



# **Welcome to**

# **Developing Palm OS 2.0**

# **Applications**

## **Part II: System Management**

**Navigate this online document as follows:**

<b>To see bookmarks</b>	<b>Type Control-7</b>
<b>To see information on Adobe Acrobat Reader</b>	<b>Type Control-?</b>
<b>To navigate</b>	<b>Click on any blue hypertext link any <a href="#">Table of Contents</a> entry arrows in the menu bar</b>





**U.S. Robotics®**

**Developing Palm OS™ 2.0  
Applications**

**Part II**

©1996, 1997 U.S. Robotics, Inc. All rights reserved.

Documentation stored on the compact disk may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from U.S. Robotics.

U.S. Robotics, the U.S. Robotics logo and Graffiti are registered trademarks, and Palm Computing, HotSync, the Palm OS, and the Palm OS logo are trademarks of U.S. Robotics and its subsidiaries.

All other trademarks or registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISC ARE SUBJECT TO THE LICENSE AGREEMENT.

## Contact Information:

---

<b>Metrowerks U.S.A. and international</b>	Metrowerks Corporation 2201 Donley Drive, Suite 310 Austin, TX 78758 U.S.A.
<b>Metrowerks Canada</b>	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
<b>Metrowerks Mail order</b>	Voice: 1-800-377-5416 Fax: 1-512-873-4901
<b>U.S. Robotics, Palm Computing Division Mail Order</b>	U.S.A. and Canada: 1-800-881-7256 elsewhere 1-408-848-5604
<b>Metrowerks World Wide Web</b>	<a href="http://www.metrowerks.com">http://www.metrowerks.com</a>
<b>U.S. Robotics, Palm Computing Division World Wide Web</b>	<a href="http://www.usr.com/palm">http://www.usr.com/palm</a>
<b>Registration information</b>	<a href="mailto:register@metrowerks.com">register@metrowerks.com</a>
<b>Technical support</b>	<a href="mailto:support@metrowerks.com">support@metrowerks.com</a>
<b>Sales, marketing, &amp; licensing</b>	<a href="mailto:sales@metrowerks.com">sales@metrowerks.com</a>
<b>CompuServe</b>	goto Metrowerks

---

# Table of Contents

---

<b>Table of Contents . . . . .</b>	<b>.v</b>
<b>About This Document. . . . .</b>	<b>11</b>
Palm OS SDK Documentation . . . . .	11
What This Guide Contains . . . . .	12
Conventions Used in This Guide . . . . .	12
<b>1 Using Palm OS System Managers . . . . .</b>	<b>13</b>
The Alarm Manager. . . . .	14
Alarm Manager Overview. . . . .	15
Using the Alarm Manager . . . . .	17
Alarm Manager Function Summary . . . . .	17
The Error Manager . . . . .	17
Displaying Development Errors . . . . .	18
Using the Error Manager Macros . . . . .	19
Understanding the Try-and-Catch Mechanism . . . . .	20
Using the Try and Catch Mechanism . . . . .	21
Error Manager Function Summary . . . . .	22
The Feature Manager . . . . .	22
The System Version Feature . . . . .	23
Application-Defined Features . . . . .	23
Using the Feature Manager . . . . .	23
Feature Manager Function Summary . . . . .	24
The Sound Manager. . . . .	25
Using the Sound Manager. . . . .	25
Sound Manager Function Summary . . . . .	25
The String Manager . . . . .	26
The System Manager . . . . .	27
System Boot and Reset . . . . .	27
Power Management . . . . .	28
The Microkernel . . . . .	30
Application Support . . . . .	31
System Manager Functions . . . . .	35
The System Event Manager . . . . .	36

## Table of Contents

---

Event Translation: Pen Strokes to Key Events. . . . .	37
Pen Queue Management . . . . .	37
Key Queue Management . . . . .	38
Auto-Off Control. . . . .	39
System Event Manager Function Summary . . . . .	40
The Time Manager . . . . .	41
Using Real-Time Clock Functions. . . . .	41
Using System Ticks Functions . . . . .	41
Time Manager Structures . . . . .	42
Time Manager Function Summary . . . . .	43
<b>2 Palm OS System Functions . . . . .</b>	<b>45</b>
Alarm Manager API. . . . .	45
AlmGetAlarm . . . . .	45
AlmSetAlarm . . . . .	46
Functions for System Use Only. . . . .	47
Error Manager Functions . . . . .	48
ErrDisplay . . . . .	48
ErrDisplayFileLineMsg . . . . .	49
ErrFatalDisplayIf. . . . .	50
ErrNonFatalDisplayIf. . . . .	51
ErrThrow . . . . .	52
Event Manager Functions . . . . .	53
EvtAddEventToQueue . . . . .	53
<u>EvtAddUniqueEventToQueue</u> . . . . .	<u>53</u>
EvtCopyEvent . . . . .	54
EvtDequeuePenPoint . . . . .	54
EvtDequeuePenStrokeInfo. . . . .	55
EvtEnableGraffiti. . . . .	55
EvtEnqueueKey . . . . .	56
<u>EvtEventAvail</u> . . . . .	<u>57</u>
EvtFlushKeyQueue. . . . .	57
EvtFlushNextPenStroke . . . . .	58
EvtFlushPenQueue . . . . .	58
EvtGetEvent. . . . .	59
EvtGetPen. . . . .	59

---

EvtGetPenBtnList . . . . .	60
EvtKeyQueueEmpty . . . . .	60
EvtKeyQueueSize . . . . .	61
EvtPenQueueSize . . . . .	61
EvtProcessSoftKeyStroke . . . . .	62
EvtResetAutoOffTimer . . . . .	62
<a href="#">EvtSysEventAvail</a> . . . . .	<a href="#">63</a>
EvtWakeup . . . . .	63
Functions for System Use Only . . . . .	64
Feature Functions. . . . .	66
FtrGet . . . . .	66
FtrGetByIndex . . . . .	67
FtrSet . . . . .	68
FtrUnregister . . . . .	69
For System Use Only . . . . .	70
Find Functions . . . . .	71
FindDrawHeader . . . . .	71
FindGetLineBounds . . . . .	71
FindSaveMatch . . . . .	72
FindStrInStr . . . . .	72
Float Manager Functions . . . . .	74
Using the New Floating Point Arithmetic . . . . .	74
Using 1.0 Floating-Point Functionality . . . . .	74
FplAdd . . . . .	75
FplAToF. . . . .	75
FplBase10Info . . . . .	75
FplDiv . . . . .	76
FplFloatToLong . . . . .	77
FplFloatToULong . . . . .	77
FplFree . . . . .	78
FplFToA . . . . .	78
FplInit . . . . .	79
FplLongToFloat . . . . .	79
FplMul . . . . .	80
FplSub . . . . .	80
Miscellaneous System Functions . . . . .	81

---

## Table of Contents

---

CmBroadcast . . . . .	81
Crc16CalcBlock . . . . .	81
MdmDial . . . . .	81
MdmHangUp . . . . .	83
<a href="#"><u>PhoneNumberLookup</u></a>	<a href="#"><u>83</u></a>
ResLoadForm . . . . .	84
ResLoadMenu . . . . .	84
System Preferences Functions . . . . .	85
<a href="#"><u>PrefGetAppPreferences</u></a>	<a href="#"><u>85</u></a>
<a href="#"><u>PrefGetAppPreferencesV10</u></a>	<a href="#"><u>86</u></a>
<a href="#"><u>PrefGetPreference</u></a>	<a href="#"><u>87</u></a>
PrefGetPreferences . . . . .	88
PrefOpenPreferenceDBV10 . . . . .	88
<a href="#"><u>PrefSetAppPreferences</u></a>	<a href="#"><u>89</u></a>
<a href="#"><u>PrefSetAppPreferencesV10</u></a>	<a href="#"><u>89</u></a>
<a href="#"><u>PrefSetPreference</u></a>	<a href="#"><u>90</u></a>
PrefSetPreferences . . . . .	91
Password Functions. . . . .	92
PwdExists. . . . .	92
PwdRemove. . . . .	92
PwdSet . . . . .	93
PwdVerify. . . . .	93
String Manager Functions . . . . .	94
StrAToI . . . . .	94
StrCat. . . . .	94
StrCaselessCompare . . . . .	95
StrChr . . . . .	95
StrCompare . . . . .	96
StrCopy. . . . .	96
<a href="#"><u>StrDelocalizeNumber</u></a>	<a href="#"><u>97</u></a>
StrIToA . . . . .	97
StrIToH . . . . .	98
StrLen . . . . .	98
<a href="#"><u>StrLocalizeNumber</u></a>	<a href="#"><u>99</u></a>
<a href="#"><u>StrNCaselessCompare</u></a>	<a href="#"><u>99</u></a>
<a href="#"><u>StrNCat</u></a>	<a href="#"><u>100</u></a>
<a href="#"><u>StrNCompare</u></a>	<a href="#"><u>100</u></a>



## Table of Contents

---

<a href="#"><u>StrNCopy</u></a>	<a href="#"><u>101</u></a>
<a href="#"><u>StrPrintf</u></a>	<a href="#"><u>101</u></a>
StrStr . . . . .	102
StrToLower . . . . .	102
<a href="#"><u>StrVPrintf</u></a>	<a href="#"><u>102</u></a>
Sound Manager Functions . . . . .	105
SndDoCmd . . . . .	105
SndGetDefaultVolume . . . . .	106
SndPlaySystemSound. . . . .	106
SndSetDefaultVolume. . . . .	107
Functions for System Use Only. . . . .	107
System Functions . . . . .	108
SysAppLaunch . . . . .	108
SysAppLauncherDialog. . . . .	109
SysBatteryInfo . . . . .	109
<a href="#"><u>SysBinarySearch</u></a>	<a href="#"><u>110</u></a>
SysBroadcastActionCode . . . . .	111
SysCopyStringResource. . . . .	112
<a href="#"><u>SysCreateDataBaseList</u></a>	<a href="#"><u>112</u></a>
<a href="#"><u>SysCreatePanelList</u></a>	<a href="#"><u>113</u></a>
SysCurAppDatabase . . . . .	113
<a href="#"><u>SysErrString</u></a>	<a href="#"><u>114</u></a>
SysFatalAlert . . . . .	114
SysFormPointerArrayToStrings . . . . .	115
<a href="#"><u>SysGraffitiReferenceDialog</u></a>	<a href="#"><u>115</u></a>
SysHandleEvent . . . . .	116
SysInsertionSort . . . . .	117
SysInstall . . . . .	119
<a href="#"><u>SysKeyboardDialog</u></a>	<a href="#"><u>119</u></a>
<a href="#"><u>SysKeyboardDialogV10</u></a>	<a href="#"><u>120</u></a>
<a href="#"><u>SysLibLoad</u></a>	<a href="#"><u>120</u></a>
SysQSort . . . . .	121
SysRandom . . . . .	122
SysReset . . . . .	123
SysSetAutoOffTime. . . . .	123
<a href="#"><u>SysStringByIndex</u></a>	<a href="#"><u>124</u></a>
SysTaskDelay . . . . .	124

## Table of Contents

---

<a href="#"><u>SysTicksPerSecond</u></a>	<a href="#"><u>125</u></a>
SysUIAppSwitch . . . . .	125
Functions for System Use Only . . . . .	126
Time Manager Functions . . . . .	135
DateAdjust . . . . .	135
DateDaysToDate . . . . .	135
DateSecondsToDate . . . . .	136
DateToAscii . . . . .	136
DateToDays . . . . .	137
DateToDOWDMFormat . . . . .	137
DayOfMonth . . . . .	138
DayOfWeek . . . . .	138
DaysInMonth . . . . .	139
TimAdjust . . . . .	139
TimDateTimeToSeconds . . . . .	140
TimGetSeconds . . . . .	140
TimGetTicks . . . . .	140
TimSecondsToDateTime . . . . .	141
TimSetSeconds . . . . .	141
TimeToAscii . . . . .	142
Functions for System Use Only . . . . .	143
<b>Index . . . . .</b>	<b>145</b>



# About This Document

---

Developing Palm OS 2.0 Applications, Part II, is part of the Palm OS Software Development Kit (SDK). This introduction provides an overview of the 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 2.0 Tutorial	21 Phases step developers through using the different parts of the system. Example applications for each phase are included in the SDK.
Developing Palm OS 2.0 Applications. Part I: Interface Management	A programmer's guide and reference document that discusses all important aspects of developing an applications.
Developing Palm OS 2.0 Applications. Part II. System Management.	A programmer's guide and reference document for all system managers, such as the string manager or the system event manager. See <a href="#">What This Guide Contains</a> for details.
Developing Palm OS 2.0 Applications, Part III. Memory and Communications Management	Programmer's guide and reference document about <ul style="list-style-type: none"><li>• Memory management; both the database manager and the memory manager.</li><li>• The Palm OS communications library for serial communication.</li><li>• The Palm OS network library, which provides basic network services.</li></ul>
Palm OS 2.0 Cookbook.	Provides a variety of design guidelines, including localization, UI design, and optimization. Information about using CodeWarrior for Pilot to create projects and executables.

## About This Document

### *What This Guide Contains*

---

## What This Guide Contains

This section provides an overview of the chapters in this guide.

- Chapter 1, [“Using Palm OS System Managers,”](#) discusses the managers that provide system functionality, including the system event manager, time manager, and error manager.
- Chapter 2, [“Palm OS System Functions,”](#) provides reference-style information for each API function that allows applications to interact with the system.

## Conventions Used in This Guide

This guide uses the following typographical conventions:

This style...	Is used for...
<code>fixed width font</code>	Code elements such as function, structure, field, bitfield.
<u><code>fixed width underline</code></u>	Emphasis (for code elements).
<b>bold</b>	Emphasis (for other elements).
<a href="#">blue and underlined</a>	Hot links.
<u>black and underlined</u>	2.0 function names (headings only)
<u>red and underlined</u>	2.0 function names (in Table of Contents only)



# Using Palm OS System Managers

---

In contrast to desktop computer operating systems, Palm OS consists of only one library. This library, however, contains several managers, which are groups of functions that work together to implement certain functionality. As a rule, all functions that belong to one manager use the same three-letter prefix and work together to implement a certain aspect of functionality.

In this chapter, you learn about all Palm OS managers that aren't directly responsible for interface management or memory management. As you investigate the managers more closely you'll find that some of them are mostly services provided by the system, while others contain a large number of API calls.

This chapter presents the managers in alphabetical order for easier access.

- [The Alarm Manager](#) provides support for setting real-time alarms to perform some periodic activity or display a reminder.
- [The Error Manager](#) can be used by applications or system software for displaying unexpected runtime errors, such as those that typically show up during program development.

Final production versions of applications or system software are not expected to use error manager.

- [The Feature Manager](#) provides information about the system software version and the optional system features and third-party extensions that are installed. An application can also use the feature manager to keep track of its own data.
- [The Sound Manager](#) lets applications and system modules control sound manager settings and play custom and predefined system sounds.

## Using Palm OS System Managers

### *The Alarm Manager*

---

- [The String Manager](#) is a set of string manipulation functions available to applications. Use these routines instead of the standard C routines.
- [The System Manager](#) is responsible for the basic operation of the system, including booting and resetting the system, managing power, managing the microkernel, and supporting applications.
- [The System Event Manager](#) provides an interface to the low-level pen and key event queues, translates taps on silk-screened icons into key events, sends pen strokes in the Graffiti area to the Graffiti recognizer, and puts the system into low-power doze mode when there is no user activity.
- [The System Manager](#) provides real-time clock functions and system tick functions.

## The Alarm Manager

The Palm OS alarm manager provides support for setting real-time alarms, for performing some periodic activity, or for displaying a reminder. This section helps you use the alarm manager by discussing these topics:

- [Alarm Manager Overview](#)
- [Using the Alarm Manager](#)
- [Alarm Manager Function Summary](#)

## Alarm Manager Overview

The alarm manager:

- Works closely with the time manager to handle real-time alarms.
- Sends launch codes to applications that set a specific time alarm to inform the application the alarm is due.
- Handles alarms by application in a two cycle operation
  - First, it notifies each application that the alarm has occurred.
  - Second, it allows each application to display some UI.
- Allows only one alarm to be set per application

However, the alarm manager

- Doesn't provide reminder dialog boxes.
- Doesn't play the alarm sound.

The following section looks in some detail at how the alarm manager and applications interact when processing an alarm.

### Alarm Queue

The alarm queue contains all alarm requests. Triggered alarms are queued up until the alarm manager can send the launch code to the application that created the alarm. However, if the alarm queue becomes full, the oldest entry that has been both triggered and notified is deleted to make room for a new alarm.

### Alarm Manager Processing

When an alarm is triggered, the alarm manager notifies each application that set an alarm for that alarm time via the `sysAppLaunchCmdAlarmTriggered` launch code.

After each application has processed this launch code, the alarm manager sends each application the `sysAppLaunchCmdDisplayAlarm` launch code in order for the application to display the alarm.

If a new alarm time is triggered while an older alarm is still being displayed, all applications with alarms scheduled for this second alarm time are sent the `sysAppLaunchCmdAlarmTriggered` launch code, but the display cycle is postponed until all earlier alarms have finished displaying.

#### **Alarm Scenario**

The alarm manager typically first notifies each application that an alarm has been triggered, then notifies each application to display the alarm. Here's how an application and the alarm manager typically interact when processing an alarm

1. When the alarm time is reached, the alarm manager finds the first application in the alarm queue that set an alarm for this alarm time.
2. The alarm manager sends this application the `sysAppLaunchCmdAlarmTriggered` launch code.
3. The application can now:
  - Set the next alarm.
  - Play a short sound.
  - Perform some maintenance activity.
4. The alarm manager finds in the alarm queue the next application that set an alarm and repeats steps 2 and 3.
5. This process is repeated until no more applications are found with this alarm time.
6. The alarm manager then finds once again the first application in the alarm queue who set an alarm for this alarm time and sends this application the `sysAppLaunchCmdDisplayAlarm` launch code
7. The application can now:
  - Display a dialog box
  - Display some other type of reminder
8. The alarm manager processes the alarm queue for the next application that set an alarm for the alarm being triggered and step 6 and 7 are repeated.
9. This process is repeated until no more applications are found with this alarm time.



## Using the Alarm Manager

An applications can use the Palm OS function [AlmSetAlarm](#) to set and/or clear an alarm.

An application can find out its current alarm setting by using the [AlmGetAlarm](#) function. This function returns the alarm date and time (expressed in seconds since 1/1/1904). The return value is 0 if no active alarm exists for the application.

## Alarm Manager Function Summary

The following alarm manager functions are for application use:

- [AlmGetAlarm](#)
- [AlmSetAlarm](#)

# The Error Manager

The error manager can be used by applications or system software for displaying unexpected runtime errors such as those that typically show up during program development. Final versions of applications or system software won't use the error manager.

The error manager API consists of a set of functions for displaying an alert with an error message, file name, and the line number where the error occurred. If a debugger is connected, it is entered when the error occurs.

The error manager also provides a “try and catch” mechanism that applications can use for handling such runtime errors as out of memory conditions, user input errors, etc. This mechanism is closely modeled after the try/catch functionality of the recent ANSI C specification.

This section helps you understand and use the error manager, discussing the following topics:

- [Displaying Development Errors](#)
- [Understanding the Try-and-Catch Mechanism](#)
- [Using the Error Manager Macros](#)
- [Error Manager Function Summary](#)

## Displaying Development Errors

The error manager provides some compiler macros that can be used in source code. These macros display a fatal alert dialog on the screen and provide buttons to reset the device or enter the debugger after the error is displayed. There are three macros: [ErrDisplay](#), [ErrFatalDisplayIf](#), and [ErrNonFatalDisplayIf](#).

- `ErrDisplay` always displays the error message on the screen.
- `ErrFatalDisplayIf` and `ErrNonFatalDisplayIf` display the error message only if their first argument is `TRUE`.

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 <code>ERROR_CHECK_LEVEL</code> to...	The compiler...
<code>ERROR_CHECK_NONE (0)</code>	Doesn't compile in any error calls.
<code>ERROR_CHECK_PARTIAL (1)</code>	Compiles in only <code>ErrDisplay</code> and <code>ErrFatalDisplayIf</code> calls.
<code>ERROR_CHECK_FULL (2)</code>	Compiles in all three calls.

During development, it makes sense to set full error checking for early development, partial error checking during alpha and beta test periods, and no error checking for the final product. At partial error checking, only fatal errors are displayed; error conditions that are only possible are ignored under the assumption that the application developer is already aware of the condition and designed the software to operate that way.

## Using the Error Manager Macros

Calls to the error manager to display errors are actually compiler macros that are conditionally compiled into your program. Most of the calls take a boolean parameter, which should be set to TRUE to display the error, and a pointer to a text message to display if the condition is true.

Typically, the boolean parameter is an in-line expression that evaluates to TRUE if there is an error condition. As a result, both the expression that evaluates the error condition and the message text are left out of the compiled code when error checking is turned off. You can call [ErrFatalDisplayIf](#), or [ErrDisplay](#), but using [ErrFatalDisplayIf](#) makes your source code look neater.

For example, assume your source code looks like this:

```
result = DoSomething();
ErrFatalDisplayIf (result < 0, "unexpected
                    result from DoSomething");
```

With error checking turned on, this code displays an error alert dialog if the result from `DoSomething()` is less than 0. Besides the error message itself, this alert also shows the file name and line number of the source code that called the error manager. With error checking turned off, both the expression evaluation `err < 0` and the error message text are left out of the compiled code.

The same net result can be achieved by the following code:

```
result = DoSomething();
#if ERROR_CHECK_LEVEL != ERROR_CHECK_NONE
if (result < 0)
    ErrDisplay ("unexpected result from
                DoSomething");
#endif
```

However, this solution is longer and requires more work than simply calling [ErrFatalDisplayIf](#). It also makes the source code harder to follow.

## Understanding the Try-and-Catch Mechanism

The try-and-catch mechanism of the error manager is closely modeled after the ANSI C try and catch standard.

The error manager is aware of the machine state of the Palm OS device and can therefore correctly save and restore this state. The built-in try and catch of the compiler can't be used because it's machine dependent.

Try and catch is basically a neater way of implementing a `goto` if an error occurs. A typical way of handling errors in the middle of a routine is to go to the end of the routine as soon as an error occurs and have some general-purpose cleanup code at the end of every routine. Errors in nested routines are even trickier because the result code from every subroutine call must be checked before continuing.

When you set up a try/catch, you are providing the compiler with a place to jump to when an error occurs. You can go to that error handling routine at any time by calling [`ErrThrow`](#). When the compiler sees the `ErrThrow` call, it performs a `goto` to your error handling code. The greatest advantage to calling `ErrThrow`, however, is for handling errors in nested subroutine calls.

Even if `ErrThrow` is called from a nested subroutine, execution immediately goes to the same error handling code in the higher-level call. The compiler and runtime environment automatically strip off the stack frames that were pushed onto the stack during the nesting process and go to the error handling section of the higher-level call. You no longer have to check for result codes after calling every subroutine; this greatly simplifies your source code and reduces its size.

## Using the Try and Catch Mechanism

The following example illustrates the possible layout for a typical routine using the error manager's try and catch mechanism.

### Listing 1.1 Try and Catch Mechanism Example

---

```
ErrTry {
    p = MemPtrNew(1000);
    if (!p) ErrThrow(errNoMemory);
    MemSet(p, 1000, 0);
    CreateTable(p);
    PrintTable(p);
}

ErrCatch(err) {
    // Recover or cleanup after a failure in the
    // above Try block. "err" is an int
    // identifying the reason for the failure.

    // You may call ErrThrow() if you want to
    // jump out to the next Catch block.

    // The code in this Catch block doesn't
    // execute if the above Try block completes
    // without a Throw.

    if (err == errNoMemory)
        ErrDisplay("Out of Memory");
    else
        ErrDisplay("Some other error");
} ErrEndCatch
// You must structure your code exactly as
//above. You can't have an ErrTry without an
//ErrCatch { } ErrEndCatch, or vice versa.
```

---

Any call to [ErrThrow](#) within the ErrTry block results in control passing immediately to the ErrCatch block. Even if the subroutine CreateTable called ErrThrow, control would pass directly to the

## Using Palm OS System Managers

### *The Feature Manager*

---

ErrCatch block. If the ErrTry block completes without calling ErrThrow, the ErrCatch block is not executed.

You can nest multiple ErrTry blocks. For example, if you wanted to perform some cleanup at the end of CreateTable in case of error,

- Put ErrTry/ErrCatch blocks in CreateTable
- Clean up in the ErrCatch block first
- Call ErrThrow to jump to the top-level ErrCatch

## Error Manager Function Summary

The following error manager functions are available for application use:

- [ErrDisplay](#)
- [ErrDisplayFileLineMsg](#)
- [ErrFatalDisplayIf](#)
- [ErrNonFatalDisplayIf](#)
- [ErrThrow](#)

## The Feature Manager

A **feature** is a 32-bit value that has special meaning to both the feature publisher and to users of that feature. Features can be published by the system or by applications.

Each feature is identified by a feature creator and a feature number:

- The feature creator is usually the database creator type of the application that publishes the feature.
- The feature number is any 16-bit value used to distinguish between different features of a particular creator.

Once a feature is published, it remains present until it is explicitly deleted. A feature published by an application sticks around even after the application quits.

## The System Version Feature

An example for a feature is the system version. This feature is published by the system and contains a 32-bit representation of the system version. The system version has a feature creator of “psys” and a feature number of 1. Currently, the different versions of the system software have the following number:

The first version of the Palm OS system software has the following values

---

0x01003001	Pilot 1000 and Pilot 5000 (Palm OS 1.0)
0x02003000	PalmPilot and PalmPilot Professional (Palm OS 2.0)

---

Any application can find out the system version by looking for this feature.

## Application-Defined Features

When an application adds or removes capabilities from the base system, it can create features to test for the presence or absence of those capabilities. This allows an application to be compatible with multiple versions of the system by refining its behavior, depending on which capabilities are present or not. Future hardware platforms may lack some capabilities present in the first platform, so checking the system version feature is important.

This section introduces the feature manager by discussing these topics:

- [Using the Feature Manager](#)
- [Feature Manager Function Summary](#)

## Using the Feature Manager

Applications may find the feature manager useful for their own private use. For example, an application may want to publish a feature that contains a pointer to some private data it needs for processing launch codes. Because an application’s global data is not generally available while it processes launch codes, using the feature manager is usually the easiest way for an application to get to its data.

## Using Palm OS System Managers

### *The Feature Manager*

---

To check whether a particular feature is present, call [FtrGet](#) and pass it the feature creator and feature number. If the feature exists, [FtrGet](#) returns the 32-bit value of the feature. If the feature doesn't exist, an error code is returned.

To publish a new feature or change the value of an existing one, call [FtrSet](#) and pass the feature creator and number, and the 32-bit value of the feature. A published feature remains available until it is explicitly removed by a call to [FtrUnregister](#) or until the system resets; simply quitting an application doesn't remove a feature published by that application.

Features are split into two groups: ROM-based and RAM-based. ROM-based features are stored in a separate table in ROM and can never be removed; only system-defined features are in this table. All features installed at runtime are in the RAM table. [FtrGetByIndex](#) accepts a parameter that specifies whether to search the ROM table or RAM table.

Call [FtrUnregister](#) to remove RAM-based features created at runtime by calling [FtrSet](#).

You can get a complete list of all published features by calling [FtrGetByIndex](#) repeatedly. Passing an index value starting at 0 to [FtrGetByIndex](#) and incrementing repeatedly by 1 eventually returns all available features.

## Feature Manager Function Summary

The following feature manager functions are available for application use:

- [FtrGet](#)
- [FtrGetByIndex](#)
- [FtrSet](#)
- [FtrUnregister](#)



## The Sound Manager

The Palm OS sound manager lets applications and system modules play custom and predefined system sounds and control sound manager settings.

The sound manager provides an extendable API for playing custom sounds and system sounds, and for controlling default sound settings. Although the API accommodates multichannel design, only a single sound channel is currently supported. The user can control the alarm, system, and master sound amplitudes, typically using the Preferences application.

Currently supported system sounds are Information, Warning, Error, Startup, Alarm, Confirmation, and Click.

### Using the Sound Manager

To execute a sound manager command, call [SndDoCmd](#) and pass the sound channel pointer (presently, only null is supported and maps to the shared channel), a pointer to a structure of `SndCommandType`, and a flag indicating whether the command should be performed asynchronously. Asynchronous execution is not yet implemented; all commands execute synchronously.

To play a default system sound, such as a click or an error beep, call [SndPlaySystemSound](#), passing the system sound id. For the complete list of system sound IDs, see `SoundMgr.h`.

---

**Note:** All sound amplitudes greater than 0 are currently played as `MaxVolume`.

---

### Sound Manager Function Summary

The following sound manager functions are available for application use:

- [SndDoCmd](#)
- [SndGetDefaultVolume](#)
- [SndPlaySystemSound](#)
- [SndSetDefaultVolume](#)

## The String Manager

The string manager provides a set of string manipulation functions. The string manager API is closely modeled after the standard C string-manipulation functions like `strcpy`, `strcat`, etc.

Applications should use the functions built into the string manager instead of the standard C functions, because doing so makes the application smaller:

- When your application uses the string manager functions, the actual code that implements the function is not linked into your application but is already part of the operating system.
- When you use the standard C functions, the code for each function you use is linked into your application and results in a bigger executable.

In addition, many standard C functions don't work on the Palm OS device at all because the OS doesn't provide all basic system functions (such as `malloc`) and doesn't support the subroutine calls used by most standard C functions.

The following functions are available for application use:

- [StrAtOI](#)
- [StrCat](#)
- [StrCaselessCompare](#)
- [StrChr](#)
- [StrCompare](#)
- [StrCopy](#)
- [StrIToA](#)
- [StrIToH](#)
- [StrLen](#)
- [StrStr](#)
- [StrToLower](#)

## The System Manager

The Palm OS system manager is responsible for the general operation of the system, including boot-up, power-up, launching applications, library management, monitoring the battery, multitasking, timing, and semaphore support. Applications need to be concerned with very few system manager API functions. Most of what the system manager does is transparent to applications and is explained here as background information only.

In this section, you learn about the following aspects of the system manager:

- [System Boot and Reset](#) — information about the different reset operations, including system reset calls
- [Power Management](#) — the three different power modes and guidelines for application developers
- [The Microkernel](#) — basic task management provided by the system
- [Application Support](#) — event processing and interapplication communication from the system's point of view
- [System Manager Functions](#) — list of all system manager functions available to applications

### System Boot and Reset

The system manager provides support for booting the Palm OS device. Booting occurs only when the user presses the reset switch on the device (see “Palm OS Device Reset Switch” in *Developing Palm OS Applications, Part I*). Palm OS differs from a traditional desktop system in that it's never really turned off. Power is constantly supplied to essential subsystems and the on/off key is merely a way of bringing the device in or out of low-power mode (see [Palm OS Power Modes](#)). The obvious effect of pressing the on/off key is that the LCD turns on or off. When the user presses the power key to turn the device off, the LCD is disabled, which makes it appear as if power to the entire unit is turned off. In fact, the memory system, real-time clock, and interrupt generation circuitry are still running, though they are consuming little current.

## Using Palm OS System Managers

### *The System Manager*

---

In this version of Palm OS, there is only one user interface application running at a time. The User Interface Application Shell (UIAS) is responsible for managing the current user-interface application. The UIAS launches the current user-interface application as a subroutine and doesn't get control back until that application quits. When control returns to the UIAS, the UIAS immediately launches the next application as another subroutine. See [Power Management Calls](#) for more information.

### System Reset Calls

The system calls [SysReset](#) to reset the device. This call does a soft reset and has the same effect as pressing the reset switch on the unit. **Normally, applications should not use this call.**

`SysReset` is used, for example, by the Sync application. When the user copies an extension onto the Palm OS device, the Sync application automatically resets the device after the sync is completed to allow the extension to install itself.

The `SysColdBoot` call is similar, but even more dangerous. It performs a hard reset that clears all user storage RAM on the device, destroying all user data.

## Power Management

This section looks at Palm OS power management, discussing the following topics:

- [Palm OS Power Modes](#)
- [Guidelines for Application Developers](#)
- [Power Management Calls](#)

### Palm OS Power Modes

At any time, the Palm OS device is in one of three power modes: sleep, doze, or running. The system manager controls transitions between different power modes and provides an API for controlling some aspects of the power management.

- **Sleep mode.** If the unit appears to be off, it is actually in sleep mode and is consuming as little current as possible. At this rate, a unit could sit for almost a year on a single set of

batteries without losing the contents of memory. To enter sleep mode, the system puts as many peripherals as possible into low-power mode and sets up the hardware so that an interrupt from any hard key or the real-time clock wakes up the system.

When the system gets one of these interrupts while in sleep mode, it quickly checks that the battery is strong enough to complete the wake-up and then takes each of the peripherals, for example, the LCD, serial port, and timers, out of low-power mode.

The system reenters sleep mode when the user presses the on/off key again, when the system has been idle for the minimum auto-off time, or when the battery level reaches a critically low level.

- **Doze mode.** In doze mode, the processor is halted, but all peripherals including the LCD are powered up. The system can come out of doze mode much faster than it can come out of sleep mode since none of the peripherals need to be woken up. In fact, it takes no longer to come out of doze mode than to process an interrupt. Usually, when the system appears on, it is actually in doze mode and goes into running mode only for short periods of time to process an interrupt or respond to user input like a pen tap or key press.
- **Running mode.** Running means that the processor is executing instructions and all peripherals are powered up. A typical application puts the system into running mode only about 5% of the time.

### Guidelines for Application Developers

Normally, applications don't need to be aware of power management except for a few simple guidelines. When an application calls [EvtGetEvent](#) to ask the system for the next event to process, the system automatically puts itself into doze mode until there is an event to process. As long as an application uses `EvtGetEvent`, power management occurs automatically. If there has been no user input for the amount of time determined by the current setting of the auto-off preference, the system automatically enters sleep mode without intervention from the application.

Applications should avoid providing their own delay loops. Instead, they should use [SysTaskDelay](#), which puts the system into doze mode during the delay to conserve as much power as possible.

## Using Palm OS System Managers

### *The System Manager*

---

If an application needs to perform periodic work, it can pass a time out to [EvtGetEvent](#); this forces the unit to wake up out of doze mode and to return to the application when the time out expires, even if there is no event to process. Using these mechanisms provides the longest possible battery life.

### **Power Management Calls**

The system calls `SysSleep` to put itself immediately into low-power sleep mode. Normally, the system puts itself to sleep when there has been no user activity for the minimum auto-off time or when the user presses the power key.

The [SysSetAutoOffTime](#) routine changes the auto-off time value. This routine is normally used by the system only during boot, and by the Preferences application. The Preferences application saves the user preference for the auto-off time in a preferences database, and the system initializes the auto-off time to the value saved in the preferences database during boot. While the auto-off feature can be disabled entirely by calling `SysSetAutoOffTime` with a time-out of 0, doing this depletes the battery.

The current battery level and other information can be obtained through the [SysBatteryInfo](#) routine. This call returns information about the battery, including the current battery voltage in hundredths of a volt, the warning thresholds for the low-battery alerts, the battery type, and whether external power is applied to the unit. This call can also change the battery warning thresholds and battery type.

### **The Microkernel**

Palm OS has a preemptive multitasking kernel that provides basic task management.

Most applications don't need the microkernel services because they are handled automatically by the system. This functionality is provided mainly for internal use by the system software or for certain special purpose applications.

The User Interface Application Shell (UIAS) is responsible for managing the current user-interface application, as described in [System Boot and Reset](#).

Usually, the UIAS is the only task running. Occasionally though, an application launches another task as a part of its normal operation. One example of this is the Sync application, which launches a second task to handle the serial communication with the desktop. The Sync application creates a second task dedicated to the serial communication and gives this task a lower priority than the main user-interface task. The result is optimal performance over the serial port without a delay in response to the user-interface controls.

Normally, there is no user interaction during a sync, so that the serial communication task gets all of the processor's time. However, if the user does tap on the screen, for example, to cancel the sync, the user-interface task immediately processes the tap, since it has a higher priority. Alternatively, the Sync application could have been written to use just one task, but then it would have to periodically poll for user input during the serial communication, which would hamper performance and user-interface response time.

### Application Support

The system manager provides application support in several functional areas. The following aspects of application support are discussed in this section:

- [Launching and Cleanup](#)
- [Event Processing](#)
- [Interapplication Communication](#)
- [Application Utilities](#)

#### Launching and Cleanup

Usually, applications on the Palm OS device are launched when the user presses one of the buttons on the case or selects an application icon from the application launcher screen. Alternatively, an application can programmatically launch another application by using the system manager function [SysAppLaunch](#).

When the current user-interface application quits, the system manager cleans up by deleting any chunks in the dynamic heap(s) that the application left around and closing any databases left open. Note, however, that applications should perform those kinds of cleanup tasks themselves.

## Using Palm OS System Managers

### *The System Manager*

---

#### Event Processing

The system manager provides the infrastructure for event generation and also contains the support for handling most system-related events. Hardware activity, such as taps on the digitizer and key presses, is interpreted by interrupt handlers of the system manager and converted into events that are eventually sent to the application through the `EvtGetEvent` call. In addition, many events returned by `EvtGetEvent` are system-related events that can be processed by the system manager call `SysHandleEvent`.

Events in Palm OS include hardware- and software-generated events. The following table provides an overview:

Hardware-generated events	Software-generated events
<u>Caused</u> directly by user interaction with the device, such as tapping on the screen with the pen, or pressing a hardware button.	<u>Generated</u> by the system software as a side effect of a user interaction.
<u>Include</u> pen-downs, pen-ups (optionally including stroke data), and hard button presses.	<u>Include</u> events like the quit event that causes an application to exit, or keyboard events generated by the Graffiti recognizer. Applications can define software-generated events for their own use.
Typically <u>posted</u> by interrupt routines.	Typically <u>posted</u> as the result of a system call. Include application-quit events, window-enter and window-exit events, user-interface control events, etc.
<ul style="list-style-type: none"><li>• Pen-generated events are <u>stored</u> in the pen queue.</li><li>• Hard button press events are <u>stored</u> in the key queue.</li></ul>	<u>Stored</u> in the software event queue.

When `EvtGetEvent` is called by the application, it first checks whether any events are in the software event queue and returns the topmost event if so.



If the software event queue is empty, `EvtGetEvent` checks the key and pen queues. The result is that all software events generated by a particular hardware event are processed before the next hardware event is processed. For example, a pen-down hardware event may trigger the system software to generate window-exit and window-enter software events. Both events are then pulled from the software event queue and processed before the next hardware event is processed.

Some event types returned by `EvtGetEvent` are not actually posted into the event queue, but are artificially generated by `EvtGetEvent` when all event queues are empty. One example is the pen-moved event, which is returned if no other events are in the queues and the pen has moved since the last time `EvtGetEvent` was called. In this way, the application is notified of low-priority events, such as pen movements, but the event queue isn't cluttered with them.

In a typical application, `SysHandleEvent` is called immediately after `EvtGetEvent`. If `EvtGetEvent` returns a pen-up event in the Graffiti writing area, `SysHandleEvent` calls the Graffiti recognizer with the pen stroke information obtained from the pen queue and uses the results of the Graffiti recognizer to post one or more keyboard events into the key queue. A similar process occurs for pen-up events detected over a silk-screened icon. `SysHandleEvent` converts the pen-up to a keyboard event with a virtual key code representing the silk-screened icon.

When an application calls `EvtGetEvent`, the event manager checks a number of system-event data structures and returns an event record to the application with information about the highest-priority event that needs processing. Events in Palm OS are stored in one of three event queues: a key queue, a pen queue, or a software event queue. The event queues are circular buffers containing event records stored in a first-in, first-out (FIFO) sequence.

Here's some additional information on hardware and software events:

- **Hardware events** are posted into their appropriate event queue by interrupt routines. The interrupt routine for handling keyboard presses immediately enqueues the keyboard event into the key queue and sets up a periodic interrupt routine to watch for auto-repeat and for key debouncing.

- **Software-generated** events include window-enter and window-exit events, application quit events, and user-interface object events like control enter, control exit, etc. These events are typically generated as a side effect of a hardware-generated event like a pen-down. Software can, however, also generate key events, usually as a result of recognizing a Graffiti stroke or a tap on a silk-screened icon.

Software-generated events are posted into the appropriate event queue, but are not typically posted at interrupt time. Many of these events are inserted into the event queue by the various user-interface managers. Others, like key events, are posted by `SysHandleEvent` after recognizing a Graffiti stroke or a tap on a silk-screened icon.

### Interapplication Communication

The system manager provides the API for interapplication communication. This API permits any application or system routine to send a **launch code** to any other application and get results back. For example, an application that is to work with the global find must support the find launch code.

Sending a launch code to another application is like calling a specific subroutine in that application: the application responding to the launch code is responsible for determining what to do given the launch code constant passed on the stack as a parameter.

Predefined launch codes are listed in “Developing Palm OS Applications, Part I” and can be found in `SystemMgr.h`. All the parameters for a launch code are passed in a single parameter block, and the results are returned in the same parameter block. “How Launch Codes Control an Application” in “Developing Palm OS Applications, Part I, describes launch codes in more detail.

### Application Utilities

The [SysHandleEvent](#) call allows applications to correctly respond to system events like key presses, Graffiti strokes, low-battery warnings, and taps on silk-screened icons. Every application should call this routine from its event loop, usually before the application even looks at the event. If an application needs to override any part of the default system behavior, it could selectively filter out events before calling [SysHandleEvent](#).

An application can force a switch to another user-interface application by calling [SysUIAppSwitch](#). This routine notifies the system which application to launch next and feeds an application-quit event into the event queue. If and when the current application responds to the quit event and returns, the system launches the new application.

Use the routine [SysCurAppDatabase](#) to get the card number and database ID of the currently running user-interface application. If your application code is called to process a launch code, it essentially is called as a subroutine from the current user-interface application. This routine doesn't return your application's database ID but the database ID of the application that initiated the launch code.

The routine [SysAppLaunch](#) is a general-purpose launch facility for launching any resource database with executable code in it. It has numerous options, including whether or not to launch the database as a separate task, whether to allocate a globals world, and whether or not to give the database its own stack. This routine is also used to send launch codes to applications (by telling it to use the caller's stack, no globals world, and not a separate task). Usually, applications use it only for sending launch codes to other user-interface applications. An alternative, simpler method of sending launch codes is the [SysBroadcastActionCode](#) call. This routine automatically finds all other user-interface applications and calls [SysAppLaunch](#) to send the launch code to each of them.

## System Manager Functions

The following system manager functions are available for application use:

- [SysReset](#)
- [SysBatteryInfo](#)
- [SysSetAutoOffTime](#)
- [SysHandleEvent](#)
- [SysUIAppSwitch](#)
- [SysCurAppDatabase](#)
- [SysBroadcastActionCode](#)
- [SysAppLaunch](#)

## The System Event Manager

The system event manager

- Manages the low-level pen and key event queues.
- Translates taps on silk-screened icons into key events.
- Sends pen strokes in the Graffiti area to the Graffiti recognizer.
- Puts the system into low-power doze mode when there is no user activity.

Most applications have no need to call the system event manager directly because most of the functionality they need comes from the higher-level event manager or is automatically handled by the system.

Applications that do use the system event manager directly might do so to enqueue key events into the key queue or to retrieve each of the pen points that comprise a pen stroke from the pen queue.

This section provides information about the system event manager by discussing these topics:

- [Event Translation: Pen Strokes to Key Events](#)
- [Pen Queue Management](#)
- [Auto-Off Control](#)
- [System Event Manager Function Summary](#)

## Event Translation: Pen Strokes to Key Events

One of the higher-level functions provided by the system event manager is conversion of pen strokes on the digitizer to key events. For example, the system event manager sends any stroke in the Graffiti area of the digitizer automatically to the Graffiti recognizer for conversion to a key event. Taps on silk-screened icons, such as the application launcher, Menu button, and Find button, are also intercepted by the system event manager and converted into the appropriate key events.

When the system converts a pen stroke to a key event, it:

- Retrieves all pen points that comprise the stroke from the pen queue
- Converts the stroke into the matching key event
- Enqueues that key event into the key queue

Eventually, the system returns the key event to the application as a normal result of calling [EvtGetEvent](#).

Most applications rely on the following default behavior of the system event manager:

- All strokes in the predefined Graffiti area of the digitizer are converted to key events
- All taps on the silk-screened icons are convert to key events
- All other strokes are passed on to the application for processing

## Pen Queue Management

The pen queue is a preallocated area of system memory used for capturing the most recent pen strokes on the digitizer. It is a circular queue with a first-in, first-out method of storing and retrieving pen points. Points are usually enqueued by a low-level interrupt routine and dequeued by the system event manager or application.

## Using Palm OS System Managers

### *The System Event Manager*

---

The following table summarizes pen management.

The user...	The system...
Brings the pen down on the digitizer.	Stores a pen-down sequence in the pen queue and starts the stroke capture.
Draws a character.	Stores additional points in the pen queue periodically.
Lifts the pen.	Stores a pen-up sequence in the pen queue and turns off stroke capture.

The system event manager provides an API for initializing and flushing the pen queue and for queuing and dequeuing points. Some state information is stored in the queue itself: to dequeue a stroke, the caller must first make a call to dequeue the stroke information ([EvtDequeuePenStrokeInfo](#)) before the points for the stroke can be dequeued. Once the last point is dequeued, another [EvtDequeuePenStrokeInfo](#) call must be made to get the next stroke.

Applications usually don't need to call `EvtDequeuePenStrokeInfo` because the event manager calls this function automatically when it detects a complete pen stroke in the pen queue. After calling `EvtDequeuePenStrokeInfo`, the system event manager stores the stroke bounds into the event record and returns the pen-up event to the application. The application is then free to dequeue the stroke points from the pen queue, or to ignore them altogether. If the points for that stroke are not dequeued by the time [EvtGetEvent](#) is called again, the system event manager automatically flushes them.

## Key Queue Management

The key queue is an area of system memory preallocated for capturing key events. Key events come from one of two occurrences:

- As a direct result of the user pressing one of the buttons on the case
- As a side effect of the user drawing a Graffiti stroke on the digitizer, which is converted in software to a key event

The following table summarizes key management:

<b>User action</b>	<b>System response</b>
Hardware button press.	Interrupt routine enqueues the appropriate key event into the key queue, temporarily disables further hardware button interrupts, and sets up a timer task to run every 10 ms.
Hold down key for extended time period.	Timer task to supports auto-repeat of the key (timer task is also used to debounce the hardware).
Release key for certain amount of time.	Timer task reenables the hardware button interrupts.
Pen stroke in Graffiti area of digitizer.	System manager calls the Graffiti recognizer, which then removes the stroke from the pen queue, converts the stroke into one or more key events, and finally enqueues these key events into the key queue.
Pen stroke on silk-screened icons.	System event manager converts the stroke into the appropriate key event and enqueues it into the key queue.

The system event manager provides an API for initializing and flushing the key queue and for enqueueing and dequeuing key events. Usually, applications have no need to dequeue key events; the event manager does this automatically if it detects a key in the queue and returns a `keyDownEvent` (documented in “Developing Palm OS Applications,” Part I) to the application through the [EvtGetEvent](#) call.

## **Auto-Off Control**

Because the system event manager manages hardware events like pen taps and hardware button presses, it’s responsible for resetting the auto-off timer on the device. Whenever the system detects a hardware event, it automatically resets the auto-off timer to 0. If an application needs to reset the auto-off timer manually, it can do so through the system event manager call [EvtResetAutoOffTimer](#).

### System Event Manager Function Summary

The following functions are part of the developer API to the system event manager:

- [EvtAddEventToQueue](#)
- [EvtCopyEvent](#)
- [EvtDequeuePenPoint](#)
- [EvtDequeuePenStrokeInfo](#)
- [EvtEnableGraffiti](#)
- [EvtEnqueueKey](#)
- [EvtFlushKeyQueue](#)
- [EvtFlushNextPenStroke](#)
- [EvtFlushPenQueue](#)
- [EvtGetEvent](#)
- [EvtGetPen](#)
- [EvtKeyQueueEmpty](#)
- [EvtKeyQueueSize](#)
- [EvtKeyQueueEmpty](#)
- [EvtGetPenBtnList](#)
- [EvtPenQueueSize](#)
- [EvtProcessSoftKeyStroke](#)
- [EvtResetAutoOffTimer](#)
- [EvtWakeup](#)



## The Time Manager

The date and time manager (called time manager in this chapter) provides access to both the 1-second and 0.01-second timing resources on the Palm OS device.

- The 1-second timer keeps track of the real-time clock (date and time), even when the unit is in sleep mode.
- The 0.01-second timer, also referred to as the **system ticks**, can be used for finer timing tasks. This timer is not updated when the unit is in sleep mode and is reset to 0 each time the unit resets.

The basic time-manager API provides support for setting and getting the real-time clock in seconds and for getting the current system ticks value (but not for setting it). The system manager provides more advanced functionality for setting up a timer task that executes periodically or in a given number of system ticks.

This section discusses the following topics:

- [Using Real-Time Clock Functions](#)
- [Using System Ticks Functions](#)
- [Time Manager Function Summary](#)

### Using Real-Time Clock Functions

The real-time clock functions of the time manager include [TimSetSeconds](#) and [TimGetSeconds](#). Real time on the Palm OS device is measured in seconds from midnight, Jan 1, 1904. Call [TimSecondsToDateTime](#) and [TimDateTimeToSeconds](#) to convert between seconds and a structure specifying year, month, day, hour, minute, and second.

### Using System Ticks Functions

The Palm OS device maintains a tick count that starts at 0 when the device is reset. This tick increments

- 100 times per second when running on the Palm OS device
- 60 times per second when running on the Macintosh under the Simulator

## Using Palm OS System Managers

### *The Time Manager*

---

For tick-based timing purposes, applications should use the macro `sysTicksPerSecond`, which is conditionally compiled for different platforms. Use the function [TimGetTicks](#) to read the current tick count.

Although the `TimGetTicks` function could be used in a loop to implement a delay, it is recommended that applications use the `SysTaskDelay` function instead. The `SysTaskDelay` function automatically puts the unit into low-power mode during the delay. Using `TimGetTicks` in a loop consumes much more current.

## Time Manager Structures

The time manager uses these structures to store information.

---

### Listing 1.2 Time Manager Structures

---

```
typedef struct{
    Sword second;
    Sword minute;
    Sword hour;
    Sword day;
    Sword month;
    Sword year;
    Sword weekDay;          //Days since Sunday (0 to 6)
}DateTimeType;
typedef DateTimeType* DateTimePtr;

typedef struct {
    Byte hours;
    Byte minutes;
}TimeType;
typedef TimeType * TimePtr;

typedef struct{
    Word year :7; //years since 1904 (Mac format)
    Word month:4;
    Word day :5;
}DateType;
typedef DateType * DatePtr;
```

---

## Time Manager Function Summary

The following time manager functions are available for application use:

- [DateAdjust](#)
- [DateDaysToDate](#)
- [DateSecondsToDate](#)
- [DateToAscii](#)
- [DateToDays](#)
- [DateToDOWDMFormat](#)
- [DayOfMonth](#)
- [DayOfWeek](#)
- [DaysInMonth](#)
- [TimAdjust](#)
- [TimDateTimeToSeconds](#)
- [TimGetSeconds](#)
- [TimGetTicks](#)
- [TimSecondsToDateTime](#)
- [TimSetSeconds](#)
- [TimeToAscii](#)

Note that two functions associated with the Date and Time object, `SelectDay` and `SelectTime` are documented in *Developing Palm OS Applications Part I*.

## Using Palm OS System Managers

*The Time Manager*

---



# Palm OS System Functions

---

## Alarm Manager API

### AlmGetAlarm

**Purpose** Return the alarm date/time in seconds since 1/1/1904 and the caller-defined alarm reference value for the given application.

**Prototype** `ULong AlmGetAlarm (   UInt cardNo,  
                          LocalID dbID,  
                          DWordPtr refP)`

**Parameters**

- > cardNo   Storage card number of the application.
- > dbID     Local ID of the application.
- <-> refP    Pointer to location for the alarm's reference value.

**Result** Alarm 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.

## AlmSetAlarm

**Purpose** Set or cancel an alarm for the given application.

**Prototype** `Err AlmSetAlarm (    UInt cardNo,  
                          LocalID dbID,  
                          DWord ref,  
                          ULong alarmSeconds,  
                          Boolean quiet)`

**Parameters**

-> cardNo	Storage card number of the application.
-> dbID	Local ID of the application.
-> ref	Caller-defined value to be passed with notifications.
-> alarmSeconds	Alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).
-> quiet	Reserved for future upgrade (set to zero).

**Result**

0	No error.
almErrMemory	Insufficient memory.
almErrFull	Alarm table is full.

**Comments** If an alarm for this application has already been set, it is replaced with the new alarm. Action code notifications are sent after the alarm is triggered and can be used by the application to set the next alarm.

## Functions for System Use Only

### **AlmAlarmCallback**

**Prototype**    `void    AlmAlarmCallback (void)`

---

WARNING: This function for use by system software only.

---

### **AlmCancelAll**

**Prototype**    `void    AlmCancelAll (Boolean enable)`

---

WARNING: This function for use by system software only.

---

### **AlmDisplayAlarm**

**Prototype**    `void    AlmDisplayAlarm (Boolean displayOnly)`

---

WARNING: This function for use by system software only.

---

### **AlmEnableNotification**

**Prototype**    `void AlmEnableNotificatio(Boolean enable)`

---

WARNING: This function for use by system software only.

---

### **AlmInit**

**Prototype**    `Err    AlmInit (void)`

---

WARNING: This function for use by system software only.

---

## Error Manager Functions

### ErrDisplay

**Purpose** Display an error alert if error checking is set to partial or full.

**Prototype** `void ErrDisplay (char* message)`

**Parameters** -> message      Error message text.

**Result** No return value.

**Comments** Call this routine to display an error message, source code filename, and line number. This routine is actually a macro that 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](#), ["Using the Error Manager Macros."](#)



## **ErrDisplayFileLineMsg**

**Purpose** Display a nonexitable dialog with an error message. Do not allow the user to continue.

**Prototype** `void ErrDisplayFileLineMsg( CharPtr filename,  
                                  UInt lineno,  
                                  CharPtr msg)`

<b>Parameters</b>	<code>filename</code>	Source code filename.
	<code>lineno</code>	Line number in the source code file.
	<code>msg</code>	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.

**See Also** [ErrFatalDisplayIf](#), [ErrNonFatalDisplayIf](#), [ErrDisplay](#)

## ErrFatalDisplayIf

**Purpose** Display an error alert dialog if `condition` is `TRUE` and error checking is set to partial or full.

**Prototype** `void ErrFatalDisplayIf ( Boolean condition,  
char* message)`

**Parameters**

-> <code>condition</code>	If <code>TRUE</code> , display the error.
-> <code>message</code>	Error message text.

**Result** No return value.

**Comments** Call this routine 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 routine is actually a macro that 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](#), ["Using the Error Manager Macros."](#)

## **ErrNonFatalDisplayIf**

**Purpose** Display an error alert dialog if `condition` is `TRUE` and error checking is set to full.

**Prototype** `void ErrNonFatalDisplayIf ( Boolean condition,  
char* message)`

**Parameters**

-> <code>condition</code>	If <code>TRUE</code> , display the error.
-> <code>message</code>	Error message text.

**Result** No return value.

**Comments** Call this routine 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 routine is actually a macro that is compiled into the code only if the compiler define `ERROR_CHECK_LEVEL` is set to 2 (`ERROR_CHECK_FULL`).

**See Also** [ErrFatalDisplayIf](#), [ErrDisplay](#), ["Using the Error Manager Macros."](#)

## ErrThrow

**Purpose** Cause a jump to the nearest Catch block.

**Prototype** void ErrThrow (Long 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](#), ["Using the Error Manager Macros."](#)

## Event Manager Functions

### **EvtAddEventToQueue**

<b>Purpose</b>	Add an event to the event queue.				
<b>Prototype</b>	<code>void EvtAddEventToQueue (EventPtr event)</code>				
<b>Parameters</b>	<table><tr><td><code>event</code></td><td>Pointer to the structure that contains the event.</td></tr><tr><td><code>error</code></td><td>Pointer to any error encountered by this function.</td></tr></table>	<code>event</code>	Pointer to the structure that contains the event.	<code>error</code>	Pointer to any error encountered by this function.
<code>event</code>	Pointer to the structure that contains the event.				
<code>error</code>	Pointer to any error encountered by this function.				
<b>Result</b>	Returns nothing.				

### **EvtAddUniqueEventToQueue**

<b>Purpose</b>	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. <ul style="list-style-type: none"><li>• If no existing event is found, the new event is added.</li><li>• If an existing event is found, the routine proceeds as follows:<ul style="list-style-type: none"><li>– if <code>inPlace</code> is <code>TRUE</code>, the existing event is replaced with the new event</li><li>– if <code>inPlace</code> is <code>FALSE</code>, the existing event is removed and the new event will be added to the end</li></ul></li></ul>						
<b>Prototype</b>	<code>void EvtAddUniqueEventToQueue ( EventPtr eventP, DWord id, Boolean inPlace)</code>						
<b>Parameters</b>	<table><tr><td><code>eventP</code></td><td>Pointer to the structure that contains the event</td></tr><tr><td><code>id</code></td><td>ID of event. 0 means match only on the type.</td></tr><tr><td><code>inPlace</code></td><td>If <code>TRUE</code>, existing event are replaced. If <code>FALSE</code>, existing event is deleted and new event added to end of queue.</td></tr></table>	<code>eventP</code>	Pointer to the structure that contains the event	<code>id</code>	ID of event. 0 means match only on the type.	<code>inPlace</code>	If <code>TRUE</code> , existing event are replaced. If <code>FALSE</code> , existing event is deleted and new event added to end of queue.
<code>eventP</code>	Pointer to the structure that contains the event						
<code>id</code>	ID of event. 0 means match only on the type.						
<code>inPlace</code>	If <code>TRUE</code> , existing event are replaced. If <code>FALSE</code> , existing event is deleted and new event added to end of queue.						
<b>Result</b>	Returns nothing.						

#### **EvtCopyEvent**

**Purpose** Copy an event.

**Prototype** `void EvtCopyEvent (EventPtr source, EventPtr dest)`

**Parameters**

<code>source</code>	Pointer to the structure containing the event to copy.
<code>dest</code>	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**

<code>retP</code>	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**

<b>Purpose</b>	Initiate the extraction of a stroke from the pen queue.
<b>Prototype</b>	<pre>Err EvtDequeuePenStrokeInfo( PointType* startPtP,                              PointType* endPtP)</pre>
<b>Parameters</b>	<p><code>startPtP</code>    Start point returned here.</p> <p><code>startPtP</code>    End point returned here.</p>
<b>Result</b>	Always returns 0.
<b>Comments</b>	<p>Called by the system function <code>EvtGetSysEvent</code>. This routine must be called before <a href="#">EvtDequeuePenPoint</a> is called.</p> <p>Subsequent calls to <a href="#">EvtDequeuePenPoint</a> return points at the starting point in the stroke and including the end point. After the end point is returned, the next call to <a href="#">EvtDequeuePenPoint</a> returns the point -1, -1.</p>
<b>See Also</b>	<a href="#">EvtDequeuePenPoint</a>

## **EvtEnableGraffiti**

<b>Purpose</b>	Set Graffiti enabled or disabled.
<b>Prototype</b>	<pre>void EvtEnableGraffiti (Boolean enable)</pre>
<b>Parameters</b>	<p><code>enable</code>        TRUE to enable Graffiti, FALSE to disable Graffiti.</p>
<b>Result</b>	Returns nothing.

## EvtEnqueueKey

**Purpose** Place keys into the key queue.

**Prototype** `Err EvtEnqueueKey (    UInt ascii,  
                              UInt keycode,  
                              UInt 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-Keys 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:

<code>evtKeyStringEscape</code>	1 byte
ASCII code	2 bytes
virtual key code	2 bytes
modifiers	2 bytes



### **EvtEventAvail**

<b>Purpose</b>	Return TRUE if an event is available.
<b>Prototype</b>	Boolean EvtEventAvail (void)
<b>Parameters</b>	None
<b>Result</b>	Returns TRUE if an event is available, FALSE otherwise.

### **EvtFlushKeyQueue**

<b>Purpose</b>	Flush all keys out of the key queue.
<b>Prototype</b>	Err EvtFlushKeyQueue (void)
<b>Parameters</b>	None.
<b>Result</b>	Always returns 0.
<b>Comments</b>	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 dequeuing. 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.

**Comment** Called by the system function `EvtSetKeyQueuePtr`.

**See Also** [EvtPenQueueSize](#)

## **EvtGetEvent**

<b>Purpose</b>	Return the next available event.				
<b>Prototype</b>	<code>void EvtGetEvent (EventPtr event, Long timeout)</code>				
<b>Parameters</b>	<table><tr><td><code>event</code></td><td>Pointer to the structure to hold the event returned.</td></tr><tr><td><code>timeout</code></td><td>Maximum number of ticks to wait before an event is returned (-1 means wait indefinitely).</td></tr></table>	<code>event</code>	Pointer to the structure to hold the event returned.	<code>timeout</code>	Maximum number of ticks to wait before an event is returned (-1 means wait indefinitely).
<code>event</code>	Pointer to the structure to hold the event returned.				
<code>timeout</code>	Maximum number of ticks to wait before an event is returned (-1 means wait indefinitely).				
<b>Comments</b>	Pass <code>timeout = -1</code> 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 <code>timeout &gt;= 0</code> .				
<b>Result</b>	Returns nothing.				

## **EvtGetPen**

<b>Purpose</b>	Return the current status of the pen.						
<b>Prototype</b>	<code>void EvtGetPen(   Sword *pScreenX,                   Sword *pScreenY,                   Boolean *pPenDown)</code>						
<b>Parameters</b>	<table><tr><td><code>pScreenX</code></td><td>x location relative to display.</td></tr><tr><td><code>pScreenY</code></td><td>y location relative to display.</td></tr><tr><td><code>pPenDown</code></td><td>TRUE or FALSE.</td></tr></table>	<code>pScreenX</code>	x location relative to display.	<code>pScreenY</code>	y location relative to display.	<code>pPenDown</code>	TRUE or FALSE.
<code>pScreenX</code>	x location relative to display.						
<code>pScreenY</code>	y location relative to display.						
<code>pPenDown</code>	TRUE or FALSE.						
<b>Result</b>	Returns nothing.						
<b>Comments</b>	Called by various UI routines.						
<b>See Also</b>	<code>KeyCurrentState</code> (documented in <i>Developing Palm OS Applications, Part I</i> )						

#### EvtGetPenBtnList

**Purpose** Return a pointer to the silk-screen button array.

**Prototype** `PenBtnInfoPtr asm  
EvtGetPenBtnList( UIntPtr 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](#)

#### 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**

<b>Purpose</b>	Return the size of the current key queue in bytes.
<b>Prototype</b>	ULong EvtKeyQueueSize (void)
<b>Parameters</b>	None.
<b>Result</b>	Returns size of queue in bytes.
<b>Comments</b>	Called by applications that wish to see how large the current key queue is.

## **EvtPenQueueSize**

<b>Purpose</b>	Return the size of the current pen queue in bytes.
<b>Prototype</b>	ULong EvtPenQueueSize (void)
<b>Parameters</b>	None.
<b>Result</b>	Returns size of queue in bytes.
<b>Comments</b>	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` (documented in *Developing Palm OS Applications, Part I*)

## 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 `SerialLinkMgr`, Can be called periodically by other managers.

**See Also** [SysSetAutoOffTime](#)

## **EvtSysEventAvail**

<b>Purpose</b>	Return TRUE if a low-level system event (such as a pen or key event) is available.		
<b>Prototype</b>	<code>Boolean EvtSysEventAvail(Boolean ignorePenUps)</code>		
<b>Parameters</b>	<table><tr><td><code>ignorePenUps</code></td><td>If TRUE, this routine ignores pen-up events when determining if there are any system events available.</td></tr></table>	<code>ignorePenUps</code>	If TRUE, this routine ignores pen-up events when determining if there are any system events available.
<code>ignorePenUps</code>	If TRUE, this routine ignores pen-up events when determining if there are any system events available.		
<b>Result</b>	Returns TRUE if a system event is available.		
<b>Comment</b>	Call <a href="#">EvtEventAvail</a> to determine whether high-level software events are available.		

## **EvtWakeup**

<b>Purpose</b>	Force the event manager to wake up and send a <code>nilEvent</code> to the current application. Events are documented in “ <i>Developing Palm OS Applications, Part I</i> ”).
<b>Prototype</b>	<code>Err EvtWakeup (void)</code>
<b>Parameters</b>	None.
<b>Result</b>	Always returns 0.
<b>Comments</b>	Called by interrupt routines, like the sound manager and alarm manager.

## EvtDequeueKeyEvent

**WARNING: System Use Only!**

**WARNING: System Use Only!**

**WARNING: System Use Only!**

**WARNING: System Use Only!**

**WARNING: System Use Only!**

## 64 Developing Palm OS 2.0 Applications, Part II



---

WARNING: System Use Only!

---

**EvtSysInit**

**Prototype**    `Err EvtSysInit (void)`

---

WARNING: System Use Only!

---

## Feature Functions

### FtrGet

**Purpose** Get a feature.

**Prototype** `Err FtrGet ( DWord creator,  
                  UInt featureNum,  
                  DWordPtr valueP)`

**Parameters**

<code>creator</code>	Creator type, should be same as the application that owns this feature.
<code>featureNum</code>	Feature number of the feature.
<code>valueP</code>	Value of the feature is returned here.

**Result** Returns 0 if no error, or `ftrErrNoSuchFtr` or `ftrErrInternalError` if an error occurs.

**Comments** The value of the feature is application-dependent.

**See Also** [FtrSet](#)

## **FtrGetByIndex**

**Purpose** Get a feature by index.  
Until the caller gets back `ftrErrNoSuchFeature`, it should pass indices for each table (ROM, RAM) starting at 0 and incrementing .

**Prototype** `Err FtrGetByIndex (    UInt index,  
                                Boolean romTable,  
                                DWordPtr creatorP,  
                                UIntPtr numP,  
                                DWordPtr valueP)`

<b>Parameters</b>	<code>index</code>	Index of feature.
	<code>romTable</code>	If TRUE, index into ROM table; otherwise, index into RAM table.
	<code>creatorP</code>	Feature creator is returned here.
	<code>numP</code>	Feature number is returned here.
	<code>valueP</code>	Feature value is returned here.

**Result** Returns 0 if no error, or `ftrErrInternalError` or `ftrErrNoSuchFeature` if an error occurs.

**Comments** This routine is normally only used by shell commands. Most applications don't need it.

## FtrSet

**Purpose** Set a feature.

**Prototype** `Err FtrSet ( DWord creator,  
                  UInt featureNum,  
                  DWord newValue)`

**Parameters**

<code>creator</code>	Creator type, should be same as the application that owns this feature.
<code>featureNum</code>	Feature number of the feature.
<code>newValue</code>	New value.

**Result** Returns 0 if no error, or `ftrErrNoSuchFeature`, `memErrChunkLocked`, `memErrInvalidParam`, or `memErrNotEnoughSpace` if an error occurs.

**Comments** The value of the feature is application-dependent.

**See Also** [FtrGet](#)

## **FtrUnregister**

**Purpose** Unregister a feature.

**Prototype** `Err FtrUnregister (DWord creator,  
                          UInt featureNum)`

**Parameters**

<code>creator</code>	Creator type, should be same as the application that owns the creator.
<code>featureNum</code>	Feature number of the feature.

**Result** Returns 0 if no error, or `ftrInternalError`, `ftrErrNoSuchFeature`, `memErrChunkLocked`, `memErrInvalidParam`, or `memErrNotEnoughSpace` if an error occurs.

## For System Use Only

### **FtrInit**

**Prototype**    `Err FtrInit (void)`

---

WARNING: This function for System use only

---

## Find Functions

### FindDrawHeader

**Purpose** Draw the header line that separates, by database, the list of found items.

**Prototype** `Boolean FindDrawHeader ( FindParamsPtr params,  
CharPtr title)`

**Parameters**

<code>params</code>	Handle of FindParamsPtr.
<code>title</code>	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 ( FindParamsPtr params,  
RectanglePtr r)`

**Parameters**

<code>params</code>	Handle of FindParamsPtr.
<code>r</code>	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** `void FindSaveMatch ( FindParamsPtr params,  
                          UInt recordNum,  
                          Word pos,  
                          UInt fieldNum,  
                          DWord appCustom,  
                          UInt dbCardNo,  
                          LocalID rdbID)`

**Parameters**

<code>params</code>	Handle of FindParamsPtr.
<code>recordNum</code>	Record index.
<code>pos</code>	Offset of the match string from start of record.
<code>appCustom</code>	Extra data the application can save with a match.
<code>dbCardNo</code>	Card number of the database that contains the match.
<code>rdbID</code>	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 is in lower-case characters.

**Prototype** `void FindStrInStr( CharPtr strToSearch,  
                    CharPtr strToFind,`



WordPtr posP)

<b>Parameters</b>	strToSearch	String to search.
	strToFind	Converted, caseless version of the ASCII text string to be found.
	posP	Pointer to offset in search string of the match.

**Result** Returns TRUE if the string was found.

**Comment** To convert a standard ASCII, null-terminated text string into the appropriate format for strToFind, use the conversion table returned by GetCharCaselessValue in code similar to the following:

```
CharPtr origStr;
    /* Standard null-terminated ascii string */
CharPtr strToFind;
    /* Converted string to be passed to */
    /* FindStrInStr */
BytePtr convTab;
    /* Conversion table returned from */
    /* GetCharCaselessValue*/
int i;
convTab = GetCharCaselessValue();
for (i=0; origStr[i] != 0; i++)
{
    strToFind[i] = convTab[origStr[i]];
}
strToFind[i] = 0;
    /* Now pass strToFind to FindStrInStr...*/
```

Note that the strToFind element of the parameter block passed by the system's Find utility is preconverted, so it can be passed straight through to FindStrInStr, just as in the example in the tutorial.

**See Also** [GetCharCaselessValue](#) (documented in "Developing Palm OS Applications, Part I)

## Float Manager Functions

Palm OS 2.0 implements floating point arithmetic differently than in Palm OS 1.0 did. The new floating-point library provides 32-bit and 64-bit floating point arithmetic.

### Using the New Floating Point Arithmetic

To take advantage of the new floating-point arithmetic, applications can now use the mathematical symbols  $+$   $-$   $*$   $/$  instead of using functions like FlpAdd, FlpSubtract, etc.

When compiling the application, you then have to link in the new library under certain circumstances. Choose from one of these options:

- **Simulator application or application for 1.0 device** — link in the new floating point library explicitly.

This library adds approximately 8KB to the size of your prc file. The new library provides 32-bit and 64-bit floating-point arithmetic. The original Palm OS Fpl functions only provided 16-bit floating-point arithmetic. Linking in the library explicitly won't cause problems when you compile for a 2.0 device.

- **2.0 Palm OS device**—It's not necessary to link in the library.

The compiler generates trap calls to equivalent floating-point functionality in the system ROM.

There are control panel settings in the IDE which let you select the appropriate floating-point model.

Floating-point functionality is identical in either method.

### Using 1.0 Floating-Point Functionality

The original Fpl calls (documented in this section) are still available. They may be useful for applications that don't need high precision, don't want to incur the size penalty of new float library, want to run on 1.0 device. To get 1.0 behavior, use the 1.0 calls (FplAdd, etc) and don't link in the library.

## **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, most applications will want to use the arithmetic symbols instead. See [Using the New Floating Point Arithmetic](#).

## **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.

## Palm OS System Functions

### *Float Manager Functions*

---

**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.

`FplBase10Info` 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, most applications will want to use the arithmetic symbols instead. See [Using the New Floating Point Arithmetic](#).

## **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.yyyyyyye[-]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)`

**Parameters** `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, most applications will want to use the arithmetic symbols instead. See [Using the New Floating Point Arithmetic](#).

#### **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, most applications will want to use the arithmetic symbols instead. See [Using the New Floating Point Arithmetic](#).



# Miscellaneous System Functions

## CmBroadcast

<b>Purpose</b>	Initiate connection establishment by broadcasting the “wakeup” packet.
<b>Prototype</b>	<code>Err CmBroadcast (CmParamPtr paramP)</code>
<b>Parameters</b>	<code>paramP</code> Pointer to Connection Manager parameters
<b>Result</b>	0 on success; otherwise: <code>cmErrParam</code> , <code>cmErrMemory</code> , <code>cmErrTimedOut</code> , <code>cmErrComm</code> , <code>cmErrCommBusy</code> , <code>cmErrUserCan</code> , <code>cmErrCommVersion</code>

## Crc16CalcBlock

<b>Purpose</b>	Calculate the 16-bit CRC of a data block using the table lookup method.
<b>Prototype</b>	<code>Word Crc16CalcBlock (VoidPtr bufP,                           UInt count,                           Word crc)</code>
<b>Parameters</b>	<code>bufP</code> Pointer to the data buffer. <code>count</code> Number of bytes in the buffer. <code>crc</code> Seed crc value.
<b>Result</b>	A 16-bit CRC for the data buffer.

## MdmDial

<b>Purpose</b>	Initialize the modem, dial the phone number and wait for result. When executing this function, the system goes through these steps: <ul style="list-style-type: none"><li>• Switch to the requested initial baud rate.</li></ul>
----------------	---

- 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.

**Prototype**    `Err MdmDial (     MdmInfoPtr modemP,  
                         CharPtr okDialP,  
                         CharPtr setupP,  
                         CharPtr phoneNumP)`

**Parameters**

<code>modemP</code>	Pointer to modem info structure (filled in by caller)
<code>okDialP</code>	(NOT IMPLEMENTED) Pointer to string of chars allowed in dial string
<code>setupP</code>	Pointer to modem setup string without the AT prefix.
<code>phoneNumP</code>	Pointer to phone number string

**Result**    0 if successful; otherwise `mdmErrNoTone`, `mdmErrNoDCD`, `mdmErrBusy`, `mdmErrUserCan`, `mdmErrCmdError`

## MdmHangUp

<b>Purpose</b>	Hang up the modem.
<b>Prototype</b>	<code>Err MdmHangUp (MdmInfoPtr modemP)</code>
<b>Parameters</b>	<code>modemP</code> Pointer to modem info structure (filled in by caller)
<b>Result</b>	0 if successful;

---

**Warning:** This function alters configuration of the serial port (without restoring it).

---

## PhoneNumberLookup

<b>Purpose</b>	This routine called the Address Book application to lookup a phone number. See the <code>phonelookup.c</code> example program for more information.
<b>Prototype</b>	<code>void PhoneNumberLookup (FieldPtr fld)</code>
<b>Parameters</b>	<code>fld</code> Field object in which the text to match is found.
<b>Comments</b>	<p>When trying to match a field, this function first tries to match selected text.</p> <ul style="list-style-type: none"><li>• If there is some selected text, the function replaces it with the phone number if there is a match.</li><li>• 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.</li><li>• If there is no match, the function displays the Address Book short list.</li></ul>
<b>Result</b>	Nothing returned; it's locked.

## 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 (Word 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` structure, this is also a `WindowHandle`.

## 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** `VoidPtr ResLoadMenu (Word 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` structure this is also a `WindowHandle`.

# System 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** `SWord PrefGetAppPreferences (DWord creator,  
Word id,  
VoidPtr prefs,  
Word *prefsSize,  
Boolean saved)`

**Parameters**

<code>creator</code>	Application creator.
<code>id</code>	ID number (lets an application have multiple preferences).
<code>prefs</code>	Pointer to a buffer to hold preferences.
<code>prefsSize</code>	Pointer to size the buffer passed.
<code>saved</code>	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.

**See Also** [PrefSetPreferences](#), [PrefGetAppPreferencesV10](#)

## **PrefGetAppPreferencesV10**

**Purpose** Return a copy of an application's preferences.

**Prototype** Boolean PrefGetAppPreferencesV10 (ULong type,  
Int version,  
VoidPtr prefs,  
Word 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.

**See Also** [PrefSetPreferences](#), [PrefGetAppPreferences](#)

## **PrefGetPreference**

**Purpose** Return a system preference. Use this instead of [PrefGetPreferences](#).

**Prototype** `DWord PrefGetPreference(  
SystemPreferencesChoice choice)`

**Parameters** System preference choice; see `Preferences.h` for available options.

**Comment** 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.

**Result** Returns the system preference.

**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.
- Note** This function is system use only in Palm OS 2.0.
- See Also** [PrefGetPreferences](#), [PrefSetPreferences](#)



## **PrefSetAppPreferences**

**Purpose** Set an application's preferences in the preferences database.

**Prototype** `void PrefSetAppPreferences ( DWord creator,  
Word id,  
SWord version,  
VoidPtr prefs,  
Word prefsSize,  
Boolean saved)`

**Parameters**

<code>creator</code>	Application creator type.
<code>id</code>	Resource ID (usually 0).
<code>version</code>	Version number of the application.
<code>prefs</code>	Pointer to a buffer that holds preferences.
<code>prefsSize</code>	Size of the buffer passed.
<code>saved</code>	If TRUE, set the saved preferences. If not, set the current preferences.

**Result** Nothing.

**Note** Unless you really want to set all preferences, use [PrefSetAppPreference](#) instead.

**See Also** [PrefSetAppPreferencesV10](#)

## **PrefSetAppPreferencesV10**

**Purpose** Save an application's preferences in the preferences database.

**Prototype** `void PrefSetAppPreferencesV10 ( ULong type,  
Int version,  
VoidPtr prefs,  
Word prefsSize)`

<b>Parameters</b>	<code>type</code>	Application creator type.
	<code>version</code>	Version number of the application.
	<code>prefs</code>	Pointer to a buffer holding preferences.
	<code>prefsSize</code>	Size of the buffer passed.
<b>Result</b>	Nothing.	
<b>Comments</b>	The content and format of an application preference is application-dependent.	
<b>See Also</b>	<a href="#">PrefSetAppPreferences</a> , <a href="#">PrefGetPreferences</a>	

## **PrefSetPreference**

<b>Purpose</b>	Set a system preference. Using this function instead of <code>PrefSetPreferences</code> allows you to set selected preferences without having to access the whole structure.	
<b>Prototype</b>	<pre>void PrefSetPreference(                         SystemPreferencesChoice choice,                         DWord value)</pre>	
<b>Parameters</b>	<code>choice</code>	A <code>SystemPreferencesChoice</code> (see <code>Preferences.h</code> )
	<code>value</code>	Value to assign to the item in <code>SystemPreferencesChoice</code> .
<b>Result</b>	Returns nothing. Changes the value of the system preference.	

## **PrefSetPreferences**

**Purpose** Set the system preferences.

**Prototype** `void PrefSetPreferences (SystemPreferencesPtr p)`

**Parameters** p      Pointer to system preferences.

**Result** Returns nothing.

**Comment** Unless there's a reason for you to access the whole preferences structure, call [PrefSetPreference](#) instead.

**See Also** [PrefGetPreferences](#)

## 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** `extern void PwdRemove()`

**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 (CharPtr oldPassword,  
CharPtr newPassword)`

**Parameters**

<code>oldPassword</code>	The old password must be successfully verified or the new password isn't accepted
<code>newPassword</code>	CharPtr 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 (CharPtr string)`

**Parameters**

<code>string</code>	String to compare to the system password. NULL means no current password.
---------------------	---

**Result** Returns TRUE if the string matches the system password.

## String Manager Functions

### StrATol

**Purpose** Convert a string to an integer.

**Prototype** `Int StrAToI (CharPtr str)`

**Parameters** `str` String to convert.

**Result** Returns the integer.

**Comments** Use this function instead of the standard `atoi` routine.

### StrCat

**Purpose** Concatenate one string to another.

**Prototype** `CharPtr StrCat (CharPtr dst, CharPtr 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.

## StrCaselessCompare

<b>Purpose</b>	Compare two strings with case and accent insensitivity.
<b>Prototype</b>	<code>Int StrCaselessCompare (CharPtr s1, CharPtr s2)</code>
<b>Parameters</b>	Two string pointers.
<b>Result</b>	Returns 0 if the two strings match, or non-zero if they don't.
<b>Comments</b>	Use this function instead of the standard <code>strcmp</code> routine. Use it to find strings but not sort them because it ignores case and accents.
<b>See Also</b>	<a href="#">StrCompare</a>

## StrChr

<b>Purpose</b>	Look for a character within a string.				
<b>Prototype</b>	<code>CharPtr StrChr (CharPtr str, Int chr)</code>				
<b>Parameters</b>	<table><tr><td><code>str</code></td><td>String to search.</td></tr><tr><td><code>chr</code></td><td>Character to search for.</td></tr></table>	<code>str</code>	String to search.	<code>chr</code>	Character to search for.
<code>str</code>	String to search.				
<code>chr</code>	Character to search for.				
<b>Result</b>	Returns a pointer to the first occurrence of character in <code>str</code> , or NULL if not found.				
<b>Comments</b>	Use this function instead of the standard <code>strchr</code> routine. This routine does not correctly find a <code>'\0'</code> character.				
<b>See Also</b>	<a href="#">StrStr</a>				

## StrCompare

**Purpose** Compare two strings.

**Prototype** `Int StrCompare (CharPtr s1, CharPtr 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** This function is case sensitive. Use it to sort strings but not to find them.  
Use this function instead of the standard `strcmp` routine.

**See Also** [StrCaselessCompare](#)

## StrCopy

**Purpose** Copy one string to another.

**Prototype** `CharPtr StrCopy (CharPtr dst, CharPtr src)`

**Parameters** `s1, s2` 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 return 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** `CharPtr StrDelocalizeNumber(  
CharPtr s,  
Char thousandSeparator,  
Char decimalSeparator)`

**Parameters**

<code>s</code>	Pointer to the number ASCII string.
<code>thousandSeparator</code>	Current thousand separator.
<code>decimalSeparator</code>	Current decimal separator.

**Result** Returns a pointer to the changed number and modifies the string in `s`.

**See Also** [StrLocalizeNumber](#), `LocGetNumberSeparators` (documented in *“Developing Palm OS Applications, Part I”*)

## **StrIToA**

**Purpose** Convert an integer to ASCII.

**Prototype** `CharPtr StrIToA (CharPtr s, Long i)`

**Parameters**

<code>s</code>	String pointer to store results.
<code>i</code>	Integer to convert.

**Result** Returns a pointer to the result string.

**See Also** [StrAToI](#), [StrIToH](#)

## StrToH

**Purpose** Convert an integer to hexadecimal ASCII.

**Prototype** `CharPtr StrToH (CharPtr s, ULong i)`

**Parameters** `s` String pointer to store results.  
`i` Integer to convert.

**Result** Returns the string pointer `s`.

**See Also** [StrToA](#)

## StrLen

**Purpose** Compute the length of a string.

**Prototype** `UInt StrLen (CharPtr src)`

**Parameters** `src` String pointer

**Result** Returns the length of the string.

**Comments** Use this function instead of the standard `strlen` routine.

## **StrLocalizeNumber**

**Purpose** Convert a number (passed in as a string) to localized format, using a specified thousandSeparator and decimalSeparator.

**Prototype** `void LocalizeNumber( CharPtr s,  
Char thousandSeparator,  
Char decimalSeparator)`

**Parameters**

<code>s</code>	Number ASCII string to localize
<code>thousandSeparator</code>	Localized thousand separator.
<code>decimalSeparator</code>	Localized decimal separator.

**Result** Returns nothing. Converts the number string in `s`.

**See Also** [StrDelocalizeNumber](#)

## **StrNCaselessCompare**

**Purpose** Compares two strings out to N characters with case and accent insensitivity.

**Prototype** `Int StrNCaselessCompare( const Char* s1,  
const Char* s2,  
DWord n)`

**Parameters**

<code>s1</code>	Pointer to first string.
<code>s2</code>	Pointer to second string.
<code>n</code>	Number of characters to compare.

**Result** 0 if they match, non-zero if not: positive if `s1 > s2`, negative if `s1 < s2`

**See Also** [StrNCompare](#)

## **StrNCat**

**Purpose** Concatenates 1 string to another clipping the destination string to a max of N characters (including null at end).

**Prototype** `CharPtr StrNCat( CharPtr dstP,  
                          const Char* srcP,  
                          Word n)`

**Parameters** `dstP` Pointer to destination string.  
`srcP` Pointer to source string.  
`n` Maximum number of characters for `dstP`.

**Result** Returns a pointer to the destination string.

## **StrNCompare**

**Purpose** Compare two strings out to N characters. This function is case and accent sensitive.

**Prototype** `Int StrNCompare(const Char* s1,  
                          const Char* s2,  
                          DWord n)`

**Parameters** `s1` Pointer to first string.  
`s2` Pointer to second string.  
`n` Number of characters to compare.

**Result** Returns 0 if the strings match, non-zero if they don't match. In that case:  
+ if `s1 > s2`  
- if `s1 < s2`

**See Also** [StrNCaselessCompare](#)

## **StrNCopy**

**Purpose** Copies up to N characters from `src` string to `dst` string. Terminates `dst` string at index N-1 if `src` string length was N-1 or less.

**Prototype** `CharPtr StrNCopy( CharPtr dstP,  
                          const Char* srcP,  
                          Word n)`

**Parameters**

<code>dstP</code>	Destination string.
<code>srcP</code>	Source string.
<code>n</code>	Maximum number of bytes to copy from <code>src</code> string.

**Result** Returns a pointer to destination string

## **StrPrintf**

**Purpose** Implements a subset of the ANSI C `sprintf()` call.  
Currently, only `%d`, `%i`, `%u`, `%x` and `%s` are implemented and don't accept field length or format specifications except for the `l` (long) modifier.

**Prototype** `SWord StrPrintf(CharPtr s,  
                          const Char* formatStr,  
                          ...)`

**Parameters**

<code>s</code>	Destination string
<code>formatStr</code>	Format string.
<code>* ...</code>	Arguments for format string.

**Result** Number of characters written to destination string.

**See Also** [StrVPrintf](#)

## StrStr

- Purpose** Look for a substring within a string.
- Prototype** `CharPtr StrStr (CharPtr str, CharPtr token)`
- Parameters**
- |                    |                       |
|--------------------|-----------------------|
| <code>str</code>   | String to search.     |
| <code>token</code> | 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.
- See Also** [StrChr](#)

## StrToLower

- Purpose** Convert all the characters in a string to lowercase.
- Prototype** `CharPtr StrToLower (CharPtr dst, CharPtr src)`
- Parameters** `dst, src` Two string pointers.
- Result** Returns a pointer to the destination string.
- Comments** This function **doesn't** convert accented characters.

## StrVPrintf

- Purpose** Implements a subset of the ANSI C `vsprintf()` call. Currently, only `%d`, `%i`, `%u`, `%x` and `%s` are implemented and don't accept field length or format specifications except for the `l` (long) modifier.

**Prototype**    `SWord StrVPrintf( CharPtr s,  
                              const Char* formatStr,  
                              VoidPtr argParam)`

**Parameters**    `s`                      Destination string.  
                  `formatStr`        Format string.  
                  `argParam`        Pointer to argument list.

**Result**        Returns the number of characters written to destination string.

**Example**       Here's an example of how to use this call:  
                  `#include <stdarg.h>`  
                  `void MyPrintf(CharPtr s, CharPtr formatStr, ...)`  
                  `{`  
                      `va_list args;`  
                      `Char text[0x100];`  
                      `va_start(args, formatStr);`  
                      `StrVPrintf(text, formatStr, args);`  
                      `va_end(args);`  
                      `MyPutS(text);`  
                  `}`

**See Also**      [StrPrintf](#)





# Sound Manager Functions

## SndDoCmd

**Purpose** Send a sound manager command to a specified sound channel.

**Prototype** `Err SndDoCmd ( VoidPtr chanP,  
SndCommandPtr cmdP,  
Boolean noWait)`

**Parameters**

- > `chanP` Pointer to sound channel. Present implementation doesn't support multiple channels. Must be zero.
- > `cmdP` Pointer to a `SndCommandType` structure which contains command parameters.
- > `noWait` 0 = await completion  
!0 = immediate return (asynchronous)  
asynchronous mode is not presently supported.

**Note** Passing `NIL` for the channel pointer causes the command to be sent to the shared sound channel. This is currently the only option.

<b>Result</b>	0	No error.
	<code>sndErrBadParam</code>	Invalid parameter.
	<code>sndErrBadChannel</code>	Invalid channel pointer.
	<code>sndErrQFull</code>	Sound queue is full.

## SndGetDefaultVolume

- Purpose** Return default sound volume levels.
- Prototype** `void SndGetDefaultVolume ( UIntPtr alarmAmpP,  
                                  UIntPtr sysAmpP,  
                                  UIntPtr defAmpP)`
- Parameters**
- |                                 |  |
|---------------------------------|--|
| <code>&lt;-&gt;alarmAmpP</code> | Pointer to storage for alarm amplitude.        |
| <code>&lt;-&gt; sysAmpP</code>  | Pointer to storage for system sound amplitude. |
| <code>&lt;-&gt; defAmpP</code>  | 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.

## SndPlaySystemSound

- Purpose** Play a standard system sound.
- Prototype** `void SndPlaySystemSound (SndSysBeepType beepID)`
- Parameters** `-> beepID` System sound to play.
- Comment** The `SndSysBeepType` enum is defined in `SoundManager.h` as follows:
- ```
typedef enum SndSysBeepType {  
    sndInfo = 1,  
    sndWarning,  
    sndError,  
    sndStartUp,  
    sndAlarm,  
    sndConfirmation,
```

```
    sndClick
} SndSysBeepType;
```

**Result** Returns nothing.

## SndSetDefaultVolume

**Purpose** Set the default sound volume levels.

**Prototype** `void SndSetDefaultVolume ( UIntPtr alarmAmpP,  
 UIntPtr sysAmpP,  
 UIntPtr defAmpP)`

**Parameters**

- > alarmAmpP      Pointer to alarm amplitude (0–sndMaxAmp).
- > sysAmpP        Pointer to system sound amplitude (0–sndMaxAmp).
- > defAmpP        Pointer to master amplitude (0–sndMaxAmp).

**Result** Returns nothing.

**Comments** Any pointer arguments may be passed as NULL. In that case, the corresponding setting is not affected.

All sound amplitudes greater than 0 are currently played as MaxVolume.

## Functions for System Use Only

### SndInit

**Prototype** `Err SndInit(void)`

---

WARNING: This function for use by system software only.

---

## System Functions

### SysAppLaunch

**Purpose** Launch the specified application with the given command line arguments, given a card number and database ID of an application resource database.

**Prototype** `Err SysAppLaunch( UInt cardNo, LocalID dbID,  
                          UInt launchFlags, Word cmd,  
                          Ptr cmdPBP, DWord* resultP)`

|                   |                           |                                                                                 |
|-------------------|---------------------------|---------------------------------------------------------------------------------|
| <b>Parameters</b> | <code>cardNo, dbID</code> | <code>cardNo</code> and <code>dbID</code> identify the application.             |
|                   | <code>launchFlags</code>  | Set to 0.                                                                       |
|                   | <code>cmd</code>          | Launch code.                                                                    |
|                   | <code>cmdPBP</code>       | Launch code parameter block.                                                    |
|                   | <code>resultP</code>      | Pointer to what's returned by the application's <code>PilotMain</code> 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.

**See Also** [SysBroadcastActionCode](#), [SysUIAppSwitch](#), [SysCurAppDatabase](#)

## SysAppLauncherDialog

**Purpose** Display the launcher, 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.

## SysBatteryInfo

**Purpose** Retrieve settings for the batteries. Set `set` to `FALSE` to retrieve battery settings. (Applications should *not* change any of the settings).

---

**Warning:** Use this function only to **retrieve** settings!

---

**Prototype** `UInt SysBatteryInfo( Boolean set,  
                          UIntPtr warnThresholdP,  
                          UIntPtr criticalThresholdP,  
                          UIntPtr maxTicksP,  
                          SysBatteryKind* kindP,  
                          Boolean* pluggedIn)`

**Parameters**

|                                 |                                                                                                                         |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>set</code>                | If <code>FALSE</code> , parameters with non-nil pointers are retrieved. Never set this parameter to <code>TRUE</code> . |
| <code>warnThresholdP</code>     | Pointer to battery voltage warning threshold in volts*100, or nil.                                                      |
| <code>criticalThresholdP</code> | Pointer to the battery voltage critical threshold in volts*100, or nil.                                                 |
| <code>maxTicksP</code>          | Pointer to the battery timeout, or nil.                                                                                 |

|                        |                                                         |
|------------------------|---------------------------------------------------------|
| <code>kindP</code>     | Pointer to the battery kind, or nil.                    |
| <code>pluggedIn</code> | Pointer to <code>pluggedIn</code> return value, or nil. |

**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 `SysBatteryWarningDialog` 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.

## **SysBinarySearch**

**Purpose** Search elements in an array according to the passed comparison function. Only elements which are out of order move. Moved elements are moved to the end of the range of equal elements. Use the quick sort if you need to sort many elements.

This function uses the following 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 insertion is performed. A moved element is inserted after any other equal elements.

**Prototype** `Boolean SysBinarySearch (`  
    `VoidPtr baseP, Int numOfElements,`  
    `Int width, SearchFuncPtr searchF,`  
    `const VoidPtr searchData, const Long other,`  
    `ULongPtr position, Boolean findFirst)`

**Parameters** `baseP` Base pointer to an array of elements,

|                            |                                                                                                                          |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <code>numOfElements</code> | Number of elements to sort (must be at least 2),                                                                         |
| <code>width</code>         | Width of an element comparison function.                                                                                 |
| <code>searchF</code>       | Search function.                                                                                                         |
| <code>searchData</code>    | Search data.                                                                                                             |
| <code>other</code>         | Other data passed to the comparison function.                                                                            |
| <code>position</code>      | Pointer to the position result.                                                                                          |
| <code>findFirst</code>     | If set to <code>TRUE</code> , the first matching element is returned (only needed if the array contains nonunique data). |

**Result** Returns `TRUE` if an exact match was found at the position in the database where the element should be located. `FALSE` otherwise.

## SysBroadcastActionCode

**Purpose** Send the specified action code (launch code) and parameter block to the latest version of every UI application.

**Prototype** `Err SysBroadcastActionCode (Word cmd, Ptr cmdPBP)`

**Parameters**

|                     |                                      |
|---------------------|--------------------------------------|
| <code>cmd</code>    | Action code to send.                 |
| <code>cmdPBP</code> | Action code parameter block to send. |

**Result** Returns 0 if no error, or one of the following errors:  
`sysErrParamErr`, `memErrNotEnoughSpace`,  
`sysErrOutOfOwnerIDs`.

**Comment** Launch codes are discussed in some detail in Chapter 2 of Developing Palm OS Applications, Part I.

**See Also** [SysAppLaunch](#)

## **SysCopyStringResource**

**Purpose** Copy a resource string to a passed string.

**Prototype** `void SysCopyStringResource ( CharPtr string,  
                                  UInt theID)`

**Parameters**

|                     |                                        |
|---------------------|----------------------------------------|
| <code>string</code> | String to copy the resource string to. |
| <code>theID</code>  | 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( ULong type,  
                                          ULong creator,  
                                          WordPtr dbCount,  
                                          Handle *dbIDs,  
                                          Boolean lookupName)`

**Parameters**

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <code>type</code>       | Type of database to find (0 for wildcard).                |
| <code>creator</code>    | Creator of database to find (0 for wildcard).             |
| <code>dbCount</code>    | Pointer to contain count of matching databases.           |
| <code>dbIDs</code>      | Pointer to handle allocated to contain the database list. |
| <code>lookupName</code> | Use tAIN names and sort the list.                         |



**Result** Returns FALSE if no panels were found, TRUE if panels were found. dbCount is updated to the number of databases found; dbIDs is updated to the list of matching databases found.

## **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(  
                                WordPtr panelCount,  
                                Handle \*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.

## **SysCurAppDatabase**

**Purpose** Return the card number and database ID of the current application's resource database.

**Prototype** Err SysCurAppDatabase (   UIntPtr cardNoP,  
                                                LocalID\* dbIDP)

**Parameters** cardNoP     Pointer to the card number; 0 or 1.  
dbIDB         Pointer to the database ID.

**Result** Returns 0 if no error, or SysErrParamErr if an error occurs.

**See Also** [SysAppLaunch](#), [SysUIAppSwitch](#)

## SysErrString

|                |                                                                                                                                                                                                                |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b> | 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.

**Note:** 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   CharPtr SysErrString(  Err err,
                                   CharPtr strP,
                                   Word maxLen)
```

|                   |                     |                                     |
|-------------------|---------------------|-------------------------------------|
| <b>Parameters</b> | <code>err</code>    | Error number                        |
|                   | <code>strP</code>   | Pointer to space to form the string |
|                   | <code>maxLen</code> | Size of <code>strP</code> buffer.   |

**Result** Stores the error number string.

## SysFatalAlert

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| <b>Purpose</b> | Display a fatal alert until the user taps a button in the alert. |
|----------------|------------------------------------------------------------------|

### Prototype

```
UInt SysFatalAlert (CharPtr msg)
```

|                   |                  |                                   |
|-------------------|------------------|-----------------------------------|
| <b>Parameters</b> | <code>msg</code> | Message to display in the dialog. |
|-------------------|------------------|-----------------------------------|

**Result** The button tapped; first button is zero.

## **SysFormPointerArrayToStrings**

**Purpose** Form an array of pointers to strings in a block. Useful for setting the items of a list.

**Prototype** `VoidHand SysFormPointerArrayToStrings  
(CharPtr c,  
Int 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.

## **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.

## 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 (documented in *Developing Palm OS Applications, Part I*)

## SysInsertionSort

**Purpose** Sort elements in an array according to the passed comparison function. 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.

```
Prototype void SysInsertionSort (Byte baseP,  
                                Int numOfElements,  
                                Int width,  
                                CmpFuncPtr comparF,  
                                Long other)
```

|                   |               |                                                  |
|-------------------|---------------|--------------------------------------------------|
| <b>Parameters</b> | 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.

|                 |                                                                                                                                                                                                                                                                                                                    |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Comments</b> | <p>In Palm OS 2.0, DmComparF has 6 parameters.</p> <p>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.</p> |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

The revised callback is used by new sorting routines (and can be used the same way by your application):

```
typedef Int DmComparF (    void *,
                           void *,
                           Int other,
                           SortRecordInfoPtr,
                           SortRecordInfoPtr,
                           VoidHand 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 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 Pilot may have problems as a result of the change in arguments when running on a 2.0 device.

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](#)

## SysInstall

|                   |                                                                               |
|-------------------|-------------------------------------------------------------------------------|
| <b>Purpose</b>    | Entry point for System code resource, 'CODE' #0, in the System resource file. |
| <b>Prototype</b>  | <code>void SysInstall (Ptr tableP[])</code>                                   |
| <b>Parameters</b> | <code>tableP</code> Pointer to trap table.                                    |
| <b>Result</b>     | Returns nothing                                                               |
| <b>Comments</b>   | Called by <code>Init()</code> in the ROMMain module.                          |

## SysKeyboardDialog

|                   |                                                                                                                         |
|-------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>    | Pop up the system keyboard if there is a field object with the focus. The field object's text chunk is edited directly. |
| <b>Prototype</b>  | <code>void SysKeyboardDialog (KeyboardType kbdType)</code>                                                              |
| <b>Parameters</b> | <code>kbdType</code> The keyboard type. See <code>keyboard.h</code> .                                                   |
| <b>Result</b>     | Returns nothing. Changes the field's text chunk.                                                                        |
| <b>See Also</b>   | <a href="#">SysKeyboardDialogV10</a> <code>FrmSetFocus</code> (documented in "Developing Palm OS Applications, Part I") |

## **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.

**See Also** [SysKeyboardDialog.FrmSetFocus](#) (documented in "Developing Palm OS Applications, Part I")

## **SysLibLoad**

**Purpose** A utility routine to load a library given its database creator and type.  
Presently, the "load" functionality is NOT supported when you use the Palm OS Simulator.

**Prototype** `Err SysLibLoad( DWord libType,  
DWord libCreator,  
UIntPtr refNumP)`

**Parameters**

|                         |                                                                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <code>libType</code>    | Type of library database.                                                                                                              |
| <code>libCreator</code> | Creator of library database.                                                                                                           |
| <code>refNumP</code>    | Pointer to variable for returning the library reference number(on failure, <code>sysInvalidRefNum</code> 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** 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 should soon become available on the developer support web site.

## 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 (   Byte baseP,  
                  Int numOfElements,  
                  Int width,  
                  CmpFuncPtr comparF,  
                  Long 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 <a href="#">SysInsertionSort</a> . |
| 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** Int SysRandom (ULong newSeed)

**Parameters** newSeed New seed value, or 0 to use existing seed.

**Result** Returns a random number.

## SysReset

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>    | Perform a soft reset and reinitialize the globals and the dynamic memory heap.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Prototype</b>  | <code>void SysReset (void)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Parameters</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Result</b>     | No return value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Comments</b>   | <p>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.</p> <p>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.</p> <p>When running an application on the device, this routine simply looks for the memory cards at fixed locations.</p> |

## SysSetAutoOffTime

|                   |                                                                                   |
|-------------------|-----------------------------------------------------------------------------------|
| <b>Purpose</b>    | Set the time out value in seconds for auto-power-off. Zero means never power off. |
| <b>Prototype</b>  | <code>UInt SysSetAutoOffTime (UInt seconds)</code>                                |
| <b>Parameters</b> | <code>seconds</code> Time out in seconds, or 0 for no time out.                   |
| <b>Result</b>     | Returns previous value of time out in seconds.                                    |

## SysStringByIndex

|                |                                                                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b> | 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. |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------|

**Warning:** ResEdit always displays the items in the list as starting at 1, not 0. Consider this when creating your string list.

```
Prototype   CharPtr SysStringByIndex(  Word resID,
   Word index,
   CharPtr strP,
   Word maxLen)
```

|                   |               |                                      |
|-------------------|---------------|--------------------------------------|
| <b>Parameters</b> | <b>resID</b>  | Resource ID of the string list.      |
|                   | <b>index</b>  | String to get out of the list.       |
|                   | <b>strP</b>   | Pointer to space to form the string. |
|                   | <b>maxLen</b> | Size of <b>strP</b> 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.

## SysTaskDelay

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <b>Purpose</b> | Put the processor into doze mode for the specified number of ticks. |
|----------------|---------------------------------------------------------------------|

**Prototype** Err SysTaskDelay (Long delay)

|                   |       |                                                 |
|-------------------|-------|-------------------------------------------------|
| <b>Parameters</b> | 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** `Word SysTicksPerSecond(void)`

**Parameters** None

**Result** Returns the number of ticks per second.

## **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(    UInt cardNo,  
                          LocalID dbID,  
                          Word cmd,  
                          Ptr cmdPBP)`

**Parameters**

|                     |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <code>cardNo</code> | Card number for the new application; currently only card 0 is valid.            |
| <code>dbID</code>   | ID of the new application.                                                      |
| <code>cmd</code>    | Action code (launch code). See <i>Developing Palm OS Applications, Part I</i> . |
| <code>cmdPBP</code> | Action code (launch code) parameter block.                                      |

**Result** Returns 0 if no error.

**See Also** [SysAppLaunch](#)

## Functions for System Use Only

### **SysAppExit**

**Prototype**    `Err SysAppExit (SysAppInfoPtr appInfoP,  
                  Ptr prevGlobalsP, Ptr globalsP)`

---

WARNING: System Use Only!

---

### **SysAppInfoPtr**

**Prototype**    `SysAppInfoPtr SysCurAppInfoP (void)`

---

WARNING: System Use Only!

---

### **SysAppStartup**

**Prototype**    `Err       SysAppStartup ( SysAppInfoPtr appInfoPP,  
                          Ptr prevGlobalsP, Ptr globalsP)`

---

WARNING: System Use Only!

---

### **SysBatteryDialog**

**Prototype**    `void SysBatteryDialog (void)`

---

WARNING: System Use Only!

---

### **SysCardImageDeleted**

**Prototype**    `void SysCardImageDeleted (UInt cardNo)`

---

WARNING: System Use Only!

---

**SysCardImageInfo**

**Prototype**    `Ptr SysCardImageInfo (UInt cardNo, ULongPtr sizeP)`

---

WARNING: System Use Only!

---

**SysColdBoot**

**Purpose**        Perform a cold boot and reformat all RAM areas of both memory cards.

---

WARNING: System Use Only!

---

**SysCurAppInfoP**

**Prototype**    `SysCurAppInfoPtr SysCurrAppInfoP (void)`

---

WARNING: System Use Only!

---

**SysDisableInts**

**Prototype**    `Word SysDisableInts (void)`

---

WARNING: System Use Only!

---

**SysDoze**

**Prototype**    `void SysDoze (Boolean onlyNMI)`

---

WARNING: System Use Only!

---

**SysEvGroupCreate**

**Prototype**    `Err SysEvGroupCreate(DWordPtr evIDP, DWordPtr tagP, DWord init)`

---

WARNING: System Use Only!

---

### **SysEvGroupRead**

**Prototype**    `Err SysEvGroupRead(DWord evID, DWordPtr valueP)`

---

WARNING: System Use Only!

---

### **SysEvGroupSignal**

**Prototype**    `Err SysEvGroupSignal(DWord evID, DWord mask, DWord value, SDWord type)`

---

WARNING: System Use Only!

---

### **SysEvGroupWait**

**Prototype**    `Err SysEvGroupWait(DWord evID, DWord mask, DWord value, SDWord matchType, SDWord timeout)`

---

WARNING: System Use Only!

---

### **SysGetTrapAddress**

**Prototype**    `VoidPtr SysGetTrapAddress (UInt trapNum)`

---

WARNING: System Use Only!

---

### **SysInit**

**Prototype**    `void SysInit (void)`

---

WARNING: System Use Only!

---



## SysKernelInfo

```
Err SysKernelInfo (VoidPtr paramP)
```

**WARNING: System Use Only!**

## SysLaunchConsole

```
Err SysLaunchConsole (void)
```

**WARNING: System Use Only!**

## SysLibFind

```
Err SysLibFind (CharPtr nameP, UIntPtr refNumP)
```

**WARNING: System Use Only!**

## SysLibInstall

```
Err    SysLibInstall ( SysLibEntryProcPtr libraryP,  
                        UIntPtr refNumP)
```

**WARNING: System Use Only!**

## SysLibRemove

Err SysLibRemove (UInt refNum)

**WARNING: System Use Only!**

## SysLibTblEntry

```
SysLibTblEntryPtr SysLibTblEntry (UInt refNum)
```

**WARNING: System Use Only!**

### **SysMailboxCreate**

**Prototype**    `Err SysMailboxCreate(DWordPtr mbIDP, DWordPtr tagP, DWord depth)`

---

WARNING: System Use Only!

---

### **SysMailboxDelete**

**Prototype**    `Err SysMailboxDelete(DWord mbID)`

---

WARNING: System Use Only!

---

### **SysMailboxFlush**

**Prototype**    `Err SysMailboxFlush(DWord mbID)`

---

WARNING: System Use Only!

---

### **SysMailboxSend**

**Prototype**    `Err SysMailboxSend(DWord mbID, VoidPtr msgP, DWord wAck)`

---

WARNING: System Use Only!

---

### **SysMailboxWait**

**Prototype**    `Err SysMailboxWait(DWord mbID, VoidPtr msgP, DWord priority, SDWord timeout)`

---

WARNING: System Use Only!

---

**SysNewOwnerID**

**Prototype**    `UInt SysNewOwnerID (void)`

---

WARNING: System Use Only!

---

**SysPowerOn**

**Prototype**    `void SysPowerOn (   Ptr card0P, UInt card0Size,  
                          Ptr card1P, UInt card1Size,  
                          DWord sysCardHeaderOffset,  
                          Boolean reFormat)`

---

WARNING: System Use Only!

---

**SysRestoreStatus**

**Prototype**    `void SysRestoreStatus (Word status)`

---

WARNING: System Use Only!

---

**SysSetA5**

**Prototype**    `DWord SysSetA5 (DWord newValue)`

---

WARNING: System Use Only!

---

**SysSetTrapAddress**

**Prototype**    `Err SysSetTrapAddress (   UInt trapNum,  
                                          VoidPtr procP)`

---

WARNING: System Use Only!

---

### **SysSleep**

**Prototype**    `void SysSleep (   Boolean untilReset,  
                         Boolean emergency)`

---

WARNING: System Use Only!

---

### **SysTaskResume**

**Prototype**    `Err SysTaskResume(DWord taskID)`

---

WARNING: System Use Only!

---

### **SysTaskSuspend**

**Prototype**    `Err SysTaskSuspend(DWord taskID)`

---

WARNING: System Use Only!

---

### **SysUILaunch**

**Prototype**    `void SysUILaunch (void)`

---

WARNING: System Use Only!

---

### **SysTaskWait**

**Prototype**    `Err SysTaskWait(SDWord timeout)`

---

WARNING: System Use Only!

---

### **SysTaskWaitClr**

**Prototype**    `Err SysTaskWaitClr(void)`

---

WARNING: System Use Only!

---

### **SysTaskWake**

**Prototype**    `Err SysTaskWake(DWord taskID)`

---

WARNING: System Use Only!

---



# Time Manager Functions

## DateAdjust

|                   |                                                                                                     |                                                                                                |
|-------------------|-----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| <b>Purpose</b>    | Return a new date +/- the days adjustment.                                                          |                                                                                                |
| <b>Prototype</b>  | <code>void DateAdjust (DatePtr dateP, Long adjustment)</code>                                       |                                                                                                |
| <b>Parameters</b> | <code>dateP</code>                                                                                  | A <code>DateType</code> structure with the date to be adjusted (see <code>DateTime.h</code> ). |
|                   | <code>adjustment</code>                                                                             | The adjustment in seconds.                                                                     |
| <b>Result</b>     | Changes <code>dateP</code> to contain the new date.                                                 |                                                                                                |
| <b>Comments</b>   | 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

|                   |                                                              |                                                        |
|-------------------|--------------------------------------------------------------|--------------------------------------------------------|
| <b>Purpose</b>    | Return the date, given days.                                 |                                                        |
| <b>Prototype</b>  | <code>void DateDaysToDate (ULong days, DatePtr dateP)</code> |                                                        |
| <b>Parameters</b> | <code>days</code>                                            | Days since 1/1/1904.                                   |
|                   | <code>dateP</code>                                           | Pointer to <code>DateType</code> structure (returned). |
| <b>Result</b>     | Returns nothing, stores the date in <code>dateP</code> .     |                                                        |
| <b>See Also</b>   | <a href="#">TimAdjust</a> , <a href="#">DateToDays</a>       |                                                        |

## DateSecondsToDate

**Purpose** Return the date given seconds.

**Prototype** `void DateSecondsToDate ( ULong seconds,  
DatePtr dateP)`

**Parameters**    `seconds`                Seconds since 1/1/1904.  
                  `dateP`                Pointer to `DateType` structure (returned).

**Result** Returns nothing; stores the date in `dateP`.

## DateToAscii

**Purpose** Convert the time passed to an ASCII string in the passed `DateFormatType`. Handles long and short formats.

**Prototype** `void DateToAscii( Byte months,  
Byte days,  
Word years,  
DateFormatType dateFormat,  
CharPtr pString)`

**Parameters**    `months`        Months (1-12).  
                  `days`        Days (1-31).  
                  `years`        Years (for example 1995).  
                  `dateFormat` Long or short `DateFormatType`.  
                  `pString`    Pointer to string which gets the result. Must be of  
                                 length `dateStringLength` for standard formats or  
                                 `longDateStrLength` for long date formats.

**Result** Returns nothing. Stores the result in `pString`.

**See Also**    [TimeToAscii](#), [DateToDOWDMFormat](#)



## DateToDays

**Purpose** Return the date in days since 1/1/1904.

**Prototype** `ULong DateToDays (DateType date)`

**Parameters** `date`            `DateType` structure.

**Result** Returns the days since 1/1/1904.

**See Also** [TimAdjust](#), [DateDaysToDate](#)

## DateToDOWDMFormat

**Purpose** Convert the date passed to an ASCII string.

**Prototype** `void DateToDOWDMFormat(   Byte months,  
                                  Byte days,  
                                  Word years,  
                                  DateFormatType dateFormat,  
                                  CharPtr pString)`

**Parameters** `months`            Month (1-12).  
              `days`            Day (1-31).  
              `years`            Years (for example 1995).  
              `dateFormat`        FALSE to use AM and PM.  
              `pString`          Pointer to string which gets the result. The  
                                  string must be of length `timeStringLength`.

**Result** Returns nothing; stores ASCII string in `pString`.

**See Also** [DateToAscii](#)

## DayOfMonth

**Purpose** Return the day of a month on which the specified date occurs (for example, dom2ndTue).

**Prototype** `UInt DayOfMonth (UInt month, UInt day, UInt year)`

**Parameters**

|                    |                          |
|--------------------|--------------------------|
| <code>month</code> | Month (1-12).            |
| <code>day</code>   | Day (1-31).              |
| <code>year</code>  | Year (for example 1995). |

**Result** Returns the day of the month as a `DayOfWeekType`, see `DateTime.h`.

## DayOfWeek

**Purpose** Return the day of the week.

**Prototype** `UInt DayOfWeek (UInt month, UInt day, UInt year)`

**Parameters**

|                    |                          |
|--------------------|--------------------------|
| <code>month</code> | Month (1-12).            |
| <code>day</code>   | Day (1-31).              |
| <code>year</code>  | 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** `UInt DaysInMonth (UInt month, UInt year)`
- Parameters**
- |                    |                           |
|--------------------|---------------------------|
| <code>month</code> | Month (1-12).             |
| <code>year</code>  | 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,  
Long adjustment)`
- Parameters**
- |                         |                                                                   |
|-------------------------|-------------------------------------------------------------------|
| <code>dateTimeP</code>  | A <code>DateType</code> structure (see <code>DateTime.h</code> ). |
| <code>adjustment</code> | 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 date and time in seconds since 1/1/1904.

**Prototype** ULong TimDateTimeToSeconds (DateTimePtr dateTimeP)

**Parameters** dateTimeP            A DateType structure (see DateTime.h).

**Result** The time in seconds since 1/1/1904.

**See Also** [TimSecondsToDateTime](#)

## TimGetSeconds

**Purpose** Return seconds since 1/1/1904.

**Prototype** ULong 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** ULong TimGetTicks (void)

**Parameters** None.

**Result** Returns the tick count.

## TimSecondsToDateTime

- Purpose** Return the date and time, given seconds.
- Prototype** `void TimSecondsToDateTime( ULong 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** Return seconds since 1/1/1904.
- Prototype** `void TimSetSeconds (ULong seconds)`
- Parameters** `seconds` Place to return the seconds since 1/1/1904.
- Result** Returns nothing; modifies `seconds`.
- See Also** [TimGetSeconds](#)

## TimeToAscii

**Purpose** Convert the time passed to an ASCII string.

**Prototype**    `void TimeToAscii(   Byte hours,  
                          Byte minutes,  
                          TimeFormatType timeFormat,  
                          CharPtr pString)`

**Parameters**

|                         |                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------|
| <code>hours</code>      | Hours (0-23).                                                                              |
| <code>minutes</code>    | Minutes (0-59).                                                                            |
| <code>timeFormat</code> | FALSE to use AM and PM.                                                                    |
| <code>pString</code>    | Pointer to string which gets the result. Must be of length <code>timeStringLength</code> . |

**Result** Returns nothing. Stores pointer to the text of the current selection in `pString`.

**See Also**    [DateToAscii](#)

## Functions for System Use Only

### TimGetAlarm

**Prototype**    `ULong TimGetAlarm (void)`

---

WARNING: System use only!

---

### TimHandleInterrupt

**Prototype**    `void TimHandleInterrupt (Boolean periodicUpdate)`

---

Warning: System use only!

---

### TimInit

**Prototype**    `Err TimInit (void)`

---

Warning: System use only!

---

### TimSetAlarm

**Prototype**    `ULong TimSetAlarm (ULong alarmSeconds)`

---

Warning: System use only!

---





# Index

## Numerics

- 0.01-second timer 41
- 1-second timer 41
- 2.0 functionality
  - float manager 74

## A

- accented characters and StrToLower 102
- adding event to event queue 53
- alarm manager 14–17
  - and alarm sound 15
  - reminder dialog boxes 15
- alarm sound 15, 25
- alarms
  - canceling 46
  - setting 46
- alerts
  - SysFatalAlert 114
- AlmCancelAll 47
- AlmDisplayAlarm 47
- AlmEnableNotification 47
- almErrFull 46
- almErrMemory 46
- AlmGetAlarm 45
- AlmInit 47
- AlmSetAlarm 17, 46
- amplitude 25
- application preferences 85
- application-defined features 23
- auto-off 30
  - setting 123
  - timer 39, 62
- auto-repeat 33, 39

## B

- base 10 form of floating-point number 75
- battery 30
- battery conservation using modes 29
- battery timeout 109
- battery voltage warning threshold 109
- booting 27
- bound of next line for global find 71

- buttons
  - silk-screened icons 33

## C

- C library
  - and float manager 74
  - and string manager 26
- canceling alarms 46
- cleanup of dynamic heap 31
- Click 25
- CmBroadcast 81
- cmErrComm 81
- cmErrCommBusy 81
- cmErrCommVersion 81
- cmErrMemory 81
- cmErrParam 81
- cmErrTimedOut 81
- cmErrUserCan 81
- code #0 resource 119
- Confirmation sound 25
- connection, initiating 81
- conserving battery using modes 29
- Crc16CalcBlock 81

## D

- database ID
  - and launch codes 35
- databases
  - SysCreateDataBaseList 112
- date and time manager 41
- DateAdjust 135
- DateDaysToDate 135
- DateSecondsToDate 136
- dateStringLength 136
- DateToAscii 136
- DateToDays 137
- DayOfMonth 138
- DayOfWeek 138
- DaysInMonth 139
- default sound volume 107
- dialog boxes (reminder) 15
- digitizer
  - and pen queue 37

## Index

- EvtProcessSoftKeyStroke 62
  - pen stroke to key event 37
- DmComparF 117
- doze mode 29
  - SysTaskDelay 124
- dynamic heap
  - cleanup 31
  - reinitializing 123

## E

- ErrCatch 52
- ErrDisplay 18, 19, 48
- ErrEndCatch 52
- ErrFatalDisplayIf 18, 19, 50
- ErrNonFatalDisplayIf 51
- error manager 17–22
  - try-and-catch mechanism 20
- Error sound 25
- ERROR\_CHECK\_FUL 48
- ERROR\_CHECK\_FULL 51
- ERROR\_CHECK\_LEVEL 18, 19, 48, 50, 51
- ERROR\_CHECK\_PARTIAL 48
- ErrThrow 20, 52
- ErrTry 52
- event processing 32
- event queue
  - adding event 53
- events
  - hard button presses 32
  - hardware generated 32, 33
  - software generated 32, 34
- EvtAddEventToQueue 53
- EvtAddUniqueEventToQueue 53
- EvtCopyEvent 54
- EvtDequePenStrokeInfo 38
- EvtDequeuePenPoint 54
- EvtDequeuePenStrokeInfo 55
- EvtEnableGraffiti 55
- EvtEnqueueKey 56
- EvtEventAvail 57
- EvtFlushKeyQueue 57
- EvtFlushNextPenStroke 58
- EvtFlushPenQueue 58
- EvtGetEvent 32

- EvtGetPen 59
- EvtGetPenBtnList 60
- EvtKeyQueueEmpty 60
- EvtKeyQueueSize 61
- EvtPenQueueSize 61
- EvtProcessSoftKeyStroke 62
- EvtResetAutoOffTimer 39, 62
- EvtSysEventAvail 63
- EvtWakeup 63

## F

- fatal alert 114
- feature manager 22–24
- features
  - See functions starting with Ftr
  - application-defined 23
  - system version 23
- FIFO queue 33
- FindDrawHeader 71
- FindGetLineBounds 71
- FindSaveMatch 72
- FindStrInStr 72
- float manager overview 74
- flushing pen queue 58
- FplAdd 75
- FplAToF 75
- FplBase10Info 76
- FplDiv 76
- FplFloatToLong 77
- FplFloatToULong 77
- FplFree 78
- FplFToA 78
- FplInit 79
- FplLongToFloat 79
- FplMul 80
- FplSub 80
- ftrErrInternalError 66, 67
- ftrErrNoSuchFeature 67, 68, 69
- ftrErrNoSuchFtr 66
- FtrGet 24, 66
- FtrGetByIndex 24, 67
- ftrInternalError 69
- FtrSet 24, 68
- FtrUnregister 24, 69

**G**

- GetCharCaselessValue  
and FindStrInStr 73
- global find
  - FindDrawHeader 71
  - FindGetLineBounds 71
- Graffiti
  - enabling and disabling 55
  - events 32
- Graffiti recognizer 37
  - EvtDequeuePenPoint 54
- Graffiti Reference Dialog 115

**H**

- hard button press events 32
- hardware-generated events 32, 33
- header line for global find 71

**I**

- Information sound 25
- insertion sort 117
- interrupting Sync application 31

**K**

- kernel 30
- key debouncing 33
- key events
  - format 56
  - from pen strokes 37
- key presses 32
- key queue 38
  - size 61
- keyboard display 119

**L**

- launch codes 34
  - and returned database ID 35
  - SysBroadcastActionCode 35, 111
- launcher screen 31
- launching applications 31
- library vs. managers 13
- lists
  - setting items 115
- longDateStrLength 136

- low-battery warning 34

**M**

- managers
  - naming convention 13
  - vs. libraries 13
- MdmDial 82
- mdmErrBusy 82
- mdmErrCmdError 82
- mdmErrNoDCD 82
- mdmErrNoTone 82
- mdmErrUserCan 82
- MdmHangUp 83
- memErrChunkLocked 68, 69
- memErrInvalidParam 68, 69
- memErrNotEnoughSpace 68, 69, 108, 111
- modem 81
- modes 28
  - efficient use 29
- multiple preferences 85
- multitasking kernel 30

**N**

- nilEvent 63
- noPreferenceFound 85

**P**

- panel list (SysCreatePanelList) 113
- password functions 92
- pen
  - current status 59
  - strokes and key events 37
- pen events 32
- pen queue 37
  - flushing 58
  - size 61
- PhoneNumberLookup 83
- power modes 28
- preferences
  - auto-off 30
  - multiple application preferences 85
- PrefGetAppPreferences 85
- PrefGetAppPreferencesV10 86
- PrefGetPreference 87

## Index

PrefGetPreferences 88  
PrefOpenPreferenceDBV10 88  
PrefSetAppPreferences 89  
PrefSetAppPreferencesV10 89  
PrefSetPreference 90  
PrefSetPreferences 91  
PwdExists 92  
PwdRemove 92  
PwdSet 93  
PwdVerify 93

## Q

quitting application 32

## R

real-time clock 41  
reinitializing dynamic memory heap 123  
reminder dialog boxes 15  
reset 123  
ResLoadForm 84  
ResLoadMenu 84  
resource database, SysCurAppDatabase 113  
response time 31  
running mode 29

## S

searching for string 72  
searching for substring 102  
silk-screen buttons  
    EvtGetPenBtnList 60  
silk-screened icons 33  
sleep mode 28  
    and real-time clock 41  
SndDoCmd 105  
sndErrBadChannel 105  
sndErrBadParam 105  
sndErrQFull 105  
SndGetDefaultVolume 106  
SndInit 107  
sndMaxAmp 107  
SndPlaySystemSound 106  
SndSetDefaultVolume 107  
SndSysBeepType 106  
soft reset 123

software-generated events 32, 34  
sorting array elements 117  
sound manager 25  
    amplitude 25  
    volume 25  
sound manager functions 105–107  
sprintf (StrPrintf) 101  
StartApplication  
    and PrefGetPreferences 88  
Startup sound 25  
string  
    searching 72  
string manager 26  
string manager functions 94–103  
string resource  
    copying 112  
strokes  
    capturing 38  
    translating 62  
StrPrintf 101  
StrStr 102  
StrToLower 102  
StrVPrintf 103  
substring, searching for 102  
Sync application 31  
SysAppLaunch 31, 35, 108  
sysAppLaunchCmdAlarmTriggered 15  
sysAppLaunchCmdDisplayAlarm 15  
SysAppLauncherDialog 109  
SysBinarySearch 110  
SysBroadcastActionCode 35, 111  
SysCopyStringResource 112  
SysCreateDataBaseList 112  
SysCreatePanelList 113  
SysCurAppDatabase 35, 113  
sysErrLibNotFound 120  
sysErrNoFreeLibSlots 120  
sysErrNoFreeRAM 120  
sysErrOutOfOwnerID 108  
sysErrOutOfOwnerIDs 111  
sysErrParamErr 108, 111  
SysErrString 114  
SysFatalAlert 114  
SysFormPointerArrayToStrings 115  
SysGraffitiReferenceDialog 115

- SysHandleEvent 32, 33, 116
- SysInsertionSort 117
- SysInstall 119
- SysKeyboardDialog 119
- SysKeyboardDialogV10 120
- SysLibLoad 120
- SysQSort 121
- sysRandomMax 122
- SysReset 123
- SysSetAutoOffTime 123
- SysStringByIndex 124
- SysTaskDelay 124
- system event manager 36–40
- system events
  - checking availability 63
- system keyboard display 119
- system ticks 41
  - and Simulator 41
  - on Palm OS device 41
- system version feature 23
- sysTicksPerSecond 42
- SysUIAppSwitch 35, 125

## T

- TimAdjust 139

- TimDateTimeToSeconds 41, 140
- time manager 41
  - structures 42
- TimeToAscii 142
- TimGetSeconds 41, 140
- TimGetTicks 42, 140
- timing 42
- TimSecondsToDateTime 41, 141
- TimSetSeconds 41, 141
- try-and-catch mechanism 20
  - example 21

## U

- UIAS 28, 30
- User Interface Application Shell 28, 30
- using modes efficiently 29

## V

- voltage warning threshold 109
- volume 25
- volume default 107
- vsprintf (StVPrintf) 102

## W

- Warning sound 25