is implemented. The following groups are described:

2.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 2.0 or higher. Use this FtrGet call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romversion);
```

The romversion parameter should be 0x02003000 or greater.

Launch Codes

This feature set adds the following launch codes:

sysAppLaunchCmdLookup
sysAppLaunchCmdPanelCalledFromApp
sysAppLaunchCmdReturnFromPanel
sysAppLaunchCmdSystemLock

Functions

This feature set adds the following functions:

CategoryInitialize
CategorySetName
DmDeleteCategory
DmDatabaseProtect
EvtAddUniqueEventToQueue
EvtEventAvail
EvtSysEventAvail
FldGetNumberOfBlankLines
FldGetScrollValues
FldSetInsertionPoint
FntGetScrollValues
FntWordWrap
FntWordWrapReverseNLines
FrmPointInTitle
FrmSetMenu
FrmSetObjectBounds
KeySetMask
LocGetNumberSeparators
LstScrollList
LstGetVisibleItems
MemCmp
MenuSetActiveMenuRscID
PhoneNumberLookup

PrefSetPreference
PrefGetPreference

SclDrawScrollBar
SclGetScrollBar
SclHandleEvent
SclSetScrollBar

SerControl

StrDelocalizeNumber
StrLocalizeNumber
StrNCaselessCompare
StrNCat
StrNCompare
StrNCopy
StrPrintF
StrVPrintF

SysBinarySearch
SysCreateDataBaseList
SysCreatePanelList
SysErrString
SysGraffitiReferenceDialog
SysLibLoad
SysStringByIndex
SysTicksPerSecond

TblHasScrollBar
TblSetBounds
TblSetColumnEditIndicator
TblSetRowStaticHeight

WinSetWindowBounds

Existing Functions that Changed

Several functions that existed in 1.0 were changed in 2.0:

CategoryCreateList (old function renamed CategoryCreateListV10)

CategoryEdit (old function renamed CategoryEditV10)

CategoryFreeList (old function renamed CategoryFreeListV10)

CategorySelect (old function renamed CategorySelectV10)

SelectDay (old function renamed SelectDayV10)

DmFindSortPosition (old function renamed DmFindSortPositionV10)

PrefGetAppPreferences (old function renamed PrefGetAppPreferencesV10)

`PrefOpenPreferenceDB` (old function renamed `PrefOpenPreferenceDBV10`)

`PrefSetAppPreferences` (old function renamed `PrefSetAppPreferencesV10`)

`SerReceive` (old function renamed `SerReceive10`)

`SerSend` (old function renamed `SerSend10`)

`SysKeyboardDialog` (old function renamed `SysKeyboardDialogV10`)

Other Changes

As a rule, all Palm OS applications developed with the 1.0 SDK should run error-free on the latest device. There are two possible pitfalls for 1.0 applications:

- **fldChangedEvent Change—**The operating system now correctly sends a `fldChangedEvent` whenever a field object is changed. Previously, the event was at times not sent, especially when a `FldSetText` operation was performed. If your application doesn't catch the events that are now sent, it may have problems.
- **Non-standard tools—**If your application was not developed with Metrowerks Code Warrior for the Palm OS, it may run into problems. One known problem can occur if the application:
  - was compiled with optimization turned on
  - uses system preferences

3.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.0 or higher. Use this `FtrGet` call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romversion);
```

The `romversion` parameter should be `0x03003000` or greater.

Launch Codes

This feature set adds the following launch codes:

`sysAppLaunchCmdExgAskUser`
`sysAppLaunchCmdExgReceiveData`

In addition, the launch code `sysAppLaunchCmdGoto` is now also sent by the exchange manager, in addition to its use by the global find operation.

Font

This feature set adds the following font:

`largeBoldFont`

Functions

This feature set adds the following functions:

Dynamic User Interface Functions

`CtlNewControl`
`CtlValidatePointer`
`FldNewField`
`FrmNewBitmap`
`FrmNewForm`
`FrmNewGadget`
`FrmNewLabel`
`FrmRemoveObject`
`FrmValidatePtr`
`LstNewList`
`WinValidateHandle`

For more information on creating and using dynamic user interface elements, see the section "Dynamic UI" on page 118 of the *Palm OS Programmer's Companion*.

Font Functions

`FontSelect`
`FntDefineFont`

For more information on these functions and the support for custom fonts, see the section "Fonts in Palm OS 3.0 and Later" on page 126 of the *Palm OS Programmer's Companion*.

Progress Manager Functions

PrgHandleEvent
PrgStartDialog
PrgStopDialog
PrgUpdateDialog
PrgUserCancel

For more information, see the section "Progress Dialogs" on page 84 of *the Palm OS Programmer's Companion*.

File Streaming Functions

FileClearerr
FileClose
FileControl
FileDelete
FileDmRead
FileEOF
FileError
FileFlush
FileGetLastError
FileOpen
FileRead
FileReadLow (system use only)
FileRewind
FileSeek
FileTell
FileTruncate
FileWrite

For more information, see the section "File Streaming Application Program Interface" on page 175 of *the Palm OS Programmer's Companion*.

Sound Functions

SndCreateMidiList
SndPlaySmf
SndDoCmd (enhanced in 3.0)

Exchange Manager Functions

ExgAccept
ExgDBRead
ExgDBWrite
ExgDisconnect
ExgPut
ExgReceive
ExgRegisterData
ExgSend

For more information, see the chapter Beaming (Infrared Communication) in the *Palm OS Programmer's Companion*.

IR Library Functions

IrAdvanceCredit
IrBind
IrClose
IrConnectIrLap
IrConnectReq
IrConnectRsp
IrDataReq
IrDisconnectIrLap
IrDiscoverReq
IrIAS_Add
IrIAS_GetInteger
IrIAS_GetIntLsap
IrIAS_GetObjectID
IrIAS_GetOctetString
IrIAS_GetOctetStringLen
IrIAS_GetType
IrIAS_GetUserString
IrIAS_GetUserStringCharSet
IrIAS_GetUserStringLen
IrIAS_Next
IrIAS_Query
IrIAS_SetDeviceName
IrIAS_StartResult
IrIsIrLapConnected
IrIsMediaBusy
IrIsNoProgress
IrIsRemoteBusy
IrLocalBusy
IrMaxRxSize
IrMaxTxSize
IrOpen
IrSetConTypeLMP
IrSetConTypeTTP
IrSetDeviceInfo
IrTestReq
IrUnbind

For more information, see the chapter Beaming (Infrared Communication) in the *Palm OS Programmer's Companion*.

Miscellaneous Functions

FrmRestoreActiveState
FrmSaveActiveState
ScrDisplayMode
SysGetAppInfo (system use only)
SysGetOSVersionString
SysGetROMToken
SysGetStackInfo
SysGremlins
TblGetItemFont
TblSetItemFont

Existing Functions that Changed

Two functions that existed in 2.0 were changed in 3.0:

CategoryEdit (old function renamed CategoryEditV20)

SysBatteryInfo (old function renamed SysBatteryInfoV20)

Other Changes

- The dynamic heap has been increased in size to 96 KB.
- Storage RAM is no longer subdivided into multiple storage heaps of 64 KB each. All storage RAM on a memory card is configured as a single storage heap.
- Each flash ROM-based Palm device holds a serial number that identifies it uniquely and can be retrieved via `SysGetROMToken`. For more information, see "Retrieving the ROM Serial Number" on page 216 of the *Palm OS Programmer's Companion*.
- The Application Launcher (accessed via the silkscreen "Applications" button) is now an application, rather than a popup. The `SysAppLauncherDialog` function, which provides the API to the old popup launcher, is still present in Palm OS 3.0 for compatibility purposes, but has not been updated and generally should not be used. For more information, see "Application Launcher" on page 131 of the *Palm OS Programmer's Companion*.
- The sound manager supports MIDI sound files, adding new sounds, asynchronous playback, and other features. There are also new selectors for setting the volume preferences. For more information, see the section "Sound" on page 199 of the *Palm OS Programmer's Companion*.

The following functions existed in the system previously, but were not documented:

`RctCopyRectangle`	`RctOffsetRectangle`
`RctGetIntersection`	`RctPtInRectangle`
`RctInsetRectangle`	`RctSetRectangle`

The following event type existed in the system previously, but was not previously documented:

`frmGotoEvent`

3.1 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.1 or higher. Use this `FtrGet` call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romversion);
```

The `romversion` parameter should be `0x03103000` or greater.

Functions

This feature set adds the following functions:

ChrHorizEllipsis
ChrNumericSpace
ContrastAdjust
FntWidthToOffset
FtrPtrNew
FtrPtrFree
FtrPtrResize
SelectOneTime
WinDrawChar
WinDrawTruncChars

NOTE: The `PalmOSGlue.lib` provides compatibility functions and macros for `ChrHorizEllipsis`, `ChrNumericSpace`, `WinDrawChar`, and `WinDrawTruncChars`. If you want to use these functions on systems that don't have the 3.1 feature set, you can link your application with `PalmOSGlue.lib`. See the chapter "PalmOSGlue Library" on page 1155 for more information.

Changes to the Character Encoding

Starting in Palm OS 3.1, the character encoding used on most systems is Microsoft Windows code page 1252. Versions prior to 3.1 used an encoding that was very similar to code page 1252 but did not follow it exactly. The following changes to the character set are introduced in Palm OS 3.1:

- Some of the special Palm OS glyphs in the high ASCII range (such as the shortcut stroke and the command stroke) have been moved down into the control code range, and other characters (such as the numeric space and horizontal ellipsis)

have been copied into the control range so that they're guaranteed to exist in every encoding. For the numeric space and horizontal ellipsis, you can use the macros `ChrNumericSpace` and `ChrHorizEllipsis` to return the appropriate character regardless of the character map. In `PalmOSGlue.lib`, these two macros are named `TxtGlueGetNumericSpaceChar` and `TxtGlueGetHorizEllipsisChar`, respectively.

- The four playing-card characters have been moved from the high ASCII range in the standard four fonts to the 9-point Symbol font.

Other Changes in 3.1

- Palm OS 3.1 supports a new processor: the EZ Dragonball processor. This processor is compatible with the existing Dragonball processor, so your application should run without changes as long as it doesn't access registers or system globals directly.

  If your application needs to know if it is running on an EZ Dragonball, it can check using the following code:

```
DWord id, chip;
Word revision;
Err err;
err = FtrGet(sysFtrCreator,
      sysFtrNumProcessorID, &id);
if (!err) {
  chip = id & sysFtrNumProcessorMask;
  revision = id & 0x0ffff;
  if (chip==sysFtrNumProcessor328)
    // traditional Dragonball
  else if (chip==sysFtrNumProcessorEZ)
    // Dragonball EZ
}
```

- The constant `preferenceDataVersion` was removed and replaced with `preferenceDataVerLatest`.
- Character variables are now two bytes long. The type `WChar` defines a character variable.

- The `keyDownEvent` structure's `chr` field (which contains the input character) has been changed from a `Word` to a `WChar`.
- The string manager functions `StrChr` and `StrStr` now treat buffers as characters, not arbitrary byte arrays. If you previously used these functions to search data buffers, your code may no longer work.
- The string manager function `StrToLower` can now handle any type of characters, including accented characters.
- The underline attribute of `FieldAttrType` now has support for the value 2. Previously, the only underline modes available were no underline (0) and gray underline (1). In Palm OS 3.1 and higher, the value 2 is interpreted as solid underline. The `UnderlineModeType` enum defined in `Window.h` defines the possible values for the underline attribute.
- The use of the `DmGetNextDatabaseByTypeCreator` `onlyLatestVers` parameter changed in 3.1. If `onlyLatestVers` is `true`, you only receive one matching database for each type/creator pair. In version 3.0 and earlier, you could receive multiple matching databases if `onlyLatestVers` was `true`. See that function's description for a more detailed description.

3.2 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.2 or higher. Use this `FtrGet` call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romversion);
```

The `romversion` parameter should be `0x03203000` or greater.

Functions

This feature set adds the following functions:

AlmGetProcAlarm
AlmSetProcAlarm
ClipboardAppendItem
DmGetDatabaseLockState
ErrAlert
SndPlaySmfResource

Existing Functions that Changed

Two functions that existed in 3.0 were changed in 3.2:

SysGremlins was removed and replaced with a SysGremlins macro that maps it to the function HostGremlinIsRunning. The prototype is slightly different, but you can still call SysGremlins in the same way you did before.

PrgStartDialog (old function renamed PrgStartDialogV31)

Other Changes in 3.2

- The prototype for the system use only function AlmDisplayAlarm changed from no return value to a Boolean return value. This change may affect system patches and extensions that intercept AlmDisplayAlarm calls.

International Feature Set

You can check that this feature set is implemented by checking for the existence of the international manager. You can check by calling FtrGet as follows:

```
err = FtrGet(sysFtrCreator, sysFtrNumIntlMgr,
&value);
```

If the international manager is installed, the value parameter will be non-zero and the returned error should also be zero (for no error).

You can learn more about the international manager by reading the chapter "Localized Applications" on page 317 in the *Palm OS Programmer's Companion*.

NOTE: If you want to use international functions on systems that don't have the international feature, you can link your application with `PalmOSGlue.lib`. The functions in this library are the same as those listed below except that they use the prefix "TxtGlue" instead of "Txt." For more information, see the chapter "PalmOSGlue Library" on page 1155.

Functions

This feature set adds the following functions:

Text Manager Functions

TxtByteAttr
TxtCaselessCompare
TxtCharAttr
TxtCharBounds
TxtCharEncoding
TxtCharIsAlNum
TxtCharIsAlpha
TxtCharIsCntrl
TxtCharIsDigit
TxtCharIsGraph
TxtCharIsHardKey
TxtCharIsHex
TxtCharIsLower
TxtCharIsPrint
TxtCharIsPunct
TxtCharIsSpace
TxtCharIsUpper
TxtCharSize
TxtCharWidth
TxtCharXAttr
TxtCompare
TxtEncodingName
TxtFindString
TxtGetChar
TxtGetNextChar
TxtGetPreviousChar
TxtCharIsValid
TxtMaxEncoding
TxtNextCharSize
TxtPreviousCharSize
TxtReplaceStr
TxtSetNextChar
TxtStrEncoding
TxtTransliterate
TxtGetTruncationOffset
TxtWordBounds

Other Functions

IntlGetRoutineAddress

Removed Functions and Macros

If the international feature set exists, then the following functions and macros are no longer available:

GetCharAttr
GetCharCaselessValue
GetCharSortValue
IsAscii
IsAlNum
IsAlpha
IsCntrl
IsDigit
IsGraph
IsLower
IsPrint
IsPunct
IsSpace
IsUpper
IsHex
IsDelim

Japanese Feature Set

You can check that the Japanese feature set is implemented by checking if the unit is Japanese. You can check by calling FtrGet as follows:

```
err = FtrGet(sysFtrCreator, sysFtrNumEncoding,
&value);
```

The unit has the Japanese OS if the value parameter is charEncodingCP932.

For further information about the Japanese implementation, see the section "Notes on the Japanese Implementation" in the *Palm OS Programmer's Companion*.

Wireless Internet Feature Set

You can check that this feature set is implemented by checking for the existence of the Clipper and iMessenger™ applications. Here's an example of how to check for Clipper:

```
DmSearchStateType searchState;
UInt cardNo;
LocalID dbID;
err = DmGetNextDatabaseByTypeCreator(true,
&searchState, sysFileTApplication,
sysFileCClipper, true, &cardNo, &dbID);
```

If Clipper is not present, the `DmGetNextDatabaseByTypeCreator` routine returns an error. To check for iMessenger, you can use the creator type `sysFileCMessaging`.

You can learn more about the Palm.Net™ system for wireless Internet access and the programmatic interfaces to the Clipper and iMessenger applications by reading the chapter "Internet and Messaging Applications" in the *Palm OS Programmer's Companion*.

Launch Codes

This feature set adds the following launch codes:

`sysAppLaunchCmdAddRecord` (for iMessenger application; existed for Mail in 3.0)
`sysAppLaunchCmdGoToURL`
`sysAppLaunchCmdOpenDB`
`sysAppLaunchCmdURLParams`

Events

This feature set adds the following events:

`inetSockReadyEvent`
`inetSockStatusChangeEvent`

This feature set also adds the following `keyDownEvent` key codes:

```
vchrHardAntenna
vchrRadioCoverageOK
vchrRadioCoverageFail
```

These key codes are described in the section New keyDownEvent Key Codes.

Functions

This feature set adds the following functions.

Internet Library Functions

For more information, see the chapter "Network Communication" in the *Palm OS Programmer's Companion*.

INetLibCacheGetObject	INetLibSockClose
INetLibCacheList	INetLibSockConnect
INetLibCheckAntennaState	INetLibSockHTTPAttrGet
INetLibClose	INetLibSockHTTPAttrSet
INetLibConfigAliasGet	INetLibSockHTTPReqCreate
INetLibConfigAliasSet	INetLibSockHTTPReqSend
INetLibConfigDelete	INetLibSockOpen
INetLibConfigIndexFromName	INetLibSockRead
INetLibConfigList	INetLibSockSettingGet
INetLibConfigMakeActive	INetLibSockSettingSet
INetLibConfigRename	INetLibSockStatus
INetLibConfigSaveAs	INetLibURLCrack
INetLibGetEvent	INetLibURLGetInfo
INetLibOpen	INetLibURLOpen
INetLibSettingGet	INetLibURLsAdd
INetLibSettingSet	INetLibWiCmd

New Serial Manager Feature Set

You can check that this feature set is implemented by checking for the existence of the new serial manager. You can check by calling FtrGet as follows:

```
err = FtrGet(sysFileCSerialMgr,
sysFtrNewSerialPresent, &value);
```

If the new serial manager is installed, the `value` parameter will be non-zero and the returned error should also be zero (for no error).

You can learn more about the new serial manager and connection manager by reading the sections "The New Serial Manager" on page 232 and "The Connection Manager" on page 245 in the *Palm OS Programmer's Companion*.

Functions

This feature set adds the following functions.

New Serial Manager Functions

SrmClearErr
SrmClose
SrmControl
SrmGetDeviceCount
SrmGetDeviceInfo
SrmGetStatus
SrmOpen
SrmOpenBackground
SrmPrimeWakeupHandler
SrmReceive
SrmReceiveCheck
SrmReceiveFlush
SrmReceiveWait
SrmReceiveWindowClose
SrmReceiveWindowOpen
SrmSend
SrmSendCheck
SrmSendFlush
SrmSendWait
SrmSetReceiveBuffer
SrmSetWakeupHandler
WakeupHandlerProc

Serial Driver Functions

DrvEntryPoint
SdrvClose
SdrvControl
SdrvISP
SdrvOpen
SdrvReadChar
SdrvStatus
SdrvWriteChar

Virtual Driver Functions

DrvEntryPoint
GetSize
GetSpace
VdrvControl
VdrvOpen
VdrvStatus
VdrvWrite
WriteBlock
WriteByte

Connection Manager Functions

CncAddProfile
CncDeleteProfile
CncGetProfileInfo
CncGetProfileList

Serial Link Manager Function

SlkSocketPortID

3.5 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.5 or higher. Use this FtrGet call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romversion);
```

The romversion parameter should be 0x03503000 or greater.

Launch Codes

This feature set adds the following launch codes:

sysAppLaunchCmdNotify

Events

This feature set adds the following events:

frmGadgetEnterEvent
frmGadgetMiscEvent
menuCmdBarOpenEvent
menuOpenEvent

Functions

This feature set adds the following functions.

Bitmaps

BmpBitsSize
BmpColortableSize
BmpCompress
BmpCreate
BmpDelete
BmpGetBits
BmpGetColortable
BmpSize
ColorTableEntries

Controls

CtlGetSliderValues
CtlNewGraphicControl
CtlNewSliderControl
CtlSetGraphics
CtlSetSliderValues

Forms

FrmCustomResponseAlert
FrmNewGsi
FrmSetGadgetHandler

Menus

MenuAddItem
MenuCmdBarDisplay
MenuHideItem
MenuCmdBarAddButton
MenuCmdBarGetButtonData
MenuShowItem

Overlay Manager

OmGetCurrentLocale
OmGetIndexedLocale
OmGetRoutineAddress
OmGetSystemLocale
OmLocaleToOverlayDBName
OmOverlayDBNameToLocale
OmSetSystemLocale

Private Records

SecSelectViewStatus
SecVerifyPW

Tables

TblGetItemPtr
TblRowMasked
TblSetColumnMasked
TblSetRowMasked

UI Colors

UIColorGetTableEntryIndex
UIColorGetTableEntryRGB
UIColorSetTableEntry

UI Controls

UIBrightnessAdjust
UIPickColor

Windows

WinCreateBitmapWindow
WinDrawPixel
WinErasePixel
WinGetBitmap
WinGetPatternType
WinGetPixel
WinIndexToRGB
WinInvertPixel
WinPaintBitmap
WinPaintChar
WinPaintChars
WinPaintLine
WinPaintLines
WinPaintPixel
WinPaintPixels
WinPaintRectangle
WinPaintRectangleFrame
WinPalette
WinPopDrawState
WinPushDrawState
WinRGBToIndex
WinScreenLock
WinScreenMode
WinScreenUnlock
WinSetBackColor
WinSetDrawMode
WinSetForeColor
WinSetPatternType
WinSetTextColor

Miscellaneous New Functions

DmOpenDBNoOverlay
ExgDoDialog
DateToAscii
ResLoadConstant
TxtParamString

Existing Functions that Changed

The following functions that existed prior to 3.5 have changed in release 3.5:

`ScrDisplayMode` was changed to `WinScreenMode`.

`ContrastAdjust` was changed to `UIContrastAdjust`.

`SelectTime` (old function renamed `SelectTimeV33`)

New Data Types

The data types `Byte`, `Word`, `DWord` and so on are now deprecated. It is recommend that you use the corresponding new data types. For example, use `Int16` instead of `SWord` and `UInt32` instead of `DWord`. In particular, the unfortunate distinction between `Handle`/`VoidHand` has been fixed; use `MemHandle` instead.

To learn in general how the type names changed, see the header file `PalmOSCompatibility.h`. This file provides a mapping from the old type name to the new type name. If you need to move forward without modifying your code, you can include this file in your project to provide declarations for the old type names.

Changes to Events

- The `tapCount` field has been added to the `EventType` structure. The `tapCount` field specifies the number of times the user tapped the pen at the current location; in fields, two taps selects a word, and three taps selects a line.

IMPORTANT: Because the `tapCount` field has been added to the `EventType` structure, it has become more critical that you clear the event structure before using it to add a new event to the queue. Otherwise, the `tapCount` will be incorrect for the new event.

- The structures for `ctlRepeatEvent` and `ctlSelectEvent` have a `value` field added to them. This new field is used only for sliders; it holds the current value of the slider.
- Form objects now handle the `frmTitleSelectEvent` by adding a `keyDownEvent` with the `vchrMenu` character to the event queue (which causes the form's menu to display).

- Some of the structure definitions for system-level events have moved from `Event.h` to `SysEvent.h`.
- The `winEnterEvent` is now not generated until `FrmDrawForm` is called. Make sure to draw your form in response to `frmOpenEvent`, not `winEnterEvent`.

Other Changes

- `FrmDrawForm`

  On release 3.5, `FrmDrawForm` erases the window's rectangle before it draws, so you must perform custom drawing after the call to `FrmDrawForm`, not before. If you have drawn before the call to `FrmDrawForm`, your changes are lost. On debug ROMs, the window handle is invalid until `FrmDrawForm` is called so that draws before `FrmDrawForm` result in a bus error.

- Resource Manager

  The resource manager functions have been updated to work with overlay databases. See "Using Overlays to Localize Resources" on page 318 in the *Palm OS Programmer's Companion*.

- `DmGetDatabase`

  The order in which this call returns databases has changed. Previously all of the databases from ROM were returned first, then all from RAM. Now they are intermingled. Developers should not rely on the order in which databases are returned from this call.

- `StrToLower`

  This function is different in 3.5 Latin ROMs. Previously it only changed A through Z. Now it also changes high ASCII characters.

- Time Manager

  If you are using a debug ROM, the string buffer is filled with `dateStringLength` or `longStrLength` debugging bytes, depending on the dateFormat parameter. For the routines that return the day-of-week name in addition to the date, the

size of the buffers has been expanded, so developers need to check the max lengths defined in `DateTime.h`.

- The format of the storage heap header has changed, thus any existing saved Simulator card images are invalid and should be tossed.
- Category Data Structures

  The data structure `AppInfoType` has been documented.

  `CategoryCreateList` now has a "hide" function with two new constants; `categoryHideEditCategory, and categoryDefaultEditCategoryString`.
- FtrPtrNew

  `FtrPtrNew` now allows allocating chunks larger than 64KB.
- Dynamic heap

  The dynamic heap is now sized based on the amount of memory available to the system.

Device RAM Size	Heap Size
x < 2MB	64KB
2MB ð x < 4MB	128KB
x Š 4MB	256KB

Notification Feature Set

You can check that this feature set is implemented by checking for the existence of the notification manager. You can check by calling `FtrGet` as follows:

```
err = FtrGet(sysFtrCreator,
sysFtrNumNotifyMgrVersion, &value);
```

If the notification manager is part of the system, the `value` parameter will be non-zero and the returned error should also be zero (for no error).

Notification Manager

SysNotifyBroadcast
SysNotifyBroadcastDeferred
SysNotifyRegister
SysNotifyUnregister

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W