palmOne™

palmOne™ Fall '04 Developer Guide

Copyright

palmOne, Zire, Tungsten, Treo, Blazer, Graffiti, HotSync, VersaMail, Palm Powered, and Palm OS are among the trademarks or registered trademarks owned by or licensed to palmOne, Inc. All other brand and product names are or may be trademarks of, and are used to identify products or services of, their respective owners.

Disclaimer and Limitation of Liability

palmOne, Inc. assumes no responsibility for any damage or loss resulting from the use of this guide.

palmOne, Inc. assumes no responsibility for any loss or claims by third parties which may arise through the use of this software. palmOne, Inc. assumes no responsibility for any damage or loss caused by deletion of data as a result of malfunction, dead battery, or repairs. Be sure to make backup copies of all important data on other media to protect against data loss.

IMPORTANT Please read the End User Software License Agreement with this product before using the accompanying software program(s). Using any part of the software indicates that you accept the terms of the End User Software License Agreement.

POWERED

Contents

PART II: Hardware Developers Kit

PART III: Debugging

PART IV: Style Guide

CHAPTER 1

Overview

This chapter introduces you to the palmOne™ Product Line Software Development Kit (SDK). This chapter also discusses how this guide is organized and how to use the SDK.

How this guide is organized

This guide contains an overview of the Treo™ smartphones, the Tungsten™ handhelds, and the Zire™ handhelds as well as hardware specifications for each product and a feature matrix that shows which features are available on each product.

The libraries discussed in this guide are organized by general category, such as Multimedia Libraries and Application Libraries. Some of these libraries are completely new, and others are improvements over the previous generation of libraries.

In addition to the discussion of the libraries, this guide contains some debugging information for troubleshooting problems on the various products, as well as style conventions for how certain features should be used.

Where the APIs are described

In the past, documentation on API functions and structures has been included in the SDK guides. The complete API documentation is now generated directly from palmOne source code. Refer to the separate palmOne API guide either in compressed HTML format (.chm) or HTML format available directly from http://pluggedin.palmone.com.

Installing the SDK

The latest SDK is available for download from http://pluggedin.palmone.com. It contains all of the released libraries described in this guide, as well as sample code and examples.

CHAPTER 2

Product Line Overview

This chapter provides an overview of the Treo™ smartphones, the Tungsten™ handhelds, and the Zire™ handhelds as well as a high-level description of the features available in each product line. Also included are a hardware features table that shows the hardware and electrical specifications of each model, and a software specifications table that shows which specific features and APIs apply to each model.

Treo™ smartphone product line

The Treo smartphone by palmOne™ is the fifth generation of truly compact smartphone to integrate in one product a mobile phone, wireless data applications such as messaging and web browsing, and a Palm OS® organizer. Treo smartphones are available in two radio versions: one with quad-band GSM/GPRS/EDGE radio (with operating frequencies of 850, 900, 1800, and 1900MHz) and one with dual-band CDMA/1xRTT radio (with operating frequencies of 800MHz and 1900MHz).

One of the key differentiators of the Treo smartphone is the tight integration of the main applications and the user interface, which makes all Treo smartphone applications easy to use. Most applications are common to both versions of the Treo (CDMA & GPRS) smartphone and they include:

- Phone application.
- Short Messaging Service (SMS) messaging.

  On a GSM network, SMS messages can be received and sent. On a CDMA network, SMS services may or may not be offered and supported by the operator.
- Proxyless Blazer® web browser supporting direct download of ring tones, applications, and documents.
- Photo capture application.
- E-mail.

  E-mail applications are available. The type of application can vary by operator.
- Standard Palm OS organizer applications.
- On the Treo 650 smartphone, music playback capability and a high-resolution screen.

NOTE Some applications are applicable only to the GSM or CDMA version. Treo smartphones might also be configured differently depending on the Operator.

What's not supported by Treo smartphones

Treo smartphones use Palm™ OS version 5.x. However, as each licensee can choose to implement only certain features of the OS as it applies to its product, palmOne implements certain features and not others. Here is a list of what Treo smartphones do not support:

- INet library

  palmOne has never supported or ported the INet library, contained in the header file `INetMgr.h`, to its products.

- The `Lz77Mgr.h` header file is not supported.

- `SmsLib.h` header file

  Treo smartphones have their own SMS library. The Treo SMS library supports the Exchange Manager.

- Telephony Manager

  Treo smartphones have their own Phone library. The telephony header files `TelephonyMgr.h`, `TelephonyMgrTypes.h`, and `TelephonyMgrUI.h` are not supported in the Palm OS.

- Fax services are not supported by Treo smartphones.
- The Treo 650 smartphone does not include the `zLib` that was included in the Treo 600 smartphone. A version of this library can be found at: www.copera.com/zlib-armlet/.

Tungsten product line

The latest offerings in the Tungsten product line are designed to target the power business user. Tungsten handhelds provide easy, reliable access to business data, as well as powerful and seamless integration with the desktop business environment. As such, Tungsten handhelds provide a large, easy-to-read display, compatibility with the most popular business applications, a large amount of storage (which is not lost when battery power is depleted), powerful organization and search functions, and the ability to mount the handheld as a drive on a compatible PC available on some models.

What's not supported by Tungsten handhelds

Here is a list of what Tungsten handhelds do not support:

- INet library

  palmOne has never supported or ported the INet library, contained in the header file `INetMgr.h`, to its products.

- The `Lz77Mgr.h` header file is not supported.

- Telephony Manager

  The telephony header files `TelephonyMgr.h`, `TelephonyMgrTypes.h`, and `TelephonyMgrUI.h` are not supported.

Zire product line

The latest offerings in the Zire product line are designed to target the consumer who wants an easy-to-use handheld that is ready out of the box and useful in both a personal and business environment. Zire leverages MP3 capability, a color screen, and expandibility to appeal to casual technology users interested in value, as well as savvy young technology users interested in style and the latest functionality.

What's not supported by Zire handhelds

Here is a list of what Zire handhelds do not support:

- INet library

  palmOne has never supported or ported the INet library, contained in the header file `INetMgr.h`, to its products.

- The `Lz77Mgr.h` header file is not supported.

- SMS library

  The SMS library, included in the header file `SmsLib.h`, is not supported.

- Telephony Manager

  The telephony header files `TelephonyMgr.h`, `TelephonyMgrTypes.h`, and `TelephonyMgrUI.h` are not supported.

Hardware features

Hardware feature	Treo 600	Treo 650	Tungsten T5	Tungsten T3	Tungsten E	Tungsten C	Zire 21	Zire 31	Zire 72
Processor									
Type	TI OMAP ARM	Intel XScale ARM	ntel XScale ARM	Intel Xscale	TI OMAP 311 ARM	Intel PXA255	TI OMAP 311 ARM	Intel ARM	Intel PXA270
Speed	144 Mhz	312 Mhz	412 Mhz	400Mhz	126Mhz	400Mhz	126Mhz	200Mhz	312Mhz
Memory									
RAM	32 MB	32 MB	256 MB	64 MB	32 MB	64 MB	8 MB	16 MB	32 MB
ROM	8 MB	16 MB?	32 MB	16 MB	8 MB	16 MB	2 MB	4 MB	8 MB
Battery									
Type	Rechargeable Lithium Ion	Rechargeable Lithium Ion	Rechargeable Lithium Ion	Rechargeable Lithium Ion	Rechargeable Lithium Ion	Rechargeable Lithium Ion	Rechargeable Lithium Ion	Rechargeable Lithium Ion	Rechargeable Lithium Ion
mAh	1800	1900	1020	900	840	1500	900	900	950
Stand by or use time	300 hours	300 hours	48 hours	???	???	???	???	???	???
Talk time	4 hours	4 hours	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Removable	No	Yes	No	No	No	No	No	No	No
Form factor									
Size	4.41" x 2.36" x .87" without antenna	4.41" x 2.36" x .87" without antenna	4.5" x 3" x .5" without flipcover	4.3" x 3" x .66"	4.5" x 3.1" x .5"	4.0" x 3.07" x .65"	4.4" x 2.9" x .6"	4.4" x 2.9" x .6"	4.6" x 2.95" x .67"
5-way button	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Grafitti	None	None	Dynamic	Dynamic	Yes	None	Yes	Yes	Yes
Keyboard	Built-in QWERTY Keyboard	Built-in QWERTY Keyboard	None	None	None	Built-in QWERTY Keyboard	None	None	None
Hardware feature	**Treo 600**	**Treo 650**	**Tungsten T5**	**Tungsten T3**	**Tungsten E**	**Tungsten C**	**Zire 21**	**Zire 31**	**Zire 72**
Display									
Resolution	160 x 160 pixels	320 x 320 pixels	320 x 480 pixels	320 x 480 pixels	320 x 320 pixels	320 x 320 pixels	160x160 pixels	160x160 pixels	320 x 320 pixels

Density	16-bit (65,536 colors)	16-bit (65,536 colors)	16-bit (65,536 colors)	16-bit (65,536 colors)	16-bit (65,536 colors)	16-bit (65,536 colors)	4-bit grayscale	16-bit (65,536 colors)	16-bit (65,536 colors)
Wireless									
	CDMA or GSM/ GPRS	CDMA or GSM/ GPRS/ EDGE, Bluetooth	Bluetooth	Bluetooth	None	Wi-Fi (802.11)	None	None	Bluetooth
Camera									
	VGA resolution, 640 x 480, 0.3 mega-pixels	VGA resolution, 640 x 480, 0.3 mega-pixels	Photo: 1280 x 960 Video: 320 x 240	None	None	None	None	None	Photo: 1280 x 960 Video: 320 x 240
Interface Connector									
	Treo 600 smartphone connector (USB, Serial without flow control)	Multi-connector (USB, Serial without flow control)	Multi-connector (USB, Serial without flow control)	Universal Connector (USB, Serial with flow control)	Standard Mini-USB	Universal Connector (USB, Serial with flow control)	Standard Mini-USB	Standard Mini-USB	Standard Mini-USB

Software compatibility specifications (palmOne libraries)

Software	Treo 600	Treo 650	Tungsten T5	Tungsten T3	Tungsten E	Tungsten C	Zire 21	Zire 31	Zire 72
PalmOS	5.21	5.4	5.4	5.21	5.21	5.21	5.21	5.28	5.28
Multimedia									
Ring Tones	x	x	-	-	-	-	-	-	-
Sound and Voice Recording	x	x	x	x	-	x	-	-	x
Camera	x	x	x	-	-	-	-	-	x
Photos	x	x	x	x	x	-	-	x	x
Video	-	-	x	-	-	-	-	-	x
Codec Plug-in Manager	-	x	x	-	-	-	-	-	x
Data communications									
Network preferences	x	x	-	-	-	-	-	-	-
HTTP	x	x	x	-	-	-	-	-	-
Network services	-	-	-	-	-	x	-	-	-
Telephony									
Telephony	x	x	-	-	-	-	-	-	-
SMS	x	x	-	-	-	-	-	-	-
System extensions									
Transparency	x	x	-	-	-	-	-	-	-
File browser API	-	-	x	-	-	-	-	-	-
Smart Text Engine	x	x	-	-	-	-	-	-	-
REM sleep	x	x	-	-	-	-	-	-	-
MMS helper functions	x	x	-	-	-	-	-	-	-

Software	Treo 600	Treo 650	Tungsten T5	Tungsten T3	Tungsten E	Tungsten C	Zire 21	Zire 31	Zire 72
NVFS	-	x	x	-	-	-	-	-	-
Handspring extensions	x	x	-	-	-	-	-	-	-
Keyguard	x	x	-	-	-	-	-	-	-
Tips and tutorial	x	x	x	-	-	-	-	-	-
Full-screen writing	-	-	x	x	-	-	-	x	x
Applications									
Web browser	x	x	x	-	-	-	-	-	-
VersaMail®	-	x	x	-	-	-	-	-	-

PART II

Features and Libraries

This part of the guide details the software libraries available in the SDK.

CHAPTER 3

Multimedia

This chapter details the multimedia features and libraries available in the SDK.

Ring tones library

Available on:

- Treo™ 600 and Treo™ 650 smartphones

This section provides reference material for the API functions for the Tones library that is used to manage the phone ring tone database. The `TonesLib.h` header file provided with the SDK contains all the public equates referenced in this section, including all constants, structure definitions, and function prototypes.

Ring tone database

The ring tones for the Treo smartphone by palmOne are stored in a MIDI database similar to the Palm OS® System MIDI Sounds database. This section describes the Treo smartphone ring tone database and the MIDI file format.

The Treo ring tone database is specific to ring tones which means a smartphone can use a Palm OS tone database and a Treo ring tone database at the same time.

Palm OS MIDI File Format

The database type is *smfr*.

Each record in the database should contain one MIDI sound. The MIDI sound format is the Format 0 Standard MIDI File (SMF). For more information on the database format, see the Sound Manager reference information in the *Palm OS Reference Guide* and the Sound section of the *Palm OS Companion Guide* available at www.palmos.com/dev/tech/docs/.

Treo smartphone ring tone database information

The following table contains the details of the system ring tone database.

Database Name	#defined TonesDBName “MIDI Ring Tones”
Database Type	smfr
Creator Code	#define hsFileCMultiChannelRingToneDB MCnl

In order for the system to recognize the new ring tone, the tone must be installed in the database using an alarm management tool. If you are creating ring tone management applications, you may want to keep a separate database of archived ring tones. We recommend that you use the attributes listed in the following table for such a database. You can also use standard Palm OS database calls to install the sound records into this database from their own application. Typically, you would create a MIDI sound database, install it on the smartphone, and then use an alarm management program to copy the sounds into the system ring tone database.

Database Type:	smfr
Creator Code:	HSsf

Restoring the system ring tone database

The ring tone database is stored in RAM in order to allow applications to add and delete ring tones. The OS also has a copy of this database frozen in the ROM image. The database is copied to RAM after a hard reset or if the database has been deleted from RAM. If the user wants to restore the original ring tones, he or she can simply delete the database.

Tools for Ring Tone files

The Treo smartphone uses the standard Palm OS MIDI file for the ring tones. All the popular tools for creating Palm OS system sounds can be used to create ring tones for the smartphone products.

Voice Recording and Sound API

Available on:

- Treo 600and Treo 650 smartphones
- Tungsten™ T5, Tungsten™ T3, and Tungsten™ C handhelds
- Zire™ 72 handhelds

In general, you should use the Palm OS SndStream API to control voice recording and sounds. This API supports both synchronous and asynchronous mode.

The Treo smartphone has a two-speaker audio architecture. One speaker, called the receiver, is mostly dedicated to telephony sound and is tuned to voice frequency. The other, the external speaker, is mostly dedicated to system sounds and is tuned

to polyphonic sounds and is also used for the speakerphone mode. The Treo smartphone also includes the ability to play stereo sound through the headset jack.

Treo smartphones have improved sound support, and their audio subsystem consists of three major categories:

- Radio audio control

  Enables the audio of the cellular radio, playback of received streams, and the encoding and sending of recorded voice.

- System audio control

  Controls the speaker output for polyphonic sound playback and the microphone input for voice recording.

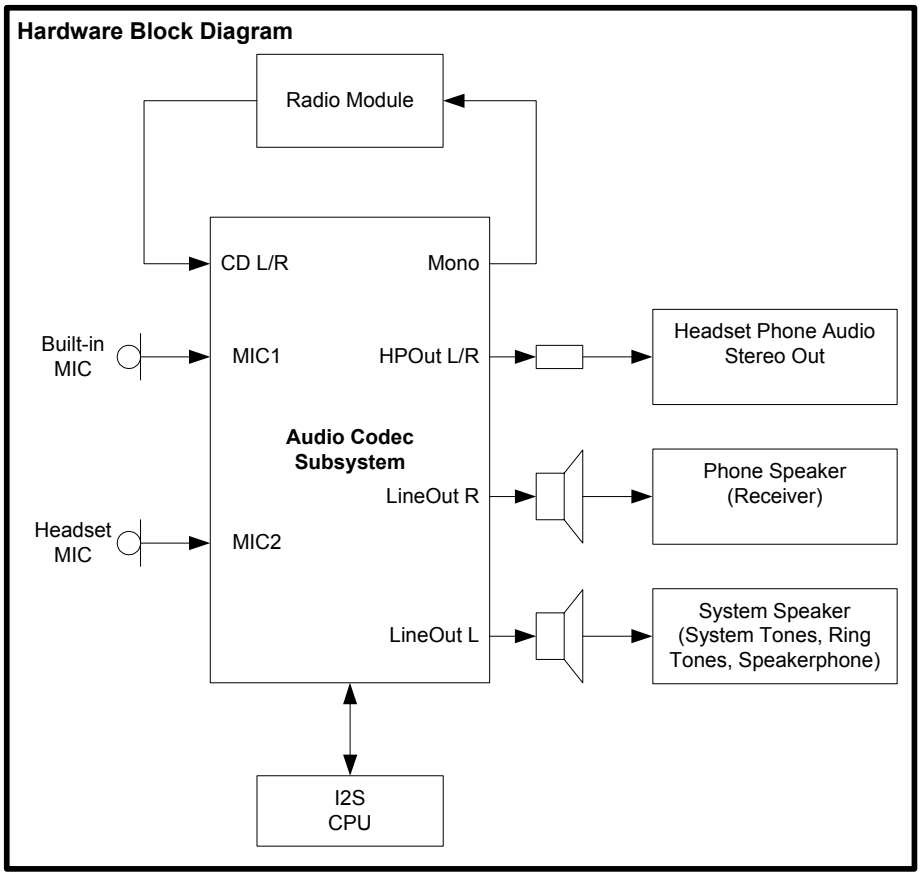
  NOTE The Treo 600 smartphone does not capture the microphone input through an A/D converter.

- Ring Tone Manager

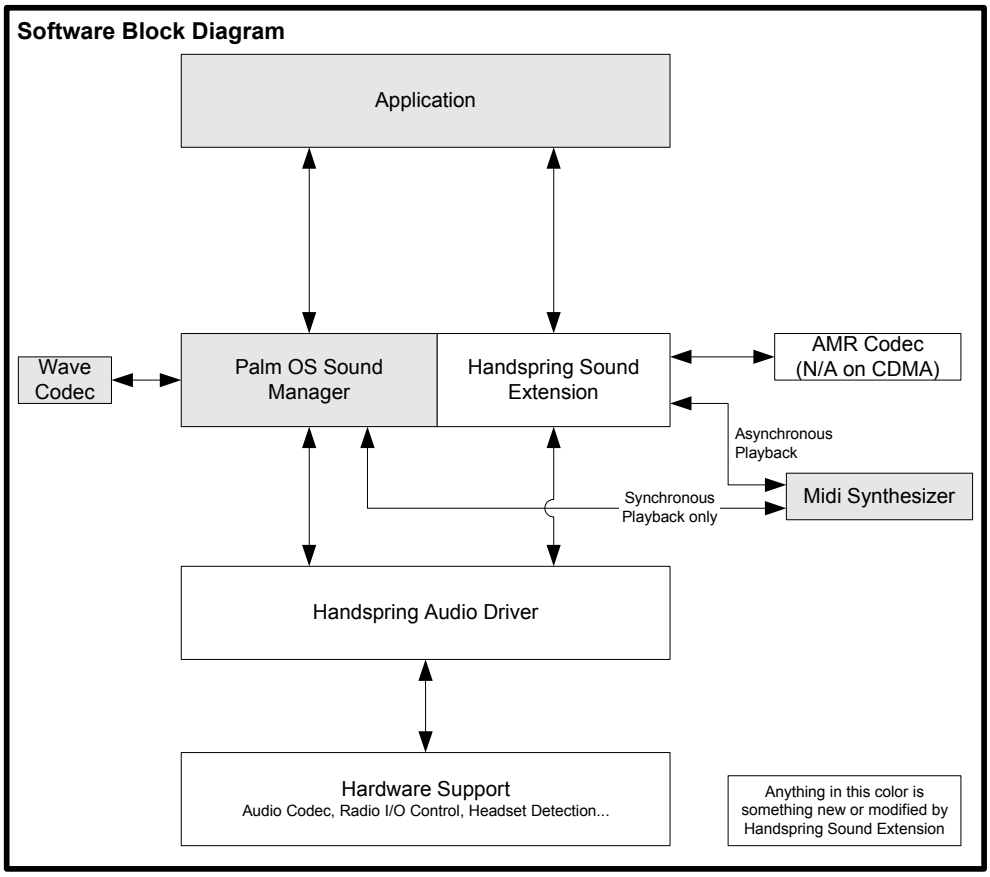
  Controls the playback of ring tones when a call is received.

It's also possible to redirect the radio module audio output to the SDIO connector so that an external card could make use of it. For example, the audio output could be sent to a Bluetooth headset using Bluetooth® wireless technology.

The following figure shows the Treo smartphone audio subsystem hardware.



The following figure shows the Treo smartphone audio subsystem software.



The usual software interface, that is used to play sound is the standard Palm OS Sound Manager. palmOne has added special controls that automatically direct the sound playback to the right output, as described in the following tables.

The only other APIs you need are found in the API guide. These enable control of the audio stream when it is needed. As mentioned earlier, palmOne™ extension to the OS should automatically take care of switching the right input/output, depending on usage.

The usage scenarios in the following tables show how the Treo 600 smartphone system interacts with the audio subsystem when playing or recording audio stream(s) (that is, what types of sounds are routed to which inputs and outputs). Some of the supported scenarios are not currently used by palmOne, but could be created by a third party (6, 7, 10–24).

	Usage Scenario	Voice Sound In	Voice Sound Out	If Phone Sounds*	If System Sounds**
1	Voice call using smartphone only	Built-in mic	Built-in receiver	Mixed-in, built-in speaker	Built-in speaker

	Usage Scenario	Voice Sound In	Voice Sound Out	If Phone Sounds*	If System Sounds**
2	Voice call using headset	Headset mic	Headset speaker	Headset speaker	Built-in speaker
3	Headset plugged in, not on a call	N/A	N/A	Headset speaker and built-in speaker	Built-in speaker
4	Stereo headphone plugged in, not on a call	N/A	N/A	Headset speaker and built-in speaker	Built-in speaker
5	Voice call using speakerphone	Built-in mic	Built-in speaker	Mixed-in, built-in speaker	Mixed-in, built-in speaker
6	Voice call using other type of headset	OT headset mic	OT headset speaker	OT headset speaker	Mixed in OT headset speaker
7	Other type of headset in use, not on a call	N/A	N/A	OT headset speaker & built-in speaker	Built-in speaker
8	Car kit on a call	Car kit mic	Car kit speaker	Mixed-in car kit speaker	Built-in speaker
9	Car kit off a call	N/A	N/A	Car kit speaker	Built-in speaker
10	PTT voice call using headset	Headset mic	Headset speaker	Headset speaker and built-in speaker	Built-in speaker
11	PTT voice call using speakerphone	Built-in mic	Built-in speaker	Mixed-in, built-in speaker	Mixed-in, built-in speaker
12	Voice record memo	Built-in mic	N/A	Built-in speaker—priority	Built-in speaker
13	Voice command using smartphone only	Built-in mic	N/A	Built-in speaker	Built-in speaker
14	Voice command using headset	Headset mic	N/A	Headset speaker and built-in speaker	Built-in speaker
15	Voice command using car kit	Car kit mic	N/A	OT headset speaker	Built-in speaker
16	MP3 music playback—smartphone only	N/A	N/A	Mixed-in, built-in speaker	Mixed-in, built-in speaker
17	MP3 music playback—headset	N/A	N/A	Mixed-in headset and built-in speaker	Mixed-in headset and built-in speaker

	Usage Scenario	Voice Sound In	Voice Sound Out	If Phone Sounds*	If System Sounds**
18	MP3 music playback—stereo headphone	N/A	N/A	Mixed-in headphone and built-in speaker	Mixed-in headphone and built-in speaker
19	Playing games—smartphone only	N/A	N/A	Mixed-in, built-in speaker	Mixed-in, built-in speaker
20	Playing games—headset	N/A	N/A	Mixed-in headset and built-in speaker	Mixed-in headset and built-in speaker
21	Playing games—stereo headphones	N/A	N/A	Mixed-in headphone and built-in speaker	Mixed-in headphone and built-in speaker
22	Voice memo playback—smartphone only	N/A	N/A	Mixed-in, built-in speaker	Mixed-in, built-in speaker
23	Voice memo playback—headset	N/A	N/A	Mixed-in headset and built-in speaker	Mixed-in headset and built-in speaker
24	Voice memo playback—stereo headphones	N/A	N/A	Mixed-in headphone and built-in speaker	Mixed-in headphone and built-in speaker

Notes:

- **Input**
  - **Built-in mic:** The microphone contained in the handset for use when the headset is held to one's ear.
  - **Headset mic:** The microphone contained in the headset.
  - **Other Headset or car kit mic:** The microphone contained in the appropriate peripheral kit.
- **Output**
  - **Built-in receiver:** The speaker on the front of the handset that is used when the handset is held next to the ear. Generally used for listening to voice calls.
  - **Built-in speaker:** The louder speaker located on the back of the handset. Generally used for system sounds and speakerphone mode.
  - **Headset, OT, or car kit speaker:** The speaker built into the appropriate peripheral.
- *"If phone sounds": This refers to how the audio is routed if the audio from a class of sounds dedicated to phone usage interrupts the current usage scenario. This includes the following types of sounds:
  - Ring tones.
  - Call progress tones.

- DTMF (dual tone multi-frequency).
- Low battery. This is technically a system sound, because it is generated by the PDA. It is treated as an exception because a low battery warning is critical phone-related information that must be heard when on an active call, regardless of what the system sound settings are.

- **"If System Sounds": All system sounds will be played through both the headset/headphone and built-in speaker except the System Sounds for Alarms, SMS alerts, and Mail alerts.

The usage scenarios in the following table show how the Treo 650 smartphone system interacts with the audio subsystem when playing or recording audio stream(s).

	Usage Scenario	Ringer switch sound on/ off	On active call?	Telephony audio	Alerts (Attention Manager)	MP3 (application audio)	High-priority system sounds
1	Base mic/speaker	OFF	YES	Receiver	No sound[a]	No sound	No sound
2	Base mic/speaker	OFF	NO	No sound	No sound	No sound	No sound
3	Base mic/speaker	ON	YES	Receiver	Receiver[a]	Mute/hold	Receiver[a]
4	Base mic/speaker	ON	NO	No sound	Speaker	Speaker	Speaker
5	Speakerphone mode	OFF	YES	No sound	No sound[a]	No sound	No sound
6	Speakerphone mode	OFF	NO	N/A	N/A	N/A	N/A
7	Speakerphone mode	ON	YES	Speaker	Speaker[a]	Mute/hold	Speaker[a]
8	Speakerphone mode	ON	NO	N/A	N/A	N/A	N/A
9	Headset inserted	OFF	YES	Headset	Headset[a]	Mute/hold	Headset
10	Headset inserted	OFF	NO	No sound	Headset	Headset	Headset
11	Headset inserted	ON	YES	Headset	Headset[a]	Mute/hold	Headset[a]
12	Headset inserted	ON	NO	No sound	Speaker and mute MP3	Headset	Speaker and mute MP3
13	Bluetooth headset in use	OFF	YES	Bluetooth	No sound	Mute/hold	Bluetooth
14	Bluetooth headset in use	OFF	NO	No sound	No sound	No sound	No sound

	Usage Scenario	Ringer switch sound on/ off	On active call?	Telephony audio	Alerts (Attention Manager)	MP3 (application audio)	High-priority system sounds
15	Bluetooth headset in use	ON	YES	Bluetooth	Speaker	Mute/hold	Bluetooth
16	Bluetooth headset in use	ON	NO	No sound	Speaker	Speaker	Speaker

Notes:

[a] Alerts should be sound mixed-in at a lower volume than the active call in progress.

Camera Manager API

Available on:

- Treo 600 and Treo 650 smartphones
- Tungsten T5 handhelds
- Zire 72 handhelds

This section provides reference information for the Camera Manager API. You can use the functions in this API to control the settings of a camera, to allow an application to capture and preview images and video through the camera, and to turn the camera on.

The Camera Manager API is declared in the header files `palmOneCameraCommon.h` and `palmOneCamera.h`. Camera slider notification information is declared in the header file `PalmCameraSlider.h` for the Zire 72 handheld.

There are three versions of the Camera Manager API: version 1, version 2, and version 3. Version 2 has several additional features available. Version 3 includes minor changes such as additional image formats. The differences are noted throughout this section.

NOTE The Treo 600 smartphone Camera APIs are still supported for the Treo 600. In the future, however, the Camera Manager API should be used.

Using the Camera Manager API

Depending on the camera hardware available on a particular handheld, various Camera Manager settings and functionality may or may not be available. Thus, you should always use the various `Query CamLibControl` command controls to check what features and settings are available. For more information, see the information on `CamLibControl` in the API function descriptions.

Overview of the camera feature

Some handhelds and smartphones include a camera with which users can capture images and video. Some handhelds also include a camera slider that opens to reveal a camera shutter button and the camera lens.

Generally, applications should check for the presence of a camera slider, register for camera slider notifications, and then turn the camera on if the slider is open or if no slider is present.

Typically, an application should follow this general workflow:

1. Open the Camera library.
2. Check to see if a camera slider is present.
3. If a camera slider is present, register for camera slider notifications.
4. If a camera slider is present, check to see if the slider is open.
5. If a camera slider is present and open or no camera slider is present, turn the camera on.
6. Proceed with camera functionality, such as turning on image preview, adjusting settings, and so forth.

Handheld resources required for camera functionality

The following requirements are necessary to take advantage of camera functionality:

- To preview images, capture images, and display the camera configuration dialog box, the handheld must be in 16-bit color-depth mode. The camera configuration dialog box may or may not be present on a particular handheld.
- Depending on the available camera hardware, images can be previewed in 320 x 240 or 160 x 120 resolution and captured in 1280 x 960, 640 x 480, 320 x 240, or 160 x 120 resolution.

Size	Memory
1280 x 960 (SXGA)	2.4MB
640 x 480 (VGA)	600KB
320 x 240 (QVGA)	150KB
160 x 120 (QQVGA)	37.5KB
176 x 144 (QCIF)	50KB
352 x 288 (CIF)	198KB

- Memory, preview, and capture requirements for video are as follows. Video is not available in version 1 of the Camera Manager API, and QCIF and CIF format are not available in version 1 or version 2.

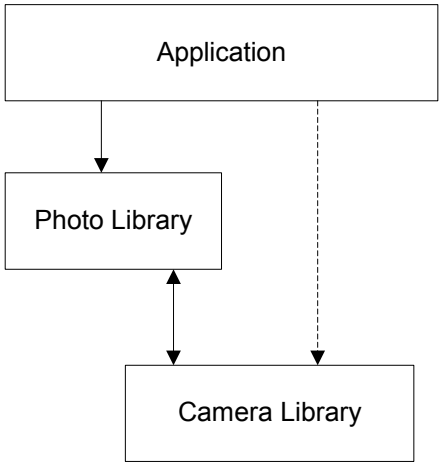
palmOne Photos API

This section provides reference information for the palmOne Photos API. You can use the functions in this API to manipulate digital images in the palmOne Photos application: select images, open and close images, get image information, delete images, and so forth.

palmOne Photos API includes two basic categories of functionality: the ability to capture images with a camera attached to a handheld and the ability to store and manipulate images and video files.

There are three versions of the palmOne Photos API: version 1, version 2, and version 3. Version 2 and 3 have several additional features available. In general, structures, function names, and so forth that end in "V2" are available only in version 2 of the palmOne Photos API and those that end in "V3" are only available in version 3 of the palmOne Photos API. Exceptions are noted where appropriate.

The palmOne Photos API is declared in the header file `palmOnePhoto.h`. (The old name used in early versions, `PalmPhoto.h`, is still compatible but will be deprecated.)



Codec Plug-in Manager API

This section provides reference information for the Codec Plug-in Manager API. The Codec Plug-in Manager API provides a standardized way to load and unload various codecs.

You can use the functions in this API to control starting codec sessions and to encode and decode data using a codec.

The Codec Plug-in Manager API is declared in the header file `palmOneCodecPluginMgr.h`.

How codecs handle individual formats are declared in the header file `palmOneCodecFormat.h`. Because there are many separate formats handled by the Codec Plug-in Manager and new formats may be added later as new codecs are

written or released, the particulars of handling the various formats are beyond the scope of this chapter. Please look at the API guide for the latest information. Also, see the Samples folder in the SDK for examples of how to handle various formats.

Codec Plug-in Manager API Overview

Codecs enumerate all supported formats for each codec included in a module file PRC.

The Codec Plug-in Manager selects a particular codec based on four criteria:

- Input format

  The format of the data input to the codec.

- Output format

  The format of the data output from the codec.

- PRC Creator ID

  The creator ID of the PRC containing the desired codec on the device. This is optional; you can have the Codec Plug-in Manager select an appropriate codec based on the input and output formats. PRC Creator IDs are unique and must be registered with PalmSource.

- Codec ID

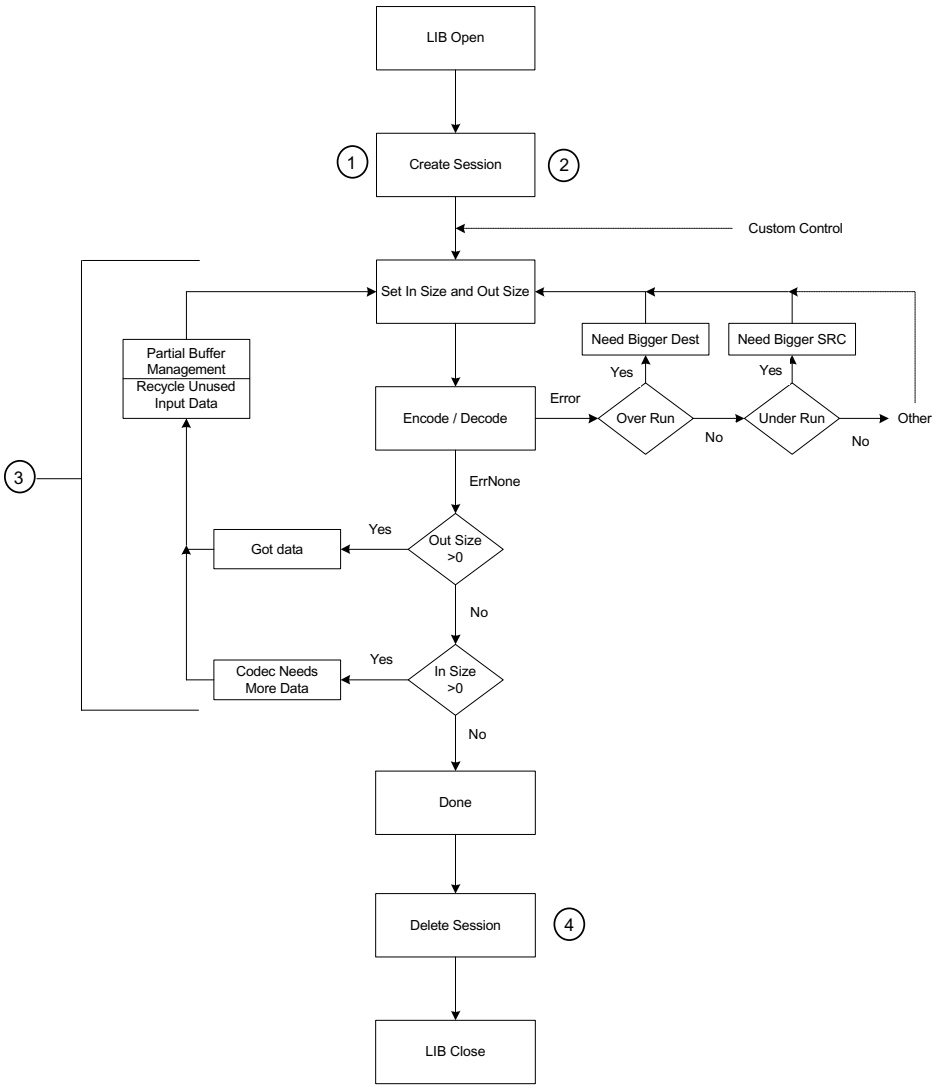
  The codec ID of the codec. This is provided because a PRC file can contain multiple codecs that have the same input and output formats. Again, this is optional; you can have the Codec Plug-in Manager select an appropriate codec based on the input and output formats. The Codec ID is unique within each PRC.

Codec Plug-in Manager process

The process of using the Codec Plug-in Manager is as follows:

1. Create a session.
2. Specify the input and output formats and, optionally, which particular codec to use.
3. Start decoding or encoding the output.

4. Delete the session once the decoding or encoding is complete.



Codec media formats

You can find a complete list of the codec media formats currently specified in the `PalmCodecFormat.h` section of the API guide. Each media format is denoted by a four-byte string (for example, `JPEG`).

Each media format has an associated generic typedef structure that contains the parameters associated with that format. These structures are passed during codec session initialization. Each codec supports some or all of the parameters in such structures. Some parameters inside the structure might not apply to a specific vendor's codec and therefore can be ignored. Each parameter is of the type `UInt32` to enable simple byte-swapping.

In general, however, codecs should always use most of the parameters in each structure. And if a parameter structure's content is `NULL`, codecs should use default values if possible. For example, encoding PCM format to ADPCM format doesn't require parameters for the ADPCM format, because all of the ADPCM parameters can be derived from the PCM parameter values.

CHAPTER 4

Data Communications

This chapter details the data communications features and APIs available in the palmOne™ SDK

NetPref library API

Available on:

- Treo™ 600 and Treo™ 650 smartphones

This section provides detailed information about the NetPref library API.

The NetPref library was created to provide better support for the GSM/GPRS and 1XRTT network parameters, dynamic UI flags, Home/Roaming network configurations, CCSM database utilization, synchronization with IOTA-provisioned settings, and configuration of fallback services required by the features of the Treo™ smartphone. The network database access was redesigned by splitting database and record access operations from the Network panel into the NetPref library.

Loading the library

The NetPref library is designed as a Palm OS® shared library. The NetPref library should be loaded for use and unloaded after use by a client application. You "link" the NetPref library when you need it and "unlink" it when you have finished. The system does not load the NetPref library at reset or start-up time and leave it permanently installed, as is done with some other libraries. This method helps avoid some HotSync® conflicts, such as an attempt to install over a protected database. For examples of linking and unlinking, take a look at `NetPrefUtilNetPrefLibLink` and `NetPrefUtilNetPrefLibUnlink` as defined in the `NetPrefUtils` package.

The following code sample demonstrates how to link the NetPref library based on the `NetPrefUtil` package.

```
extern Boolean
NetPrefUtilNetPrefLibLink (NetPrefUtilNetPrefLibType* netPrefLibP)
{
   Boolean          isSuccessful = false;
   Err              err = 0;
   UInt16           refNum = 0;
   NetPrefContextTypeTag* cxtP = NULL;
```

```
ErrNonFatalDisplayIf (!netPrefLibP, "null arg");
  ErrNonFatalDisplayIf (netPrefLibP->linkSignature
              == netPrefUtilNetPrefLinkSignature,
              "NetPref lib already linked");
   err = SysLibLoad (netPrefLibTypeID, netPrefLibCreatorID,
                     &refNum);
   if (err)
      {
      ErrNonFatalDisplay ("failed to load NetPrefLib");
      goto Exit;
      }
   err = NetPrefLibOpen (refNum, &cxtP);
   if (err)
      {
      ErrNonFatalDisplay ("failed to open NetPrefLib");
      goto Exit;
      }
   // "Construct" the NetPref lib "instance"
   isSuccessful = true;
Exit:
   if (err)
      {
      if (refNum != 0)
         SysLibRemove (refNum);
      MemSet (netPrefLibP, sizeof(*netPrefLibP), 0);
      }
   return (isSuccessful);
} // NetPrefUtilNetPrefLibLink
```

NetPref library Information

The following table shows the attributes of the NetPref library and related information. For more detail, see the API Guide.

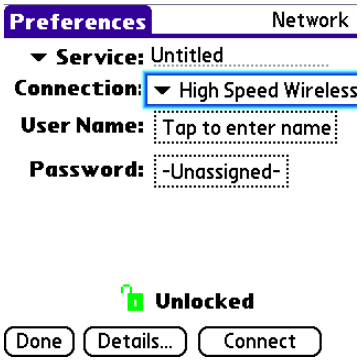
Description	Attribute
Creator ID	`HsNP`
Type ID	`libr`
Library Database Name	`NetPrefLibrary`
Library Name	`HsNetPrefLibrary.lib`

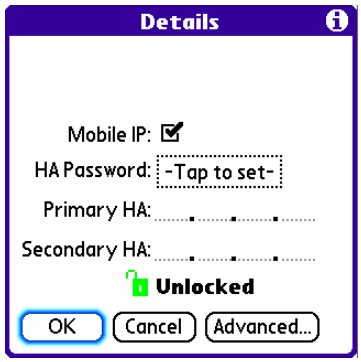
Header Files	`NetPrefUtils.h`
	`NetPrefLibrary.h`
	`NetPrefLibTypes.h`
	`NetPrefLibErrors.h`
	`NetPrefLibTarget.h`
	`NetPrefLibFieldInfoTable.h`

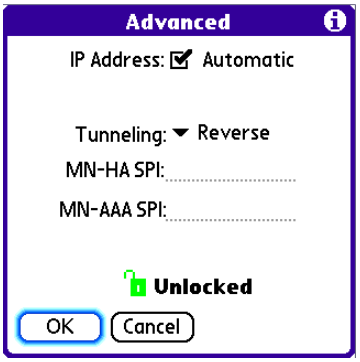
NetPref panel

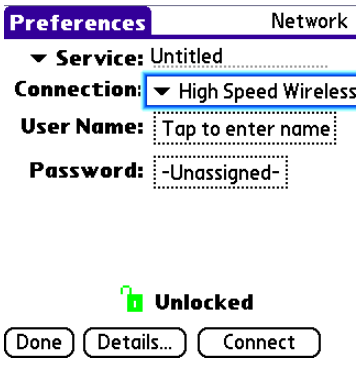
The Network Preference panel has been modified to provide support to the various Treo smartphone features that are not possible by using the original Palm OS (3.5 and 5.x) Network Preference panel implementation. Parameters specific to GSM/GPRS and CDMA/1-X (Simple-IP & Mobile-IP), IOTA support, CCSM table support, and our various UI features such as hiding certain fields and locking certain services were added. The network database record format was extended in a backward-compatible way, and the network database access logic was separated into the NetPref library. The new Network Preferences panel as well as other system components, such as NetMaster library, the IOTA application, and network profile creator, use the NetPref library to read, write, create, duplicate, and delete network service profiles.

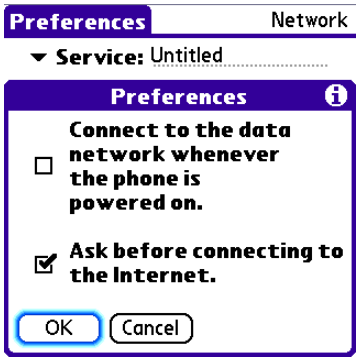
In addition, the configuration of NetLib was redesigned to dynamically perform during each network login instead of doing so only when a user selects a service. This dynamic configuration implementation was moved from the Network panel to the NetMaster library, which is described later in this section. This permits the support of dynamic network configuration based on location, such as Home versus Roaming, the execution of GPRS and One-X specific functions during login, the implementation of the service fallback feature, and so forth. The following are examples of Preference Panel displays.











The new Network Preference panel will continue to support the legacy Network Preference panel's limited launch code API, such as enumerating profile names, getting or setting the default profile, and so forth. Applications such as the HotSync® application use this API to select the appropriate network preference profile. The new implementation provides full backward-compatibility with the legacy API.

- `sysAppLaunchCmdPanelCalledFromApp`

  Displays the network panel as if it were a dialog box popped up from the calling app, returning to the calling app when the dialog box is dismissed by tapping the Done button, for example.

- `svcCFACmdQuickEdit`

  Manifestation of sysAppLaunchCmdPanelCalledFromApp that brings up the "quick- edit" form of the panel, such as the phone number form for a dial-up service profile.

  - `sysSvcLaunchCmdSetServiceID`: Sets the default service.
  - `sysSvcLaunchCmdGetServiceID`: Gets the default service.
  - `sysSvcLaunchCmdGetServiceList`: Gets a list of service names and corresponding IDs.
  - `sysSvcLaunchCmdGetServiceInfo`: Gets the service name when the unique ID of the service is given.
  - `sysSvcLaunchCmdGetQuickEditLabel`: Gets the "quick-edit" value string to display to the user, such as the phone number value from a dial-up service profile.

NOTE The network preference database has been restructured to support the new parameters needed by the persistent data connection. Any preexisting application that reads, creates, or modifies network services or profiles directly in the original Palm OS network preference database will probably not work with the products that support this new architecture. If standard Palm OS 3.5 APIs were used, such applications might still work. Legacy profiles are supported because they are converted to the new structure of the database the first time they are accessed. To create or read network profile information in new applications, use the NetPref library. The library provides all the necessary routines to interface with the network preference database. There is no longer any need to read/write directly to the network preference database.

HTTP library

Available on:

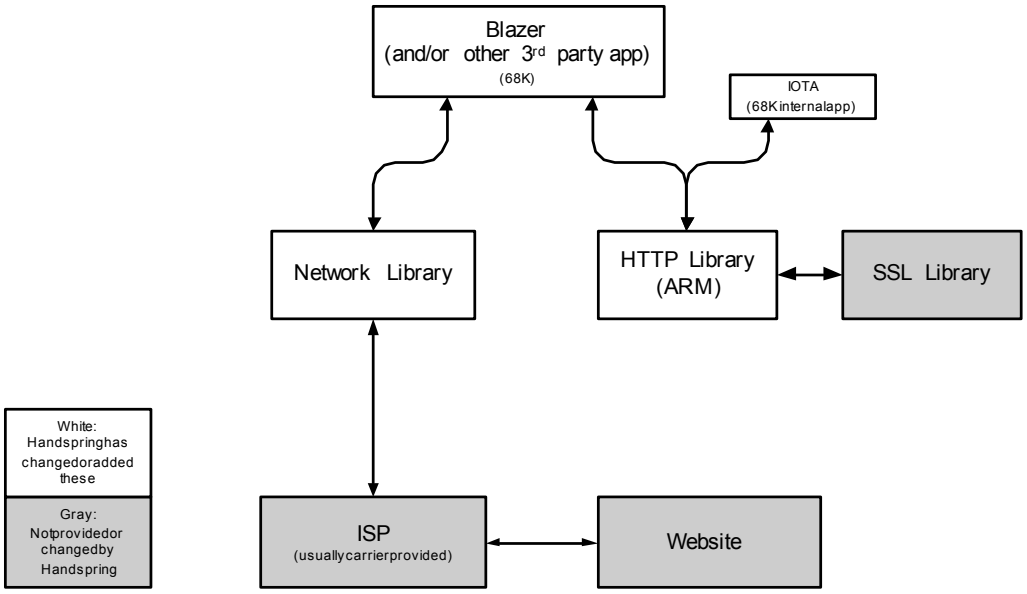
- Treo™ 600 and Treo™ 650 smartphones
- Tungsten™ T5 handhelds

The HTTP library is a shared library. This HTTP library was added to Palm OS 5.X to give applications a high-level interface to implement HTTP access.

The HTTP library is used by the palmOne™ web browser and other applications. Third-party applications can share and use the HTTP library to implement their own HTTP access requirements. The HTTP library supports HTTP protocol versions 1.0 and 1.1 and the WAP 2.0 HTTP client profile. The HTTP library also supports SSL using the Palm OS SSL library for secure (HTTPS) connections, proxy configuration, and cookies.

Architecture

The following figure shows the architecture of the HTTP library. The HTTP library is provided at the Palm OS library level. The HTTP is implemented at a peer level with the Palm OS NetLib library and SSL library. Applications must call the NetLib library to initialize and enable the TCP sockets for the HTTP library. The HTTP library will call the SSL library directly to implement authentication and cryptography necessary for secure access. Thus, after standard initialization of the NetLib library, the applications will then directly call into the HTTP library to implement all their HTTP accesses.



Functional highlights

The HTTP library is designed for easy sharing by multiple applications. The HTTP library is implemented as a single instance running in the execution task of caller applications. The library does not launch any additional tasks and is safe to use as a background task, because it does not display any UI. The library does provide the necessary callbacks to allow applications to handle UI in the application code, if necessary.

The HTTP library requires applications to handle the NetLib library interface by passing the HTTP library a reference to NetLib wrapper function callbacks.The HTTP library intelligently manages the pool of sockets. The library provides the calling application an API to specify the maximum number of sockets used by the HTTP library.

The HTTP library supports the following:

- HTTP versions 1.0 and 1.1 as defined in RFCs 1945 and 2616, and the WAP 2.0 HTTP client profile
- Basic, digest, and proxy authentication as specified in RFC 2617
- SSL (HTTPS) connections
- GZIP and Zlib (using deflate) compression formats
- Chunked encoding allowing applications to begin sending data before the application knows the total amount of data to be sent
- Individual application user agent profile

  An application can control the user agent profile by adding a simple string or WAP 248 UA Prof to the request header.
- Keep-alive connections

  However, keep-alive connections are not shared among applications. Different sockets are used for multiple keep-alive connections. For example, two sockets are used to open two separate connections to www.yahoo.com.
- Non-blocking sockets

HTTP library interface to SSL

The HTTP library includes support for SSL. In Treo smartphones, the HTTP library uses the SSL and crypto module libraries provided in Palm OS 5.x and later. These two libraries provide sufficient crypto functionality to negotiate secure SSL connections with most secure web sites.

Palm OS 5.x and later includes two security-related shared libraries: the SSL library and the Cryptography Provider Manager.

The SSL library in Palm OS 5.x and later includes:

- SSL protocol implementation, version 3 only—not SSL v2 or TLS v1
- RSA public/private key algorithm for key exchange

- RC4 symmetric cipher for bulk data encryption
- Authentication of the server-side of the connection using digital certificates and signature verification
- Message verification using the MD5 and SHA-1 hash algorithms
- SSL session resumption

The SSL library in Palm OS supports 4 SSL cipher suites:

- `sslCs_RSA_RC4_128_MD5`
- `sslCs_RSA_RC4_128_SHA1`
- `sslCs_RSA_RC4_56_SHA1`
- `sslCs_RSA_RC4_40_MD5`

The first two cipher suites include a connection that uses the RC4 symmetric cipher with a 128-bit key for data encryption, the RSA algorithm for key exchange, and either the MD5 or the SHA1 hash algorithm for verifying message integrity. These two cipher suites are widely supported by popular web servers on the Internet.

The other security-related library included in Palm OS is the Cryptography Provider Manager (CPM). The CPM exports an API that allows developers to perform specific cryptographic operations. The CPM in Palm OS provides access to the following:

- RC4 symmetric cipher, variable key length
- SHA-1 hash algorithm
- Message verification

HTTP library use of Certificates/Public Key Infrastructure

The HTTP library reads a set of root certificates from a database on start-up. These root certificates identify each of the major certificate authorities and are used by the HTTP library during SSL connection establishment to authenticate the remote web server. The database containing the root certificates is burned into the ROM with the HTTP library.

Certain large corporations and institutions act as certificate authorities and issue their own certificates. Such a corporation or institution then uses certificates signed using its self-issued root certificate to identify its secure web servers. In order for a user's web browser to negotiate a secure SSL connection with a web server identified by the corporation's certificate, the user must add the corporation's self-issued root certificate to his or her web browser's set of trusted root certificates. The HTTP library does not include a mechanism that allows the user to add trusted root certificates to the certificate database on the user's device. However, you can design your calling application to check remote server's certificates and display a UI to add them as trusted.

HTTP library implementation

The HTTP library APIs can be categorized into these four groups:

- Palm OS library management
- Library initialization and finalization
- Stream operations
- SSL

The Palm OS library management APIs include functions to open, close, sleep, wake, or count the HTTP library.

The initialization and finalization APIs include functions to load, open, initialize, close, and remove the HTTP library.

The stream operation APIs include functions to create, configure, send, and receive requests and responses. They also include functions to load, open, initialize, close, and remove the HTTP library.

The SSL category APIs include functions to authenticate, certify, encode, and decode secure connections.

See the API guide for more details about the available HTTP APIs.

General HTTP program information

The following pseudo code demonstrates the usage model of a typical client application using the HTTP library:

```
Library Open
   find or load HTTP library
Library Initialize
   Initialize global environment variables:  application,
   netlib, peer
   Open NetLib library
   Initialize HTTP library.
   Set up connection time-outs
   (Confirm certification)
   (Set proxy)
Stream Create
Stream Initialize
Stream Send Request and Read Response
   Send request
   Loop on Read Response until Read terminates
Stream Close
Library Finalize
Library Close
```

Initialization: The following sample code demonstrates the HTTP library initialization. The sample code shows that the initialization includes opening both the HTTP library and the NetLib library, setting all the callback for calling NetLib's TCP socket functions, and setting the application execution environments through the three global variables structures—struct `HS_HTTPLibNetInfo`, struct `HS_HTTPLibPeer`, and struct `HS_HTTPLibAppInfo`.

The application's SSL certificate, proxy connection, and connection timeouts are also initialized.

```
void HTTPLibInitialize(void)
{
err = HS_HTTPLibOpen(gRefNum);

   /* gPeer */
   MemSet(&gPeer, 0, sizeof(HS_HTTPLibPeer));
   gPeer.HS_HTTPLibPeerTCPOpen = &PrvTCPOpen;
   gPeer.HS_HTTPLibPeerTCPClose = &PrvTCPClose;
   gPeer.HS_HTTPLibPeerTCPIsConnected = &PrvTCPIsConnected;
   gPeer.HS_HTTPLibPeerTCPConnect = &PrvTCPConnect;
   gPeer.HS_HTTPLibPeerTCPRead = &PrvTCPRead;
   gPeer.HS_HTTPLibPeerTCPWrite = &PrvTCPWrite;
   gPeer.HS_HTTPLibPeerTCPCanReadWrite = &PrvTCPCanReadWrite;

   gAppInfo.maxSockets = 3;
   gAppInfo.isForeground = true;
   gAppInfo.cookieMaxJarSize = (UInt16)300 * (UInt16)1024;

   PrvPeerTCPInitialize();
   gLibHandle = HS_HTTPLibInitialize(gRefNum, &gAppInfo,    &gNetLibInfo,
&gPeer);
```

```
   /* set callbacks */
   //HS_HTTPLibSetSSLServerCertConfirmProc(gRefNum,    gLibHandle,
&test_confirm_cb,    (HS_HTTPLibOpaque)gLibHandle);
   //HS_HTTPLibSetTunnelingCallback(gRefNum, gLibHandle,
   &PrvTunnelingCallback, NULL);

   /* set timeout time */
   HS_HTTPLibSetConnectTimeOut(gRefNum, gLibHandle, -1);
   HS_HTTPLibSetReqTimeOut(gRefNum, gLibHandle, -1);
   HS_HTTPLibSetRspTimeOut(gRefNum, gLibHandle, 10 * 1000);

   /* set proxy info if used*/
   //HS_HTTPLibSetProxy(gRefNum, gLibHandle, ProxyHost,
   StrLen(ProxyHost), ProxyPort, ProxyPort, NoProxyHost, 0);
   //HS_HTTPLibSetUseProxy(gRefNum, gLibHandle, true);
}
.................
.................

/* Wrappers for NetLib callbacks */
void PrvPeerTCPInitialize( ) {.... NetLibOpen( .... }

Int32 PrvTCPOpen( ) {..... NetLibSocketOpen( .... }

void PrvTCPClose( ) { .... NetLibSocketClose( .... }

Int32 PrvTCPIsConnected( ) {.... NetLibSelect( .... }

Int32 PrvTCPConnect( ) {.... NetLibSocketConnect( .... }

Int32 PrvTCPRead( ) {..... NetLibReceive( .... }

Int32 PrvTCPWrite( ) {.... NetLibSend( .....}

Int32 PrvTCPCanReadWrite( ) {....  NetLibSelect( .... }
```

Finalization: The following sample code demonstrates the HTTP library finalization. The finalization appears as a short sequence of freeing memory and closing the NetLib and HTTP libraries.

```
void HTTPLibInitialize(void)
{
HS_HTTPLibFinalize(gRefNum, gLibHandle);
PrvPeerFinalize();
HS_HTTPLibClose(gRefNum, &count);
}

void PrvPeerTCPFinalize( ) { .... NetLibIFDown( .... }
```

Processing Loop: The HTTP library does not contain an internal processing loop, so applications should implement a processing loop to send and receive incoming stream data. Applications can implement the processing loop as a state machine that models the sequence of communication between the client and the server through the HTTP protocol. The following example shows how an HTTP sequence is modeled into a state machine.

Common HTTP accesses follow this sequence pattern:

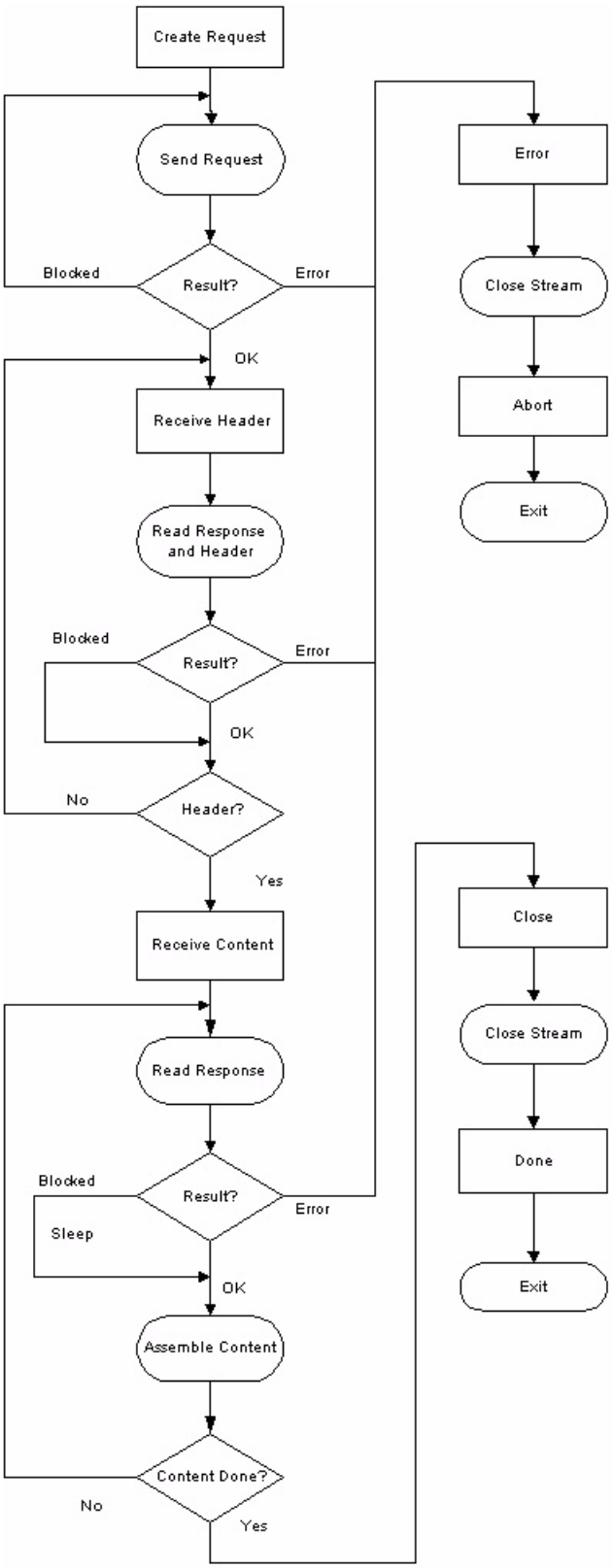
1. Generate an HTTP request and send the request.
2. Read the response header.
3. Read the content of the data stream.
4. Continue reading content until the finish or until some error occurs.
5. Close the stream connection.

Applications can define the preceding state transitions into the state machine to process streaming data.

Applications can also implement the following state transition pattern for successful HTTP accesses in the state machine:

1. Initiate the state machine with a request creation with the state `kDownloadState_Request`.
2. Through the state machine, send a request out and switch the state to `kDownloadState_ReceiveHeader`.
3. Read the initial response and header and then transition to the state `kDownloadState_ReceiveContent`.
4. Continue to loop and read the response until the data is finished, and then transition to the state `kDownloadState_Close`.
5. Close the stream, and switch to the state `kDownloadState_Done`.
6. Exit with the condition `httpErrorOK`.

An error can occur during the state `kDownloadState_Request_xyz` or `kDownloadState_Receive_xyz` when switching to the states `kDownloadState_Error`, `kDownloadState_Cancel`, or `kDownloadState_Abort`. These states trigger a loop exit, and the appropriate error conditions are recorded.



The sample code in the palmOne sample application call `HTTPLibTest` includes code for an application's processing loop to send an HTTP request and receive the response through a state machine. This processing loop can be modified to adapt to other applications. Refer to the `HTTPLibTest` project.

Net Services API

Available on:

- Tungsten™ C handhelds

This section provides reference information for the Net Services API. You can use the functions in this API to check a handheld's radio hardware, to add or change network user profiles, and to implement other 802.11 network service tasks.

The NetServices library is available only on handhelds equipped with Wi-Fi, either through built-in hardware resident on the device or through a Wi-Fi add-on accessory.

The Net Services API is declared in the header file `PalmNetServices.h`. The Net Services API also uses data structures declared in the header file `PalmWiFiCommon.h`.

Overview of the Net Services feature

Handhelds that include Wi-Fi (wireless fidelity) functionality include a panel that allows users to add network profiles in order to connect to different 802.11 Internet access points. Users can choose different profile names, SSIDs (service set identifiers), encryption methods, and other profile characteristics based on the requirements of the access points and on their own personal preferences.

Using the Net Services API, you can create your own panel to allow users to create profiles, to specify encryption methods, and to connect to access points.

NOTE The Net Services API does not provide a method to add profiles to the existing Wi-Fi panel. If you create a Net Services application, you must design your own panel to display and create profiles, connect to access points, and so forth. Furthermore, if you plan to add, delete, or modify your own separate set of profiles, you should design your own panel because the default Wi-Fi panel may overwrite the profiles you create and modify. If you simply want to replace the default Wi-Fi panel, set the creator ID and type of the panel you create to that of the default Wi-Fi panel.

CHAPTER 5

Telephony

This section provides reference material for the telephony APIs in the palmOne™ SDK.

Overview of the Telephony API libraries

Available on:

- Treo™ 600 and Treo™ 650 smartphones

The `HsPhone.h` header file provided with the Phone SDK kit contains all the public equates referenced in this section, including all constants, structure definitions, and function prototypes. Refer to the API guide for more details.

The telephony libraries include the following categories:

Phone library Category	Description
SMS	Functions that apply to sending, receiving, and managing SMS messages. Declared in `HsPhoneSMS.h`.
GSM	Functions that apply to GSM type products with some functions that apply to both GSM and CDMA type products. Declared in `HsPhoneGSM.h`.
CDMA	Functions that apply only to the CDMA type product.

The telephony libraries depend on a set of mostly common structures, types, and enumerations. You can also look at the latest version of the header files in the palmOne™ Header package under Common/Libraries/Telephony for the most up-to-date definitions.

NOTE The palmOne Telephony library does not support third-party applications that control phone calls directly. The supported method is the use of the helper application described in "Launching the Phone application in a specific view."

Because the telephony libraries contain so many different enumerations, structures and types, the telephony header files are divided into categories so that they are easier to understand. These categories are defined in the following table.

Category	Description
Library	General Palm OS® library functions and enumerations related to the telephony libraries. Declared in `HsPhoneLibrary.h`.
Network	Cellular network types and functions. Declared in `HsPhoneNetwork.h`. Developers should have a basic knowledge of how a cellular network behaves. The following concepts should be familiar to you if you want to use the Telephony Network APIs: ■ Current operator versus home operator ■ How to select the current operator when more than one is available ■ Phone registration process with the wireless network ■ Interactions with a SIM card ■ Voicemail box interaction from the network
Audio	Specific audio definition: ringing profile and slider switch.
Events	Telephony events sent to an application that registers for them. Its main enum is `PhnEventCode`, which contains the notification received by a registered application. Refer to the sample code in the SDK.
Security	Password Type enumerations.
IOTA	Internet Over The Air enumerations.
Misc	Other functions. Declared in `HsPhoneMisc.h`.

NOTE The telephony libraries support the specific requirements of the Treo 600 smartphone and the Treo 650 smartphone. Treo 600 and Treo 650 smartphones do not support the Palm OS telephony functions. If you want to use the telephony functions on the Treo smartphone, you must use the palmOne Telephony Library functions.

CDMA and GSM library differences

The CDMA and GSM Telephony APIs are almost the same. However, because the actual radio architecture differs, palmOne has two different libraries as defined in the following table. It is important to use the correct library for the correct radio architecture when using the Telephony APIs.

Library	Constant	Value	Description
Basic definition	`phnLibDbType`	`libr`	Database type ID
CDMA			
	`phnLibCDMADbCreator`	`HsCL`	CDMA library creator ID
	`phnLibCDMADbName`	`Phone Library`	CDMA library Database name
	`phnLibCDMAName`	`PhoneLib.prc`	CDMA library name
GSM			
	`phnLibGSMDbCreator`	`GSM!`	GSM library creator ID
	`phnLibGSMDbName`	`Phone Library`	GSM library Database name
	`phnLibGSMName`	`GSMLibrary.lib`	GSM library name

GSM Connected indicator

On a Treo smartphone, if you want your application to determine whether the smartphone is registered with a network, use the Telephony library function `PhnLibRegistered`.

You may want your application to update the GSM Connected indicator when the following phone events occur:

- `phnEvtRegistration`
- `phnEvtError`
- `phnEvtIndication`
  - `indicationNetworkAvailable`
  - `indicationStartingRadio`
  - `indicationResettingRadio`
  - `indicationPoweringOffRadio`

NOTE To convey whether the phone is connected to a network, the simplest approach is to use the palmOne system Signal gadget. (See "How to include the Signal gadget" for more information.) It appears when the phone is connected to a network, and does not appear when the phone is not connected.

Operator's Name indicator

To retrieve the current operator's name, use the Telephony library function `PhnLibCurrentOperator`.

You may want your application to update the Operator's Name indicator when the following phone events occur:

- `phnEvtRegistration`
- `phnEvtError`
- `phnEvtIndication`
  - `indicationNetworkAvailable`
  - `indicationStartingRadio`
  - `indicationResettingRadio`
  - `indicationPoweringOffRadio`

Voicemail indicator

To retrieve the current status of Voicemail, use the Telephony library function `PhnLibBoxInformation`.

You should update the Voicemail indicator in your application when a `phnEvtVoiceMail` event occurs. `phnEvtVoiceMail` events occur when new voicemail messages are received and when users clear their voicemail.

Launching the Phone application in a specific view

This section details how to launch the Phone application on Treo 600 and Treo 650 smartphones in a specific view. You can also refer to the sample code in the SDK for complete details.

Required headers

The headers required to launch the Phone application in a specific view are as follows:

- `Common/System/HsAppLaunchCmd.h`

  This header file includes Phone application launch commands and corresponding launch command parameter structures.

- `Common/System/HsCreators.h`

  This header file includes the Phone application creator type.

Launching the Phone application in Call Log view

To launch the Phone application in the Call Log view, use the `phoneAppLaunchCmdViewHistory` launch command as follows:

```
DmGetNextDatabaseByTypeCreator ( true,
                                 &searchState,
                                 sysFileTApplication,
                                 hsFileCPhone,
                                 true,
                                 &cardNo,
                                 &dbID);

err = SysUIAppSwitch( cardNo,
                      dbID,
                      phoneAppLaunchCmdViewHistory,
                      NULL /*paramsP*/);
```

Launching the Phone application in Dial Pad view

When launching the Phone application in Dial Pad view, you can do one of the following:

- Launch in Dial Pad view and automatically dial a specified number.
- Launch in Dial Pad view and prefill the number field in the Dial Pad view with a specified number without automatically dialing it.
- Launch in Dial Pad view with no number filled in.

To launch in Dial Pad view and automatically dial a phone number

Use the following code as reference:

```
PhoneAppLaunchCmdDialPtrparamsP = NULL;
UInt16 size = sizeof(PhoneAppLaunchCmdDialType);
Char* numberP = <THE_PHONE_NUMBER_YOU_WANT_AUTO_DIALED>;

// Set up a parameter block so the Phone application
// automatically dials a phone number

if (numberP)
  {
size += StrLen(numberP) + 1;
  }

paramsP = MemPtrNew (size);
MemSet (paramsP, size, 0);

paramsP->version = 1;
paramsP->failLaunchCreator = <YOUR_APP_CREATOR>;
if (numberP)
  {
paramsP->number = MemPtrNew (StrLen(numberP) + 1);
    StrCopy(paramsP->number, numberP);
    MemPtrSetOwner (paramsP->number);
  }

MemPtrSetOwner (paramsP, 0);

DmGetNextDatabaseByTypeCreator ( true,
                                 &searchState,
                                 sysFileTApplication,
```

```
                                    hsFileCPhone,
                                    true,
                                    &cardNo,
                                    &dbID);

err = SysUIAppSwitch( cardNo,
                      dbID,
                      phoneAppLaunchCmdDial,
                      paramsP);
```

To launch in Dial Pad view and prefill the number field

Use the following code as reference:

```
PhoneAppLaunchCmdDialPtrparamsP = NULL;
UInt16 size = sizeof(PhoneAppLaunchCmdDialType);
Char* numberP = < PHONE_NUMBER_TO_PREFILL_FIELD_WITH>;

// Setup a parameter block so the Phone application pre-fills
// a phone number in the Dial Pad number field

if (numberP)
  {
size += StrLen(numberP) + 1;
  }

paramsP = MemPtrNew (size);
MemSet (paramsP, size, 0);

paramsP->version = 1;
paramsP->failLaunchCreator = <YOUR_APP_CREATOR>;
if (numberP)
  {
   paramsP->number = MemPtrNew (StrLen(numberP) + 1);
   StrCopy(paramsP->number, numberP);
   MemPtrSetOwner (paramsP->number);
  }

MemPtrSetOwner (paramsP, 0);

DmGetNextDatabaseByTypeCreator ( true,
                                 &searchState,
                                 sysFileTApplication,
                                 hsFileCPhone,
                                 true,
                                 &cardNo,
                                 &dbID);

err = SysUIAppSwitch( cardNo,
                      dbID,
                      phoneAppLaunchCmdViewKeypad,
                      paramsP);
```

To launch in Dial Pad view without a phone number

Use the following code as reference:

```
DmGetNextDatabaseByTypeCreator ( true,
```

```
                                    &searchState,
                                    sysFileTApplication,
                                    hsFileCPhone,
                                    true,
                                    &cardNo,
                                    &dbID);

err = SysUIAppSwitch( cardNo,
                      dbID,
                      phoneAppLaunchCmdViewKeypad,
                      NULL /*paramsP*/);
```

Launching the Phone application in the Favorites view

To launch the Phone application in the Favorites view, use the `phoneAppLaunchCmdViewSpeed` launch command as follows:

```
DmGetNextDatabaseByTypeCreator ( true,
                                    &searchState,
                                    sysFileTApplication,
                                    hsFileCPhone,
                                    true,
                                    &cardNo,
                                    &dbID);

err = SysUIAppSwitch( cardNo,
                      dbID,
                      phoneAppLaunchCmdViewSpeed,
                      NULL /*paramsP*/);
```

Launching the Contacts application with the New Contact window open

On Treo smartphones, you can launch the Contacts application with the New Contact window open.

Required headers

The headers required for launching the Contacts application with the New Contact window open are as follows:

- `Common/System/HsAppLaunchCmd.h`

  This header file includes Contact application launch commands and corresponding launch command parameter structures.

- `Common/System/palmOneCreators.h`

  This header file contains the Contacts application creator type.

To launch the Contacts application with the New Contact window open, use the `addrAppNotificationCreateNewRecord` launch command as follows:

```
notifyParamP = MemPtrNew(sizeof(SysNotifyParamType));
```

```
MemSet(notifyParamP, notifyParamSize, 0);

   notifyParamP->notifyType = addrAppNotificationCreateNewRecord;
   notifyParamP->broadcaster = <YOUR_APP_CREATOR>;
   notifyParamP->notifyDetailsP = NULL;
   notifyParamP->handled = false;

DmGetNextDatabaseByTypeCreator( true,
                                &searchState,
                                sysFileTApplication,
                                kPalmCreatorIDContacts,
                                true,
                                &cardNo,
                                &dbID);

err = SysUIAppSwitch(   cardNo,
                        dbID,
                        sysAppLaunchCmdNotify,
                        notifyParamP);
```

CHAPTER 6

SMS

This chapter describes the SMS library usage model.

SMS stands for Short Message Service. This service allows short text messages to be sent and received by your mobile phone.

NBS stands for Narrow Band Socket. NBS is a special kind of SMS message. If a SMS message contains `//SCK <code>` it is treated as an NBS message.

What is the difference between SMS and NBS?

NBS on palmOne devices are treated in a silent manner. The message is invisible to the user. An alert is shown every time a SMS message is received. No alert is shown to the user when an NBS message is received.

NBS messages can be silently deleted.

SMS library

Available on:

- Treo™ 600 and Treo™ 650 smartphones

The SMS Messaging application is the main interface to the SMS Database and SMS features on a device. In addition to the SMS application, you can create your own applications to receive, send, and manage SMS messages.

SMS is a useful way to wake up a device remotely and trigger specific actions defined by your application. For example, an SMS could trigger an E-mail application to retrieve new e-mail from a corporate server.

Technically, the SMS library is part of the Telephony library, but it is logically a separate unit. The SMS library relies on functionality from the Telephony library and uses similar methods.

This chapter describes some of the key features of the SMS library. Among these features are:

- Sending messages
- Receiving messages
- Encoding

- Segmenting and reassembling messages
- Storing messages in a database

What is SMS?

The point-to-point SMS provides a means of sending short messages to and from a GSM phone. SMS is implemented using a service center that acts as a store and forwarding center for short messages. Note that certain CDMA operators are supporting or will be supporting SMS. The SMS library architecture described here generally applies to CDMA.

Two different point-to-point services are defined in the SMS specification: *mobile originated* and *mobile terminated*. Mobile originated messages are transported from a phone to a service center. These messages may be destined for other mobile users or an e-mail gateway. Mobile terminated messages are transported from the service center to a phone. These messages may have originated from another phone or from a variety of other sources such as an e-mail application or a web site.

A single message sent on the SMS network is limited to 160 characters. Longer messages must be segmented.

Why use the SMS library?

The SMS library enables the sending and receiving of short messages from a smartphone to another SMS-enabled recipient.

On the Treo 650 smartphone and the Treo 600 smartphone, these messages can be sent much more quickly than with a PPP connection. This is because SMS does not require an initial connection to an ISP. Connecting to an ISP can take up to 10 to 15 seconds, including modem negotiation time and user authentication.

NOTE When sending an SMS message, the current SMS library architecture has a three- to four- second delay before finishing the sending API call. This is due to radio architecture latency.

Cost is another consideration for using SMS. A typical U.S. service provider charges 15 cents a minute for data service, but only 10 cents for a short message.

Additionally, SMS messages can be sent directly to a device. They do not require a client application to dial in to the network to check for new e-mail; instead, the message is sent directly to the client application. The client can even receive an SMS message while a voice or data call is in progress.

Understanding the SMS library

The SMS library takes care of the low-level details of communicating with a device. All incoming messages are indicated to a registered application using launch codes. A successful or unsuccessful transmission of a message is also indicated using launch codes.

The SMS library handles the following functions for SMS messages:

- Sending
- Receiving
- Encoding

  This refers only to character encoding. Currently, only the GSM alphabet is supported. The GSM alphabet is different from the Palm OS® alphabet. Additionally, encryption and compression of messages are not supported in the current version of the SMS library.

NOTE The current version of the SMS library handles text messages for both the GSM and CDMA versions of devices. All messages handled by the library are assumed to be text and are handled appropriately.

The GSM library can also handle binary messages, but the CDMA library cannot. If a client application wants to send binary data on CDMA, it must encode and decode the data properly.

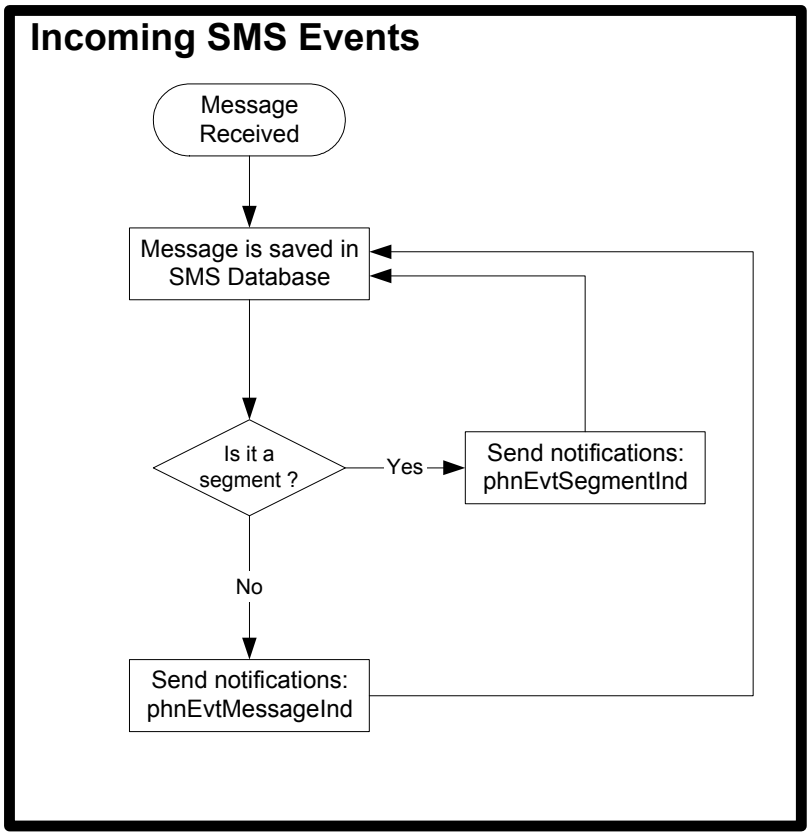
Also, the CDMA version supports only 7-bit ASCII characters, 160 characters maximum, no segmentation.

Incoming messages and message events

An application must register itself with the Telephony library in order to receive incoming message events. You register for the SMS service using `PhnLibRegister` where `services = phnServiceSMS`:

```
PhnLibRegister (libRef, appFileCreator, phnServiceSMS);
```

The following figure shows the workflow of incoming SMS messages.



Each incoming message is indicated to an application by sending a `phnEvtMessageInd` event unless it is a segment of a message. In that case, the library sends a `phnEvtSegmentInd` event. The library sends a `phnEvtMessageInd` only after all of a multisegmented messages's parts have been received.

If a new message is added to the database, the application is sent the `phnEvtSegmentInd` event to trigger an update of the message list. The new message is saved in the SMS database and stays there until it's deleted unless the message is a type NBS (narrow band socket) message. In that case, the message is deleted after the `phnEvtMessageInd` notification is acknowledged by one of the registered applications.

Outgoing messages

A message is sent by creating a new outgoing message using `PhnLibNewMessage` and filling in the desired recipient and message text. More than one recipient may be specified if the message is being sent to a group.

The message text may be longer than 160 characters. The library segments and reassembles such messages automatically so that the segmentation is transparent to the user of the SMS library.

Handling the GSM alphabet and Palm OS alphabet

A special alphabet is used to encode SMS messages. The text of all incoming and outgoing messages stored in the message database is encoded using standard Palm OS encoding.

When a message is received, its encoding is changed from the GSM alphabet to the Palm OS alphabet. Any missing characters are replaced by substitution strings. A message is encoded in the GSM alphabet when being sent. Optionally, substitution strings may be converted to their character equivalents. The following table shows the character with its corresponding substitution string.

GSM	Palm OS	GSM	Palm OS
Δ	\Delta	Π	\Pi
Φ	\Phi	Ψ	\Psi
Γ	\Gamma	Σ	\Sigma
Λ	\Lambda	Θ	\Theta
Ω	\Omega	Ξ	\Xi

NOTE The client application should not assume that a message contains 160 characters or fewer. Even if the actual message contains fewer than 160 characters, the message's text may be longer than 160 characters because characters that are not available on Palm OS are replaced by substitution strings.

Message segmentation

Most mobile phones allow the user to compose messages of no more than 160 characters; however, in GSM, the Messaging application sets the maximum length of a message to 650 characters. Messages that contain more that 160 characters are segmented.

The SMS library supports three types of segmentation schemes. Two of these methods are text-based while the third is binary. The binary method is preferred, because it allows for the reassembly of messages even if they arrive out of order.

Binary segmentation

The binary segmentation scheme works by adding a UDH (user data header) to each segment of a segmented message. The UDH contains a reference number to identify the message, the segment's index, and the total number of segments. Because the UDH takes 6 bytes, the number of characters in a message is reduced to 154.

Using the UDH, it is possible to reassemble messages from their segments even if the segments arrive out of order. The reassembly of incoming segmented messages is transparent to the client application. The client application may retrieve a

message's text even if all the segments have not been received. Use `PhnLibGetText` to retrieve the message.

When the first segment of a segmented message is received, the client receives a `phnEvtSegmentInd` event after the segment has been stored in the database. When all of the segments have been received, a `phnEvtMessageInd` event is sent.

Automatic reassembly of messages works if all parts of the message are received within six hours after the first segment is received. After six hours, segmentation information is deleted by the library.

NOTE The automatic reassembly of messages is a GSM feature. The CDMA version of products do not support automatic reassembly of segmented messages.

Textual segmentation

The SMS library supports two textual segmentation schemes. One segmentation scheme is used only for sending segmented messages to an e-mail gateway. The other is used for segmenting regular text messages.

The segmentation scheme used to send messages to an e-mail gateway does not allow the reassembly of the message if the messages arrive out of order. This scheme is a recognized GSM standard. It works by inserting "+" signs into the message's text. The length of an "inner" segment is reduced to 158 characters, and the length of the first and last segments is 159 characters. A message with three parts is segmented as follows:

```
First segment+
+Inner segment+
+Last segment
```

NOTE For messages sent to an e-mail gateway, the recipient's address adds to the message's length.

This segmentation scheme is used exclusively to send messages to an e-mail gateway. The SMS library does not use this scheme to send text messages to normal subscribers.

The scheme for segmenting regular messages adds header information to every segment. The header is of the form *i/k*, where *i* is the segment's index and *k* is the total number of segments. The length of the header is not constant and is dependent upon the values of *i* and *k*. A message with three parts is segmented as follows:

```
1/3 First segment
2/3 Second segment
3/3 Last segment
```

The library does not attempt to reassemble messages when they are sent using this segmentation scheme. The application must reassemble the messages as needed. The reassembly is not automatically done by the library because the header does not indicate clearly to which message a segment belongs.

Message database

All incoming and outgoing SMS messages are stored in a message database on the device except for NBS-type messages. NBS message content is stored in the `paramBlock` when the application receives the notification.

An outgoing message stored in the database may be sent using the SMS library. Incoming messages received are also stored in this database. The SMS library handles only messages stored in this database. If you want an application to store the messages in a separate database, you must have the application copy the messages from the SMS database.

This section describes the fields for a single SMS message. The standard Palm OS routines for databases are used to manage these records.

IMPORTANT A message's internal structure is private and not simple. Client applications should modify data in an SMS message only by using the functions provided by the SMS library.

A record in the message database has four separate parts. The first part has a fixed size, and the size of the other three parts is variable. As a result, a complete record has a variable size.

The four separate parts of a message database record are as follows:

- Header information
- Segmentation information
- Address information
- Message text

Header information

The first part of each record is the message header information. It has a fixed size. The fields in the header information section of the record are used to store flags and determine the size of the *size[]* field. Some fields are not used in all messages. For example, *validity* is used only for outgoing messages, and *segments* is used only for incoming messages.

SMS Header Structure

```
typedef struct  {
  UInt32 owner;
  SMSMessageType type;
  SMSMessageStatus status;
  UInt32 date;
  UInt32 flags;
  UInt8 validity;
  UInt8 segments;
  UInt16 size[1];
}SmsHeader;
```

See the API guide for details on each field.

Segmentation information

The second part of a message record contains segmentation information. This part is variable in size and is accessed through the `size` array. Each segment's size is represented with 2 bytes.

Address information

The third part of a message record contains address information. This part is variable in size. The size of this address information is defined in the `size` field. The address information contains the data in a `PhnAddressList` structure.

Message text

The fourth part of a message record is the actual text of the message. This part is variable in size. The size is calculated by taking the size of the complete message and subtracting the sizes of the first three parts. All characters in this part of the record are considered to be Palm OS encoded. For outgoing messages, the characters are converted to the GSM alphabet when the message is sent.

NOTE The conversion of the text is done just before the message is sent. The result of the conversion is not stored with the message. If sending a message fails, the text is converted again when the message is sent again.

Launching SMS from the New SMS screen

On Treo smartphones, to launch the SMS application in the New SMS screen, use the Palm OS system Helper API:

```
HelperNotifyEventType   param;
HelperNotifyExecuteType execute;
SysNotifyParamType      notifyParam;
Err                     err = errNone;

MemSet(&param, sizeof(HelperNotifyEventType), 0);
param.version = kHelperNotifyCurrentVersion;
param.actionCode = kHelperNotifyActionCodeExecute;

execute.helperAppID = 0;  // Setting the helperAppID to 0 means
                          // "use default helper"
execute.serviceClassID = kHelperServiceClassIDSMS;

// If you want the New SMS screen to be prefilled with a phone number
// or e-mail address, then set execute.dataP to the string (Char*)
// representing the phone number or e-mail address.  If you do not
// want the New SMS screen to be prefilled with a number, set
// execute.dataP to NULL

execute.dataP = NULL;
execute.displayedName = NULL;
execute.detailsP = NULL;
```

```
param.data.executeP = &execute;

MemSet (&notifyParam, sizeof(SysNotifyParamType), 0);
notifyParam.broadcaster = <YOUR_APP_CREATOR>;
notifyParam.notifyType = sysNotifyHelperEvent;
notifyParam.notifyDetailsP = &param;
err = SysNotifyBroadcast(&notifyParam);
```

CHAPTER 7

System Extensions

This chapter provides details about the system extension features and APIs available in the palmOne™ SDK.

Transparency API

Available on:

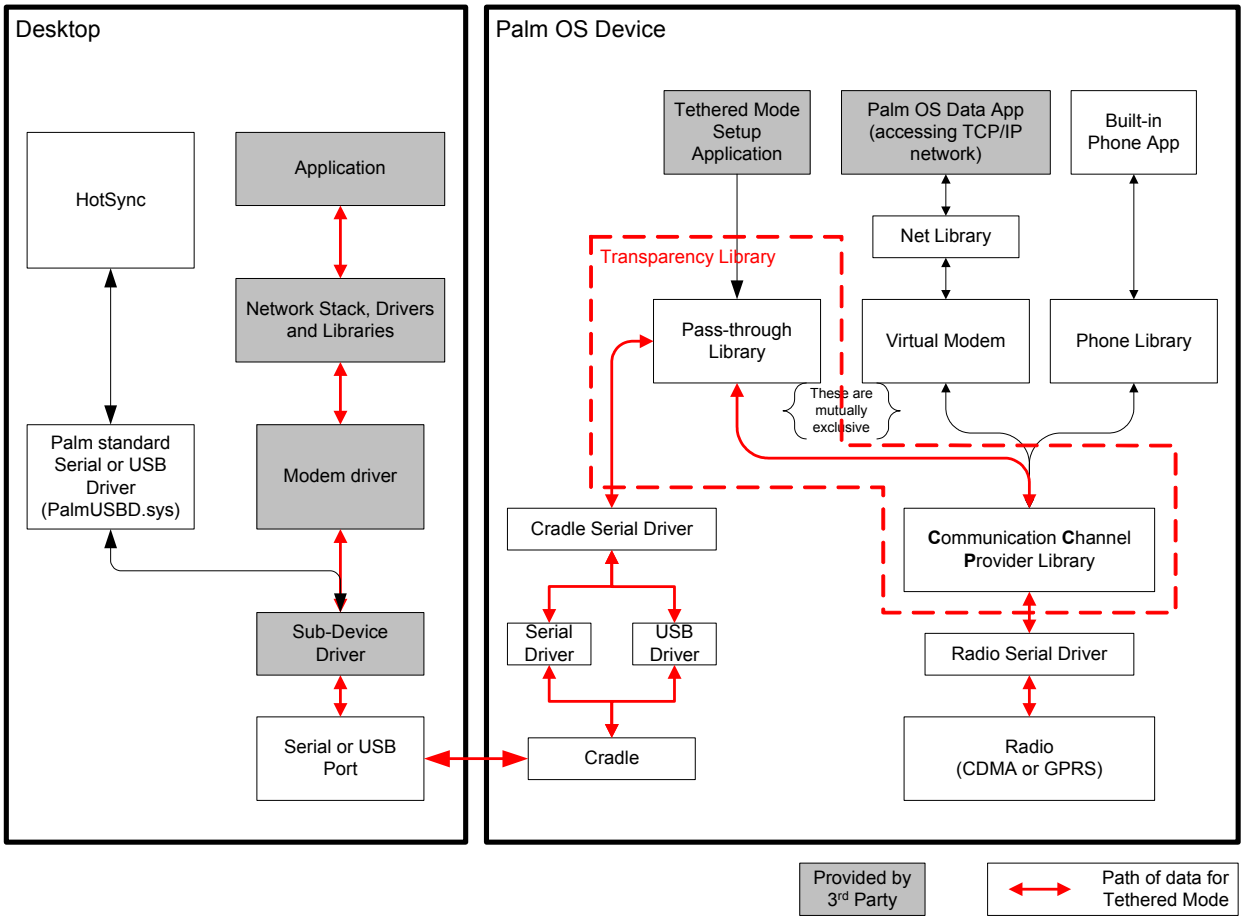
- Treo™ 600 and Treo™ 650 smartphones

The Transparency library is used to connect the radio modem directly to one of the data ports on a Treo™ smartphone.

The library is the preferred method to enable tethered mode in an application because it configures the radio in the right mode to transmit and receive data to the network. The Transparency API is preferable to directly trying to control the radio through its lower level serial driver and through the AT command.

It supports diagnostics or tethered mode. It does not support both simultaneously. See the API guide for more details.

The following figure shows a possible architecture that one could use to connect the Treo smartphone modem to a desktop using tethered mode and the Transparency library.



File Browser API

Available on:

- Tungsten™ T5 handhelds

The File Browser uses the Exchange Manager registry with some new enums defined by PalmSource:

```
#define exgRegEditCreatorID         0xff7b  // creator ID registry
#define exgRegEditExtensionID       0xff7d  // filename
                                              extension registry
#define exgRegEditTypeID            0xff7e  // MIME type registry
```

These constants are defined in `FileBrowserLibCommon.h`. Applications should register using `ExgRegisterDatatype` with ID `exgRegEditExtensionID`. The description of a file type being registered should look like this:

"<Icon 1000,1100>Image"

The first number is the ID of a bitmap family resource for the large icon provided by the application. The second number is the ID of the small icon provided by the application. You can skip the second number if the small icon ID is one greater than that of the large icon:

"<Icon 1000>Image"

The text after the close-angle-quote is the description string that would normally be used. It isn't currently exposed in the File Browser UI, but it could easily be exposed later on so, be sure to include a description.

The size of the large and small icons should match the dimensions of the large and small application icons in the Launcher: 22 x 22 pixels and 15 x 9 pixels. These dimensions are expressed in normal density; the double-density dimensions are 44 x 44 and 30 x 18. The absolute maximum dimensions supported by the File Browser and Favorites applications are 32 x 22 and 16 x 11—64 x 44 and 32 x 22 in double-density. As always, be sure to include a normal-density bitmap as the first element in each bitmap family.

When the File Browser wants to open a file, it does so through the Exchange Manager. Applications typically receive `sysAppLaunchCmdExgReceiveData` to open files. The application won't receive a `sysAppLaunchCmdExgAskUser` sublaunch but will receive a `sysAppLaunchCmdExgReceiveData` sublaunch, just as it would for incoming beams.

Applications should check the name field in the socket to see if it is a "file:" URL. If so, the application should set the `goToCreator` field in the socket to its own creator ID. It can then proceed as if a beam was received, but it shouldn't create a new record. Instead, it should store the parsed data somewhere temporarily and set the `goToParams` struct in the socket to refer to this temporary area.

Alternatively, once you know you're handling a "file:" URL, you can have your application stop using the Exchange Manager. Have your application put the URL into a feature pointer and set the `goToCreator`. Then, when the application receives the `sysAppLaunchCmdGoTo` launch command, you can have the application look for this feature pointer. If the feature pointer is found, you can have the application parse the URL to get a `volRefNum` and path. Then you can have the application proceed as described earlier for `kSysAppLaunchCmdOpenFile`. This is the approach used in the sample application. The code to parse the "file:" URLs is included in the File Browser API library. The sample application available in the SDK includes wrappers around this entry point to make the code easier to use.

The wrapper function is as follows:

```
static Char *ParseFileURL (const Char *url, UInt16 *volRefNumP)
   {
      Char *path = NULL;
      UInt16 refNum;
      Err err;

      err = SysLibFind(kFileBrowserLibName, &refNum);
      ErrFatalDisplayIf(err, "Can't find file browser lib");
      err = FileBrowserLibOpen(refNum);
      ErrFatalDisplayIf(err, "Can't open file browser lib");
```

```
        FileBrowserLibParseFileURL(refNum, url, volRefNumP, &path);
        FileBrowserLibClose(refNum);
        return path;
    }
```

The entry point used in this wrapper function is:

```
Err FileBrowserLibParseFileURL( UInt16 refNum, const Char *urlP,
        UInt16 *volRefNumP, Char **filePathP );
```

It takes a "file:" URL as input and outputs a `volRefNum` and path. It also allocates the path. The caller is responsible for freeing it. If the URL can't be parsed for any reason, it passes back `vfsInvalidVolRef` for the `volRefNum` and `NULL` for the path.

A `fileType` parameter is included in the Open and Save As dialog box APIs. Pass `NULL` for the `fileType` parameter to go to the root directory when switching volumes. The initial folder is the root directory unless you specify a root folder. To specify an initial file name, you must use a full path.

Pass an extension with the initial period or a MIME type to use the registered default directory for the specified file type. The default directory is used when switching volumes and when no initial directory is specified in the initial path.

For example, you could invoke the Save As dialog box with ".jpg" or "image/jpgeg" as the `fileType`, "MonaLisa.jpg" as the initial path, and the volRefNum for the SD card. The initial directory would then be /DCIM, and the filename would be prepopulated with `MonaLisa.jpg`. When the user presses the internal drive button, the/Photos and Videos directory of the internal drive is displayed. When the user presses the internal drive button again, the root directory is displayed.

By registering with the Exchange Manager and handling `sysAppLaunchCmdExgReceiveData`, an application can ensure that its files appear with the correct icon in Files, and that selecting these files opens them in the application. This is the most important aspect of the File Browser API, but there is another side to it as well. Applications can use the File Browser library to display an Open or Save As dialog box. This is most appropriate for applications that attempt to mimic their desktop counterparts. We recommend that these dialog boxes not be used for other applications.

The File Browser library's entry points are declared in `FileBrowserLib68K.h`. To display the Open and Save As dialog boxes, only four of these entry points are required: `FileBrowserLibOpen`, `FileBrowserLibClose`, `FileBrowserLibShowOpenDialog`, and `FileBrowserLibShowSaveAsDialog`. Some constants defined in `FileBrowserLibCommon.h` are also needed.

For example, to display an Open dialog box:

```
UInt16 refNum;
UInt16 volRefNum;
Char *path = MemPtrNew (kFileBrowserLibPathBufferSize);
const UInt16 numExtensions = 1;
const Char *extensions[numExtensions] = {"txt"};
Err err;

ErrFatalDisplayIf (path == NULL, "Can't alloc path");
```

```
// Find the library and call the Open function. There is no need
// to load the library.
err = SysLibFind (kFileBrowserLibName, &refNum);
ErrFatalDisplayIf (err, "Can't find file browser lib");
err = FileBrowserLibOpen (refNum);
ErrFatalDisplayIf (err, "Can't open file browser lib");

// If you want to start with a particular volume, set volRefNum
// to that volume. Otherwise, use vfsInvalidVolRef.
volRefNum = vfsInvalidVolRef;

// If you want to start in a particular directory, set path to
// that directory. Otherwise, use an empty string. You can include
// a filename as well as a directory if you want a file to be
// selected initially. You can specify a filename with no path, but
// only if you specify a fileType.
path[0] = chrNull;

// Display the Open dialog box. Returns whether a file was selected.
if (FileBrowserLibShowOpenDialog (refNum, &volRefNum, path,
    numExtensions , extensions,    // filter to show only these files
    "text/plain",                  // use default folder for this
                                         fileType
    "Select Item",                 // title for the dialog
    kFileBrowserLibFlagNoFolders)) // pick a file, not a folder
{
    // Do something with volRefNum and path.
}

// Clean up. There is no need to remove the library.
MemPtrFree (path);
FileBrowserLibClose (refNum);
```

The File Browser library is preloaded when the device is reset, so you just need to find it and call the `FileBrowserLibOpen` and `FileBrowserLibClose` entry points. You don't need to call `SysLibLoad` or `SysLibRemove`.

The `volRefNum` and path are used both as input and as output. For input, you can do the following:

- Not specify a volume or a path (`vfsInvalidVolRef` and `""`)
- Specify a volume only
- Specify a volume and a directory but no filename
- Specify a volume, a directory, and a filename
- Specify a volume and a filename but no directory
- Specify a filename but no volume or directory

The last two options require that a `fileType` be specified, as well. If the user selects a file or folder and selects OK, the volume and path of the selected file or folder is passed back in the `volRefNum` and `path` parameters and the function returns `true`. Otherwise, the parameters are left as they are and `false` is returned.

If a list of extensions is passed in, only files with one of the specified extensions are listed, along with all the folders. Use `0` for `numExtensions` and `NULL` for `extensions` to disable filtering.

You can specify a MIME type or an extension for the `fileType`. Be sure to include a period before the extension: `.txt`. If a `fileType` is specified, the Open dialog box automatically navigates to the default directory for the specified `fileType` when the user switches volumes. The default directory for a `fileType` can be set using `VFSRegisterDefaultDirectory`. It can be different for different media types—for example, SD/MMC as opposed to the internal drive. If the initial path doesn't include a directory and a `fileType` is specified, the default directory for the specified `fileType` is used. We recommend that a `fileType` be specified whenever possible; it makes it faster for the user to navigate to the appropriate directory. If you use `NULL` for the `fileType`, the user is taken to the root directory when switching volumes.

The title of the Open dialog box depends upon flag settings specified by the client application. If `NULL` is passed in for the title, the Open dialog box is given the default title specified by the flag settings in the client application.

You can use various combinations of the following flags defined in `FileBrowserLibCommon.h`:

- kFileBrowserLibFlagOneVolume - no volume picker
- kFileBrowserLibFlagNoFiles - no files, only folders
- kFileBrowserLibFlagNoFolders - no folders, only files

These flags should be logically combined together. For example: `kFileBrowserLibFlagOneVolume | kFileBrowserLibFlagNoFiles`. Use zero to specify no flags. If you use `kFileBrowserLibFlagOneVolume`, be sure to specify a volume because the user won't be allowed to switch to any other volume.If you use `kFileBrowserLibFlagNoFiles`, the user is only allowed to pick a folder. Files won't be shown at all. If you use `kFileBrowserLibFlagNoFolders`, the user is only allowed to pick a file but can still navigate into folders to find a file. If you don't use either of these flags, the user is allowed to pick a file or a folder.

`FileBrowserLibShowSaveAsDialog` is very similar to `FileBrowserLibShowOpenDialog`. In addition to allowing the user to navigate to any directory, it includes a field where the user can enter a filename. If the specified path includes a filename, it appears in this field. There is an additional parameter for the default extension. If specified, this is appended to the filename entered by the user when it doesn't already have an extension. Use `NULL` for the default extension to use the entered filename exactly as it is entered.

The flags used for the Save As dialog box are as follows:

- `kFileBrowserLibFlagOneVolume` - no volume picker
- `kFileBrowserLibFlagPromptOverwrite` - warn before replacing
- `kFileBrowserLibFlagRequireExtension` - only given extensions
- `kFileBrowserLibFlagNoNewFolder` - no New Folder button

The first flag is used in the same way as it is for the Open dialog box. Use `kFileBrowserLibFlagPromptOverwrite` if you want to warn the user when a filename of an existing file in the selected directory is selected. This is preferable to checking for duplicate filenames afterward because it allows the user to edit the filename or choose a different directory. Use `kFileBrowserLibFlagRequireExtension` if you want to force the user to use a specific extension or one of several extensions. Pass in the list of legal extensions in the `numExtensions` and `extensions` arguments. Use the `kFileBrowserLibFlagNoNewFolder` flag if you want to prevent the user from creating new folders. This hides the New Folder button.

The `FileBrowserLibShowSaveAsDialog` function doesn't actually save anything. It just prompts the user for where to save the file and what to call it. It's up to you to do the following:

- Create the file, truncating the existing file, if any
- Write to the file
- Close the file

Similarly, `FileBrowserLibShowOpenDialog` doesn't open the file or folder selected by the user. It's up to you to do the following:

- Open the file or folder
- For folders, enumerate the contents
- Close the file or folder

Smart Text Engine API

Available on:

- Treo 600 and Treo 650 smartphones

The Smart Text Engine (STE) shared library enables applications to implement rich text display and processing. The STE is designed to allow any application to automatically identify, render, and link web URLs, e-mail addresses, and phone numbers to appropriate applications. For example, selecting a phone number in SMS dials the number, or selecting a URL in an e-mail launches that URL in the web browser. The STE library offers a high level of convenience for end users as it seamlessly links information and communication applications.

The STE library performs three basic functions: parsing, rendering, and displaying. Applications supply the STE with a text stream object and specify the area on the screen to render. This area can include a scroll bar that is used to scroll the contents of a multipage display. The STE passes the text through its three layers to create, display, and activate links.

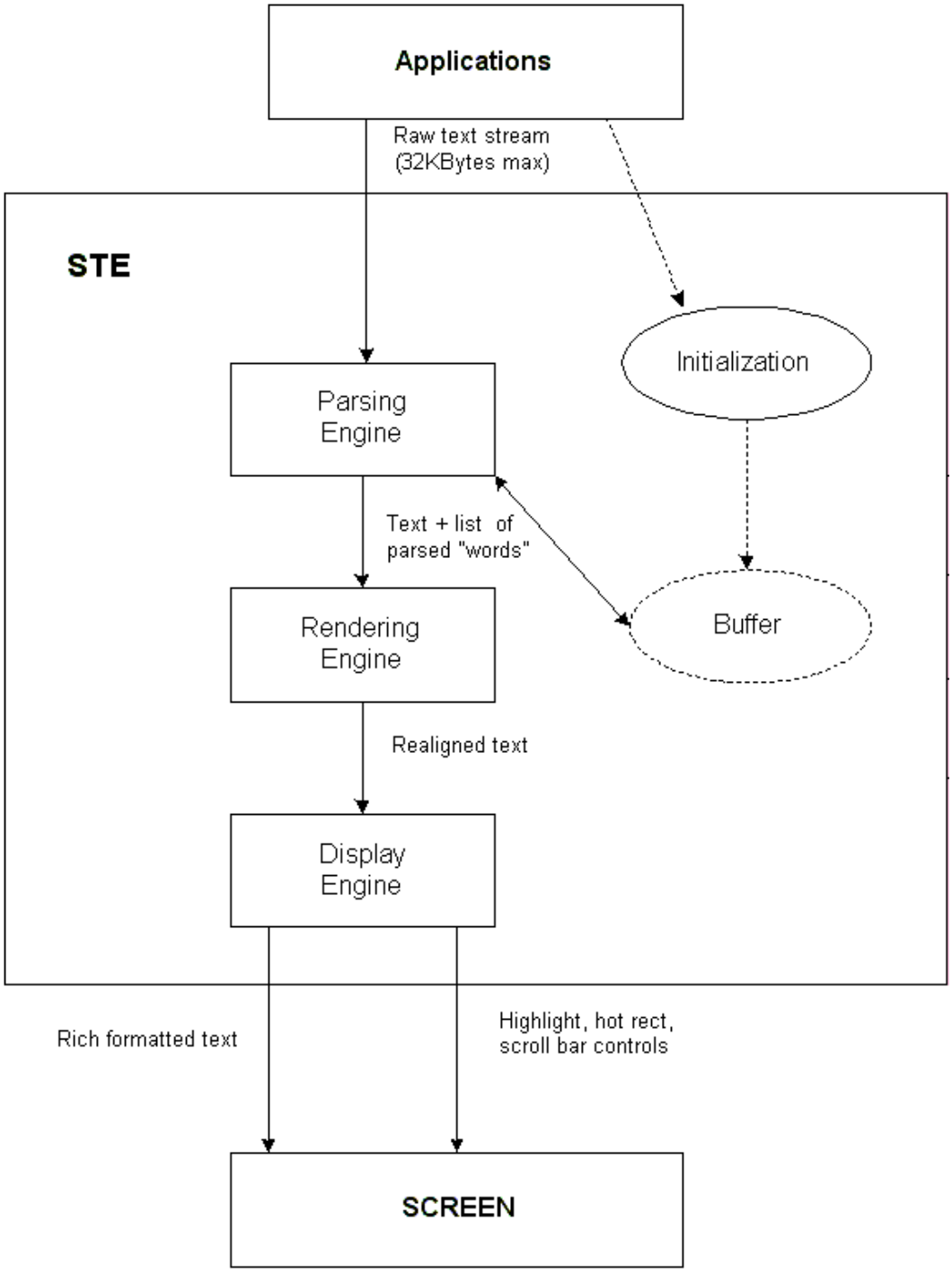
The STE library solves the traditional Palm OS® limitations of the text field: black-and-white display with one font style. Using the STE library, applications can mix bold and standard fonts, mix colors, add graphics, and even add emoticons.

In addition to the information provided here, you can also refer to the STETest.zip sample code file in the SDK for information on how to use this library.

Refer to the API guide for detailed information about each of the STE APIs.

STE Architecture

The STE includes several components—the parsing engine, the rendering engine, the display engine, event handling, and text selection. This modular design allows new components to be added easily. For example, if an HTML parser is needed, it can be added to the STE library without major architectural modifications. The following figure shows the architecture of the STE.



STE Parsing Engine

The parsing engine is responsible for finding the URLs, e-mail addresses, and phone numbers within text. It also detects emoticons and special Smart Text delimiters or tags that affect the text format display. This allows applications to quickly detect whether the text has STE delimiters.

Scanning for URLs, e-mail addresses, and phone numbers requires complex processing in the STE library. A specific algorithm is used to check the validity of characters in URL and e-mail addresses text strings. Checking phone numbers is more complex because it involves appending multiple numbers separated by spaces to compile the final phone number. A basic matching algorithm is used to find emoticons with one special condition: all smileys must have either a ':' (colon) or ';' (semicolon) for the eyes.

Smart Text delimiters are identified by the characters "//STE" followed by additional characters that define the exact properties of the delimiter.

The parsing engine works by separating the string into separate *words*—groups of characters separated by a space character. Each of the words is prescanned to check whether it is a potential URL, e-mail address, phone number, or emoticon type. If the word qualifies as any of these types, it is then further scanned to see if it matches the requirements for that type. Phone number checks require scanning for consecutive groups of words to form the final complete phone number.

The result of the parsing engine is a list of parsed items. This list is then used by the rendering engine to format and display the rich text.

STE Rendering Engine

The rendering engine takes the text input, along with the parsed info list, and determines exactly where the text belongs in the list. If there is no formatting involved, this is very much like displaying text in a text field. When special objects and formatting are added to text, the rendering process becomes more complex.

The basic algorithm goes through the text string and determines whether the next word can fit on the current line in the display. All the STE text string display properties are considered when determining fit. The font can be bold, normal, or colored. There can be emoticons and other bitmaps displayed, as well. The different widths of the text and graphics in different display modes are also considered when calculating fit. The engine keeps track of the actual text that is displayed on each line to process text selection and manipulation.

After all the text has been parsed and rendered, it can be displayed.

STE Display Engine

The display engine takes the data structures created by the rendering engine and displays the correct data at the correct location in the list. The display engine also controls text highlighting and scroll bar positioning.

REM Sleep API

Available on:

- Treo 600 and Treo 650 smartphones

The REM Sleep API allows applications on a smartphone to run while the display is off and the keyboard and digitizer are disabled. The REM Sleep API is on the Treo™ 600 smartphone and the Treo™ 650 smartphone, but not on Tungsten™ or Zire™ handhelds.

REM sleep mode is initiated at the point when the smartphone would otherwise go into deep sleep mode. You should already be familiar with the sleep-deferral and notification mechanisms in the Palm OS before using the REM Sleep API. For more information on notifications, refer to the "Notifications" section, in particular the "Sleep and Wake Notifications," in the *Palm OS Programmer's Companion, vol. I*.

The smartphone may be put to sleep for two reasons—the user presses the power button or another button that functions as a power button, or the smartphone remains idle for the duration set as the auto-off timeout value.

Normal sleep deferral

The process of a smartphone going to sleep begins with a virtual key event. In the case of a user pressing the power button, a `vchrHardPower` event occurs. In the case of the auto-off timeout value being reached, a `vchrAutoOff` event occurs.

When the initiating event is processed by SysHandleEvent, a sleep-request notification is broadcast to all registered recipient applications. If any recipient sets the `deferSleep` member of the notification parameter block, the system does not go to sleep and the smartphone continues to run.

When sleep is deferred, the application or library responsible should enqueue `vchrResumeSleep` after it has finished doing whatever caused it to defer the sleep. The event `vchrResumeSleep` works similarly to `vchrHardPower` and `vchrAutoOff` in that it causes the sleep-request notification to be broadcast. Another application may then set the `deferSleep` member, causing the defer process to continue.

If the sleep request is not deferred, the system enqueues a virtual key event with the `vchrPowerOff` character. When the `vchrPowerOff` event enters SysHandleEvent, a sleep notification is broadcast to inform its recipients that the system is going to sleep, and the system is immediately put to sleep thereafter.

REM sleep mode

REM sleep mode occurs in the time between when the sleep-request notification occurs and when the sleep notification occurs. If the sleep-request notification is not deferred, the display is turned off and then a REM-request notification is broadcast (`hsNotifyRemSleepRequestEvent`). The REM-sleep-request notification has the exact same parameter block as the sleep-request notification. A recipient of the notification that needs REM sleep simply sets the `deferSleep` member in the parameter block. If the `deferSleep` parameter is set in the REM-sleep-request

notification, the system broadcasts a REM notification to inform recipients that REM sleep has been entered.

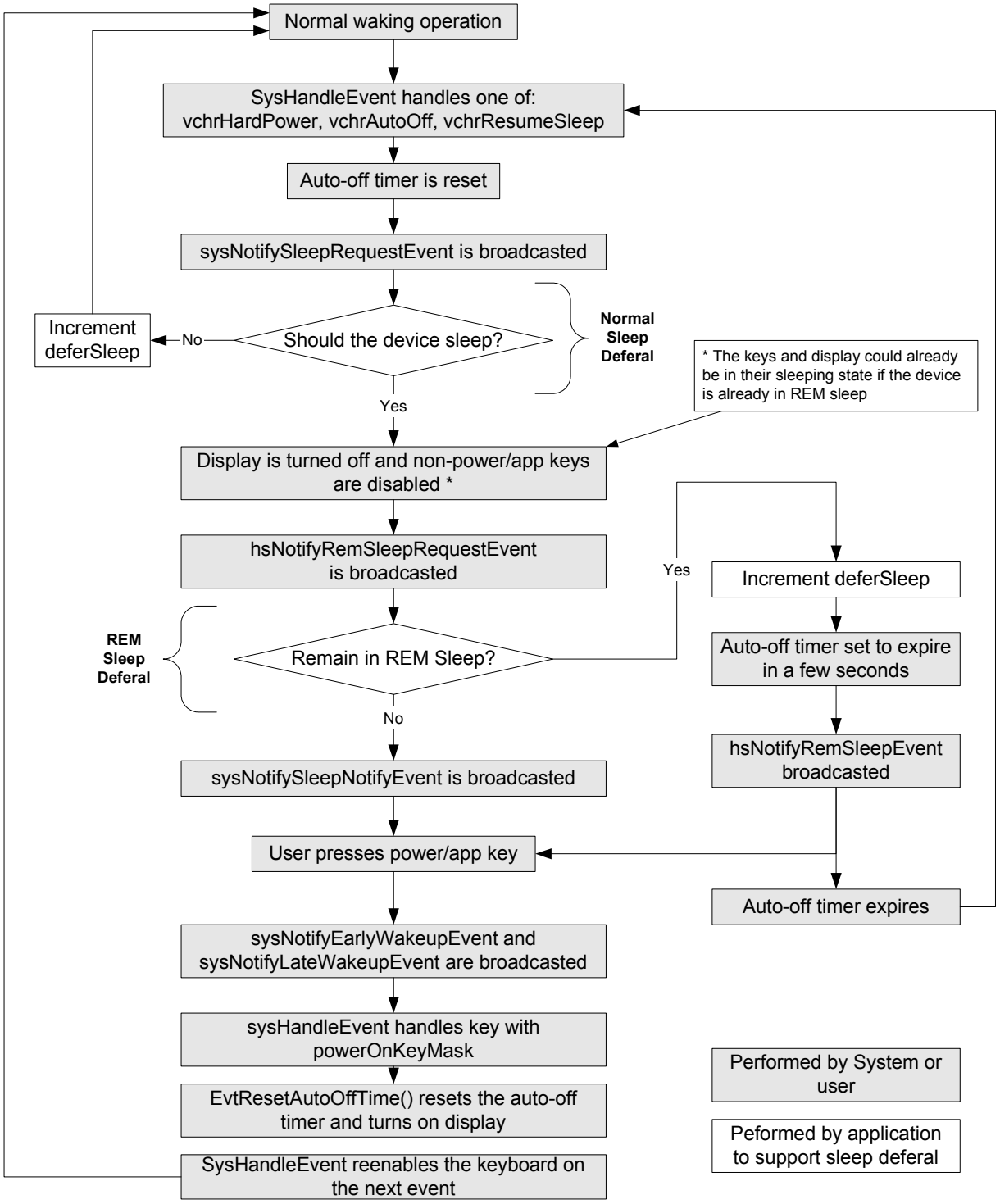
The expectation is that whatever is happening during REM sleep is a transient condition that will in most cases lead to deep sleep shortly thereafter. To accommodate this expectation, the REM-request notification is periodically resent until it is not deferred. The current implementation resets the auto-off timer to expire a few seconds after entering REM sleep so that the whole REM-request notification process repeats periodically. You should not rely on this particular implementation, but instead defer the REM request as often as it comes.

IMPORTANT An application has to register for REM sleep notification (`hsNotifyRemSleepRequestEvent`) and normal sleep (`sysNotifySleeprequestEvent`) notification to support any kind of sleep deferral.

Detecting REM sleep mode

If you need to determine whether the smartphone is already in REM sleep while handling a REM-request notification, you'll have to register for the sleep-request notification and use `HsAttrGet()` to obtain the value of `hsAttrDisplayOn`. By the time the REM request is broadcast, the display is already turned off, and the sleep reason in the REM request simply reflects whatever the sleep reason was from the preceding sleep request.

The following figure shows the sleep mode flowchart of events.



Waking up from REM sleep mode

While the system is running in REM sleep mode, it may be necessary to "wake up" and turn the display on, which you can accomplish by calling `EvtResetAutoOffTimer`.

When displaying any UI that must be seen by the user immediately, you should call `EvtResetAutoOffTimer` to ensure that the display is on. However, it's likely that a user won't even pay attention to their smartphone that are in REM sleep mode. If the user must be notified, consider using the Attention Manager, instead.

In order to maintain the expected user experience, while in REM sleep mode the keyboard and touch screen are set to behave as if the smartphone is sleeping. That is, the touch screen is disabled and only those keys that would normally wake the smartphone are active. If one of the keys that would normally wake the smartphone is pressed, it generates a key event with the `poweredOnKeyMask` modifier bit set. When `EvtGetEvent` sees a key event with this bit set, it turns the display back on by calling `EvtResetAutoOffTimer`.

A version of REM sleep has been present in the Palm OS since version 3.5: After waking up from deep sleep, the auto-off timeout has effectively already expired. If the event queue is emptied without the auto-timer ever being reset, a `vchrAutoOff` event is immediately returned from `EvtGetEvent`, causing `SysHandleEvent` to initiate putting the smartphone back to sleep right away. This behavior has not been changed, but has been accounted for in the REM Sleep API. There will not be a barrage of auto-off events if REM sleep is entered from deep sleep without the display ever being turned on. (This happens during mail sync in certain applications, for example.)

Keyguard API

Available on:

- Treo 600 and Treo 650 smartphones

It prevents the accidental turning on of a smartphone because of an accidental key press. For example, when a smartphone is in a user's pocket or purse, the power button sometimes gets pressed inadvertently. Applications can set or query the state of the Keyguard feature on the smartphone.

When Keyguard is active, the digitizer is locked and `EvtGetEvent` will not return any key events that came from the keyboard. Other key events not generated by the keyboard are returned.

To programmatically enable or disable Keyguard, use `HsAttrSet()` to set the value of `hsAttrKeyboardLocked` to `true`. Do *not* send an `hsChrKeyboardLock` key event to enable Keyguard.

To query whether Keyguard is active, use `HsAttrGet()` to get the value of `hsAttrKeyboardLocked`.

NOTE `hsAttrKeyboardLocked` does not always indicate that the Keyguard dialog box is being displayed. There are states in Keyguard in which key and pen events are filtered and the Keyguard dialog box is not displayed.

To prevent Keyguard from being enabled, block the virtual character `hsChrKeyboardLock` from being handled by `SysHandleEvent`. In an active application, check for `hsChrKeyboardLock` between `EvtGetEvent` and `SysHandleEvent`. In an application running in the background, register for `sysNotifyVirtualCharHandlingEvent` notification and mark the notification handled when a virtual character comes through. Blocking `hsChrKeyboardLock` is all that is required. Even if Keyguard is already enabled, such as through auto-keyguard, Keyguard is disabled when `hsChrKeyboardLock` is blocked.

Setting the value of `hsAttrKeyboardLocked` to `true` does *not* immediately enable Keyguard. Instead, an internal state is set and `hsChrKeyboardLock` is enqueued. Only when `hsChrKeyboardLock` reaches `SysHandleEvent` is Keyguard enabled. At that time, the Keyguard dialog box is displayed, taking control from the active app.

When auto-keyguard is on, Keyguard is enabled after waking up from REM or deep sleep until either the dialog box is displayed or it is determined that Keyguard doesn't need to be enabled because the smartphone wasn't asleep long enough. This prevents extra keystrokes from being lost while a smartphone wakes up. If the value of `hsAttrKeyboardLocked` is set to `false` or `hsChrKeyboardLock` is blocked while the auto-keyguard process is occurring, Keyguard is disabled before the dialog box is displayed.

Option and Shift key APIs

The Option and Shift keys increase the range of input possibilities on a Treo smartphone. For example, a map application, which uses a 5-way Up key press to scroll up, can use an Option+5-way Up key press to zoom in.

There are APIs available to detect if Caps Lock or Option Lock is on. An application can also detect if the Option key or the Shift key has been pressed.

There are also APIs available to set Caps Lock or Option Lock and to programmatically simulate Option or Shift key presses.

Also, if an application includes a Graffiti Shift Indicator(GSI) UI object in the application's resource, then Option and Shift key presses can be visually detected.

MMS helper functions API

Available on:

- Treo 600 and Treo 650 smartphones

The MMS helper function APIs are only available on Treo smartphones. The MMS (Multimedia Messaging Service) Messaging application is the built-in application that provides an interface for other applications to send and receive MMS messages. Application developers can interface with the MMS Messaging application through the Helper API.

The `MMSHelperCommon.h` header file provided with the SDK contains all the public equates.

You can also refer to the MMSReceiver.zip and MMSSender.zip sample code files in the SDK to learn how to use this library.

MMS Usage Model

This section explains how to use the MMS Helper structures with the Palm OS Helper APIs to submit a MMS send request to the MMS Messaging application.

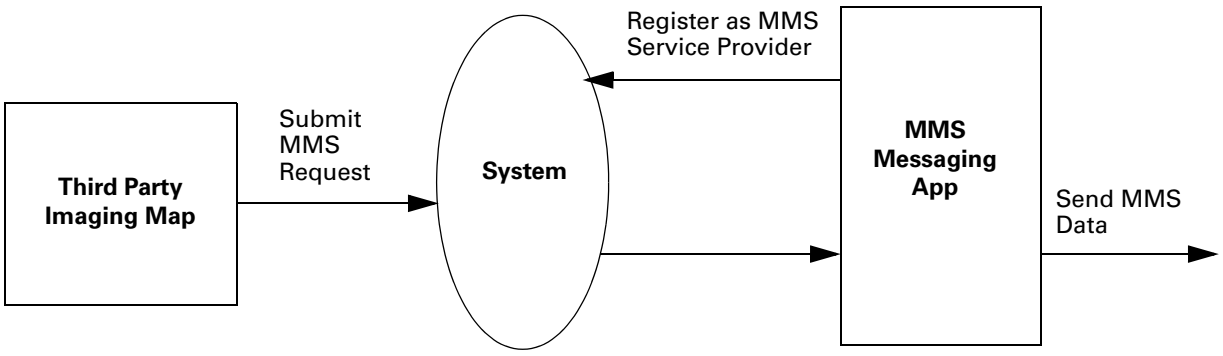
The built-in MMS Messaging application is registered in the system as a Helper service class for the MMS service type. Thus, when the system receives MMS Helper requests submitted by applications, the system broadcasts a notification to the MMS Messaging application to accept and receive the MMS helper data structure from the requesting application.

This gives audio and imaging applications an easy and convenient method to send their picture or audio files using a phone without having to implement the Phone API interface code. The application needs to know only the phone number or e-mail address contact info of the recipient in order to send MMS data to the recipient.

NOTE Currently, only the built-in MMS Messaging application is registered as an MMS Helper service provider. In the future, there may be multiple applications registered as MMS Helper service providers. Please be as specific as possible when designing an application request for an MMS Helper service.

A Helper requesting application can specifically request which Helper service provider to invoke by specifying the exact creator ID of that Helper service provider in the `helperAppID` field of the `HelperNotifyExecuteType` structure. Setting `helperAppID` to `0` indicates all Helper service providers.

The following figure shows the MMS helper usage model.



MMS Sample Code

The following sample code shows how an application submits an MMS send request to an MMS service provider. The implementation first populates the

`HelperServiceMMSDetailsType` and `HelperServiceMMSObjectType` structures with the data, address, and message information. It then attaches the detail structure into a `HelperNotifyExecuteType` so that it can be submitted to the system and be broadcast to all the service providers.

```
{
... ... ...
   case MMSSendButton:
    {
    BitmapPtr bitmapP = NULL;

    HelperServiceMMSDetailsType MMSDetails;
    HelperServiceMMSObjectTypeobject;

    MemSet(&MMSDetails, sizeof(HelperServiceMMSDetailsType),0);
    MemSet(&object,sizeof(HelperServiceMMSObjectType),0);

    bitmapP = MemHandleLock (gLockBitmapH);

    object.pageNum = 1;
    object.tempFileName = NULL;
    object.mimeType = "application/vnd.palm.bmp";
    object.bufferP = bitmapP;
    object.bufferLen = MemPtrSize(bitmapP);

    MMSDetails.object = &object;
    MMSDetails.version = 1;
    MMSDetails.justSend=false;


    MMSDetails.cc = ìa cc addressî;
    MMSDetails.subject = ìsubject textî;
    MMSDetails.message = ìmessage textî;

    err = PrvInvokeHelperService(kHelperServiceClassIDMMS,
          “recipient_address@palmOne.com”, "Helper Sender", &MMSDetails);

    MemHandleUnlock(gLockBitmapH);


... ... ...
```

NVFS API

Available on:

- Treo 650 smartphones
- Tungsten T5 handhelds

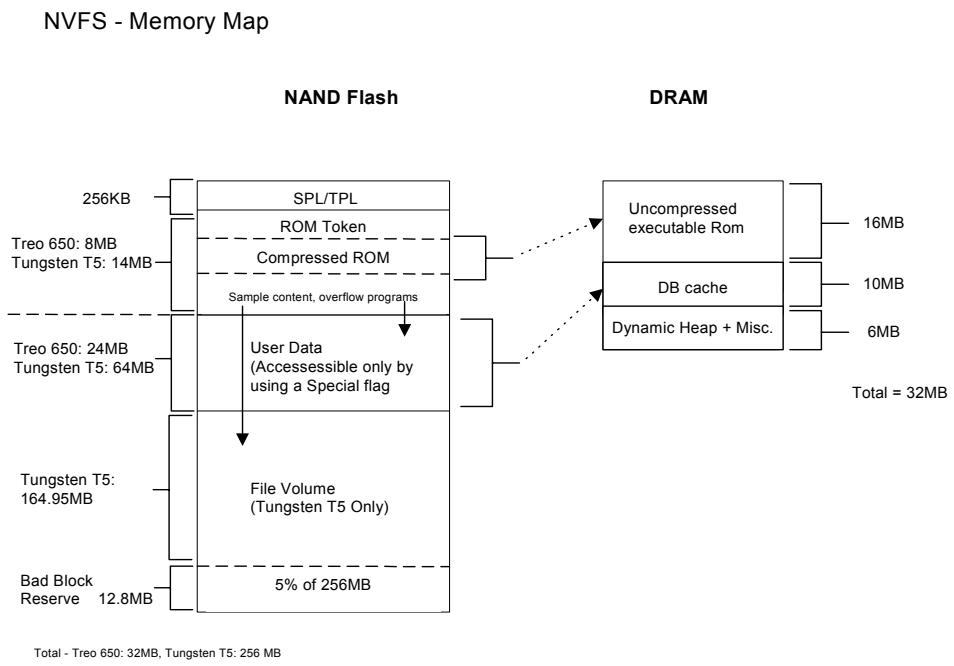
NVFS stands for Non Volatile File System. Storage in palmOne™ devices traditionally included nonvolatile NOR flash where the Palm OS and built-in applications were stored, and a volatile SDRAM (synchronous dynamic random access memory) where the Dynamic and Storage heaps were stored. If the power was removed, all data in the RAM was deleted.

The latest devices, the Treo 650 smartphone and the Tungsten T5 handheld, have NAND flash memory, which is nonvolatile. Future palmOne devices might include NAND memory, as well. This NAND Flash houses the storage heap and other data so that it is not erased even if the power is drained from the device. NAND memory provides sequential access, whereas the NOR memory was random access.

NOTE Throughout this section, the term *ROM* denotes read-only memory, and the term *rom* denotes the OS and palmOne system code, including drivers, the PIM applications, and other programs stored in the NAND flash memory.

The main difference between NOR and NAND flash memory is that NOR flash is completely XIP (execute in place), and all addresses in a NOR flash are mapped using address pins so that each byte can be accessed. In NAND flash memory, data is accessed in blocks, and only a small section. The section that houses the boot code is XIP.

What this means is that the whole rom cannot remain in NAND flash memory and execute as in the past.

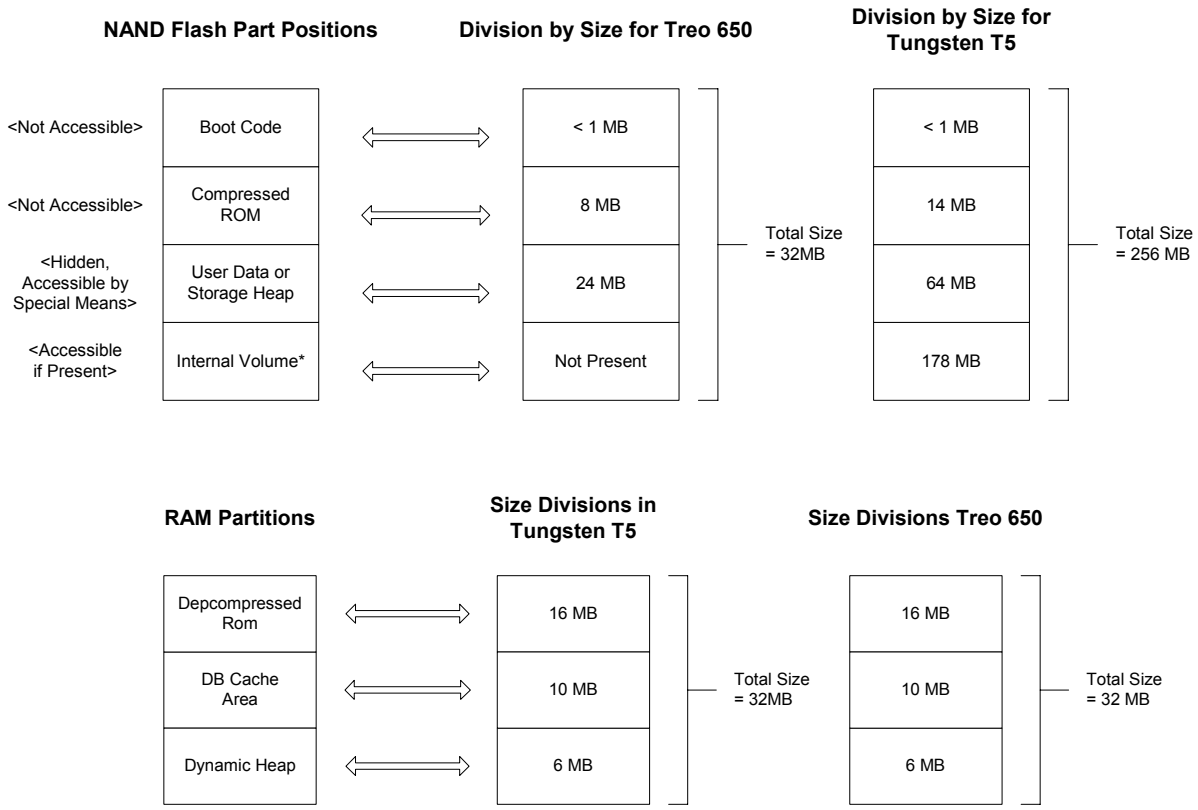


NVFS moves the role of database and application storage into flash and uses a portion of RAM as a cache, called the DBCache, which acts like a temporary storage heap. When an application is launched, its PRC is copied from flash as required into DBCache and is executed from there. In addition, when a database is opened by an application, it is copied from flash into DBCache and is accessed directly from the DBCache. All application access to its databases go directly to the DBCache only; it does not know that there is flash storage behind it. Because the

cache is write-back, writes to the DBCache do not immediately propagate back to flash. Instead the OS writes back those changes either when the database is closed, or when the application manually instructs NVFS to write back the database by calling a new NVFS API. At these times, the database is scanned for "dirty" records, and those records are written back to flash.

When the system goes to sleep or wakes up, it makes sure that database changes are committed to flash memory. The system commits changes to NAND flash when it receives a sleep notification. It does this at normal priority. Applications that make database changes in response to sleep notifications should do so at a higher priority so the system can commit the changes.

The following diagrams show how the NAND Flash and RAM are partitioned. All memory sizes shown in the figures are approximate values.



Differences between NOR and NAND flash memory

In old devices, the NOR flash or masked ROM housed the rom while RAM housed the dynamic and storage heaps. In the new devices, the RAM contains the dynamic heap and a DBCache area that acts as a temporary storage heap. The actual databases are stored in the NAND flash memory as files and are brought into the DBCache area when opened.

The rom is compressed and stored in the NAND Flash, and during boot time the rom is decompressed and brought into RAM storage. Then the Palm OS starts

executing OS code from the RAM. The portion of RAM that has the decompressed rom is read only so that users cannot corrupt OS code (see the preceding figure). The RAM also contains the dynamic heap as in the past. The compressed rom in NAND flash is not visible to or accessible to users or developers.

Performing a soft reset wipes the RAM clean and retrieves a fresh load of the rom from the NAND flash. A hard reset additionally erases the storage heap that is in the NAND flash.

The NAND flash is formatted into multiple partitions. The compressed rom is stored in one partition, the user store in a second partition, and an optional third partition can be present as shown in the preceding diagram. The Palm OS sees these partitions as nonremovable volumes accessible using `VFSMgr` calls. Data in the flash is stored as FAT (file allocation table) files. The volume that houses the storage heap is hidden in all devices and can be accessed only by using a special private flag while enumerating the volumes. This volume is known as the *private volume*. A partition called INTERNAL may also be present on some devices, such as Tungsten T5 handhelds, which you can access and use similar to the way you would use an SD slot. A `VFSVolumeEnumerate()` function yields the two volumes, one for the internal slot and one for the external slot in these devices.

Because the DBCache size is limited to approximately 10MB, resource databases are currently limited to this size. For large record databases, the OS intelligently purges or flushes data from the cache to free space for new records: If the DBCache becomes full, the OS purges records that are not locked in memory. If the records are modified, and then they are committed back to flash memory. The purging of data takes place in the background and is seamless to the user. You should be aware that there are some performance issues associated with this method. There may be a noticeable time difference in performance compared to earlier devices when a large amount of data is flushed back from the cache into the NAND flash memory.

Programming on devices that have NVFS

The changes in how NAND flash works, as opposed to NOR flash, require some changes in how you program for the new devices and future devices. Although all applications should run basically without much modification, there may be some cases that require special handling, such as those that deal with very large databases (over 5MB). Some of these cases are discussed in this section.

Checking for NVFS

To check if a device includes NVFS, use the following command:

```
FtrGet (sysFtrCreator, sysFtrNumDmAutoBackup, &returnVal);
```

If the `returnVal` is `1`, NVFS is present on the device. `sysFtrNumDmAutoBackup` is defined in the `PmPalmOSNVFS.h` header file.

Database issues

One of the issues with the DBCache area is that if a device loses power when a user is modifying a database, all modifications are lost, because database changes are

committed back into the NAND flash only when the database is closed. This also occurs when users remove the battery from Treo 650 smartphones.

NOTE There is no way to determine programmatically that the battery is about to be removed.

To overcome this problem, you might want your applications to call the `DmSyncDatbase()` API. This API ensures that database changes are committed to NAND flash. (Note that the `DmSyncDatbase()` API has nothing to do with the HotSync® feature.) `DmSyncDatabase()` takes a reference to the open database that needs to be synchronized. To use `DmSyncDatabase()`, you need the new `PmPalmOSNVFS.h` header file from the palmOne SDK.

There may be a case in which there is space in DBCache while the storage heap is full. Database creation and modification may work in this case, but `DMCloseDatabase()` may fail, because the database data cannot be committed to the storage heap.

To check the free bytes or size of DBCache, use `MemFreeBytes()` and set the high bit of the heap id as follows:

```
MemHeapFreeBytes(STORAGE_HEAP_ID | dbCacheFlag, &free, &max);
```

The call `MemHeapFreeBytes(STORAGE_HEAP_ID, &free, &max);` returns the free bytes in the storage in the NAND flash part of the memory.

To discover the size of the user store or DBCache area, you can use the Palm OS API `MemHeapSize` with the same values for the `STORAGE_HEAP_ID` described above.

Applications that work with resource databases larger than the size of the DBCache area can directly access the internal file volume as described in the next section, or split the data in databases to span multiple databases.

Accessing the internal file volumes in NVFS

The volume housing the storage heap can be accessed using the `VFSVolumeEnumerate()` function as follows:

```
#define vfsIteratorStart              0L
#define vfsIteratorStop               0xffffffffL

UInt32 volIterator = vfsIteratorStart | vfsIncludePrivateVolumes; //
0x80000000L is the flag passed in to include the private volume...

while (volIterator != vfsIteratorStop)
{
   if ((err = VFSVolumeEnumerate(&otherVolRefNum, &volIterator)) ==
errNone) {
             err = VFSVolumeInfo(otherVolRefNum, &volInfo);
             if (err)
                  goto Done;
             if (volInfo.attributes & vfsVolumeAttrHidden)
             {
             // This is an internal file volume. Perform actions now...
             }
```

```
          }
}
```

After you retrieve the volume reference number, the volume can be accessed by `VFSMgr` as with any other volume. If the `vfsIncludePrivateVolumes` flag is omitted, the private volume is not enumerated. So, for example, in a Treo 650 smartphone, the preceding code enumerates a private volume and the SD slot's volume. `vfsIncludePrivateVolumes` is defined in the `PmPalmOSNVFS.h` header file.

Applications that must access the private volume directly to store files in the FAT file system can do so if they are having problems working with large databases because of the DBCache size. You should use the private volume sparingly, however, because overuse cuts into the user storage space.

Similarly, you can use the Expansion Manager to see the private volume as a separate internal slot if it is used with the following flag:

```
#define expIteratorStart              0L
#define expIteratorStop               0xffffffffL

UInt32 slotIterator = expIteratorStart | expIncludePrivateSlots; //
0x80000000L is
flag passed in to include private slot;

while (slotIterator != expIteratorStop)
{
          if ((err = ExpSlotEnumerate(&slotRefNum, &slotIterator)) ==
          errNone)
{
             // Perform actions now...
          }
}
```

Other than the private volume, some devices may include a separate internal file volume that can be used for file storage. Tungsten T5 handhelds, for example, have an internal file volume that is available as an internal drive. The following code enumerates two volumes: the internal file volume and the SD slot volume. You can use `vfsVolumeAttrNonRemovable` to check whether a volume is nonremovable, thus indicating that it is an internal volume. You need the new `PmPalmOSNVFS.h` header file from the palmOne SDK to do so. The PalmSource SDK may not have the flag defined. You can use the `vfsIncludePrivateVolumes` flag to access the third private volume in a Tungsten T5 handheld.

```
UInt32 volIterator = vfsIteratorStart;

while (volIterator != vfsIteratorStop)
{
   if ((err = VFSVolumeEnumerate(&otherVolRefNum, &volIterator)) ==
errNone) {
          err = VFSVolumeInfo(otherVolRefNum, &volInfo);
          if (err)
            goto Done;
          if (volInfo.attributes & vfsVolumeAttrNonRemovable)
          {
             // This is the internal file volume. Perform actions now...
          }
```

```
        }
}
```

Similarly, you can use the Expansion Manager to access the slots by using the capability flags `expCapabilityNonRemovable` and `expCapabilityHidden` from the `ExpCardInfoType` structure to check the slot details. You may need the new `PmPalmOSNVFS.h` header file from the palmOne SDK for these flag definitions.

Feature pointer issues

Because feature pointers are allocated in the DBCache, it is possible to run out of space if you open a very large database. Even if the user store in the NAND flash has free space, a feature pointer call such as `FtrPtrResize` can fail if the DBCache is full. Query the free bytes in DBCache and take the appropriate actions as mentioned in the preceding sections if you allocate large feature pointers or if you are using feature pointers while working on large databases.

5-Way Navigation and Keyboard API

Available on:

- Treo 600 and Treo 650 smartphones
- Tungsten T5, Tugnsten™ T3, Tungsten™ E, and Tungsten™ C handhelds
- Zire™ 31 and Zire™ 72 handhelds

This section describes the software associated with the 5-way navigation button.

In addition to the information provided here, you can refer to the sample code in the SDK for information on how to use this module.

5-way navigation terminology

The terms *object focus mode*, *application focus mode*, *tab order*, and *action buttons* and *interaction mode* appear throughout this section. A short description for each of these terms follows:

- *Object focus mode* refers to the state when a form's focus is enabled and the 5-way keys are used for navigation.
- *Application focus mode* refers to the state when a form's focus is not enabled and Up and Down are used as page-up and page-down keys.
- *Tab order* refers to the horizontal ordering of the objects or, in other words, the order in which objects receive the focus when Right is pressed repeatedly.
- *Action buttons* refers to the command buttons that are lined up along the bottom of a form.
- *Interaction mode* refers to the state when the 5-way keys interact with the object rather than move the focus. For example, a field object is in interaction mode when the 5-way keys move its insertion point and a list object is in interaction mode when the 5-way keys move a highlight through the lists items.

Overview of 5-way navigation

The navigation model is two-dimensional. Left and Right move the focus horizontally, while Up and Down move the focus vertically.

Depending on the object type of the object that has the focus, Center either simulates a tap on the focused object or toggles the interaction mode of the focused object.

To support this functionality, the system was expanded to generate and handle additional navigation events. These include 5-way key events and the focus change events. The events are handled when an application calls `FrmDispatchEvent()`. During such a call, the events are handled in the following manner. (Note that steps 2 and 3 are part of `FrmHandleEvent()`.):

1. The current form's custom event handler receives the event.

   This is when the application can override default navigation behavior. If the handler returns `true`, no more event processing occurs and steps 2 and 3 are never reached.

2. If the event is a key event, the focused object's type is obtained. If the event is a focus change event, the type of the object specified in the event is obtained.

   The handler specific to the object type obtained is then called on the event. (For example, `CtlHandleEvent()` is called if the object is a control.) The various object type handlers have been expanded to handle navigation events that are associated with type-specific navigation behavior. If the handler returns `true`, no more event processing occurs and step 3 is never reached.

3. A generic focus handler is called on the event. This handler is primarily responsible for moving the focus in the form.

Navigation events

The Treo 600 smartphone, the Treo 650 smartphone, and the Tungsten T5 handheld not only generate `keyDown` events for keys, but they also generate `keyHold` and `keyUp` events for keys. A `keyHold` event is generated when a key is held for 1 second and a `keyUp` event is generated when a key is released. `keyHold` and `keyUp` events have the same fields as `keyDown` events—character field, modifiers field, and key code field.

When object focus mode is on, the 5-way button generates `keyDown`, `keyHold`, and `keyUp` events with the following character values: `vchrRockerUp`, `vchrRockerDown`, `vchrRockerLeft`, `vchrRockerRight`, and `vchrRockerCenter`.

When object focus mode is off, the 5-way button generates `keyDown`, `keyHold`, and `keyUp` events with the following character values: `vchrPageUp`, `vchrPageDown`, `vchrRockerLeft`, `vchrRockerRight`, and `vchrRockerCenter`.

The page keys are enqueued when object focus mode is off so that forms that are not 5-way navigation–aware do not lose their paging functionality.

Focus change events are generated as the navigation focus moves in a form. A `frmObjectFocusTake` event is sent when an object should take the focus. The system's internal focus structures are updated when the event is *handled*, not when it is sent. A `frmObjectFocusLost` event is sent after an object has lost the focus. The system's internal focus structures have already been updated when this event is sent. Therefore, it simply initiates the redrawing of the object that lost the focus.

Option and Shift modifiers

Treo 600 and Treo 650 smartphones have Option and Shift keys. If the Option key is held down while the 5-way button is pressed, the `optionKeyMask` in the 5-way button's key events is set. Similarly, if the Shift key is held down while the 5-way button is pressed, the `shiftKeyMask` in the 5-way button's key events is set.

These masks allow applications to assign secondary features and functionality to the 5-way button on the Treo 600 and Treo 650 smartphones.

Including objects as skipped objects

Editable fields, tables with fields, pop-up triggers, and selector triggers automatically get navigation focus when the user taps them. If the object is not in the tab order, then the system simply moves the focus to the first object in the tab order when the user presses a directional button after tapping the object. Such a movement of the focus may not make sense to the user.

To ensure that focus movement is always logical to the user, such objects should be included in the tab order but marked as skipped by setting the `kFrmNavObjectFlagsSkip` flag. This flag can be set through `FrmSetNavEntry()`, through `FrmSetNavOrder()`, or in a navigation resource. The object is skipped when the user moves the focus with the 5-way buttons, but the system knows how to move the focus from the object after the user taps it.

Default navigation

If a form does not have a navigation resource, the system determines the navigation order for the form dynamically. It determines whether the form is initially in object focus mode or application focus mode, which UI objects can receive the focus, the tab order, the vertical order, where the focus begins, and whether the focus cycles.

IMPORTANT To truly support 5-way navigation, we highly recommend that an application have `fnav` resources in its resource file rather than rely on the default navigation order.

Initial focus mode

Forms are separated into modal and nonmodal forms. The window of a modal form has the modal flag set. You can check this flag by calling `WinModal()` on the form's window. All other forms are considered nonmodal.

Initially, modal forms are placed in object focus mode, and nonmodal forms are placed in application focus mode. You can programmatically change the mode of a form with `FrmSetNavState()`.

UI objects included in the navigation order

All UI objects in a form are included in the navigation order. However, during form initialization, only the following objects are not marked as skipped:

- Usable, enabled controls (controls include command buttons, push buttons, check boxes, pop-up triggers, selector triggers, repeating buttons, sliders, and feedback sliders)
- Usable lists
- Usable, editable fields

NOTE An application can clear the skipped flag for any object in the form by calling `FrmSetNavOrder()` or `FrmSetNavEntry()`.

If an object is not initially included in the default order, it is not included later if it becomes a valid focus object. For example, if a control that was initially disabled is enabled, it is not included in the default order when it is enabled.

However, objects that were initially included in the default order that later become invalid focus objects are skipped. For example, if a field was initially editable and was later made noneditable, it is skipped.

Tab order

The tab order is determined by taking the UI objects that can receive the focus and sorting them by their left coordinate and then their top coordinate. A stable sort is used (insertion sort) so that after sorting:

- Objects are sorted by their top position.
- Objects with the same top position are sorted by their left position.

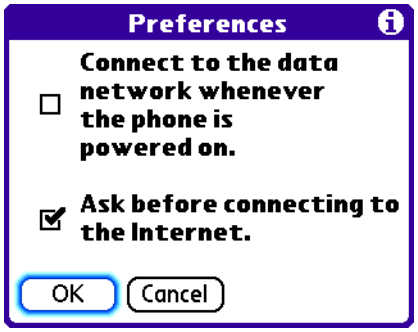
Vertical order

The objects in the vertical order are a subset of those in the tab order. The vertical order is determined by iterating from the first object in the tab order to the last and including only the following objects:

- The first object
- Any subsequent object that is completely below the previously included object

This algorithm results in only the leftmost objects of the form being included in the vertical order.

Because the vertical order may not include all the objects in the tab order, some objects that can receive the navigation focus cannot be navigated to by only using the Up and Down keys. In other words, navigating to objects that are in the tab order but not in the vertical order will require some use of the Left or Right key. For example, consider the following dialog box:



The check box and the OK button are the leftmost objects in their row and therefore are the objects in the vertical order. To get to the Cancel button from the top check box, the user would have to press Down and then Right. There is no way to get to the Cancel button by using only the Up and Down keys.

If an object that was initially usable and was part of the vertical order is later made nonusable, and an application has not made any changes to the order through the navigation API functions, the vertical order is recalculated using the algorithm previously described. The vertical order is not recalculated when an application has made changes to the order, to ensure that the application's changes are not overridden.

Initial focus

If there are any action buttons, the focus is initially given to the leftmost one. If there are no action buttons, the focus is given to the first object in the tab order.

The first action button is identified using the following rules:

- The usable, enabled objects with the greatest top coordinate values are identified. In other words, the objects in the form's bottom row are identified.
- The first action button is the leftmost command button among those objects. If there are no command buttons among these objects, the focus is given to the first object in the tab order.

Cycling

On the Treo 600 smartphone, the Treo 650 smartphone, and the Tungsten T5 handheld, the focus never cycles horizontally. When the focus is on the first object in the tab order, pressing Left does not move the focus to the last object, and when the focus is on the last object, pressing Right does not move the focus to the first object.

Although the focus never cycles vertically on Treo 600 smartphones, the focus does cycle vertically in modal dialog boxes on Treo 650 smartphones and Tungsten T5 handhelds. On Treo 600 smartphones and in nonmodal dialog boxes on Treo 650 smartphones and Tungsten T5 handhelds, when the focus is on the top object in the vertical order, pressing Up does not move the focus to the bottom object, and when the focus is on the bottom object, pressing Down does not move the focus to the top object. In modal dialog boxes on Treo 650 smartphones and the Tungsten T5 handheld, pressing Up while the focus is on the top object *does* move the focus to the bottom object, and pressing Down while the focus is on the bottom object *does* move the focus to the top object.

Custom navigation

Applications can customize navigation by providing a navigation resource for a form, by making navigation API calls, and by handling navigation events.

Hex navigation resource

A navigation resource specifies what UI objects in the form are included in the navigation order, what the tab order is, what the vertical order is, whether the focus is initially on or off, where the focus begins, where the focus can move, and what the bottom-left object is. (The bottom-left object information is needed to cycle the focus from the top of the form.)

IMPORTANT As mentioned before, we recommend that an application include `fnav` resources in its resource file rather than rely on default navigation order.

For the system to detect a navigation resource, the navigation resource must be in the same database as its associated form. For the system to detect that a form has a

navigation resource, the navigation resource must be in the same database as the form.

Technically, an application only needs a navigation resource if the behavior is not the correct behavior for a form. However, the creation of navigation resources for all forms that have navigation is recommended because the default navigation order may be different on various platforms and some platforms may not even provide a default navigation order.

The resource is a hex resource of type `formNavRscType` (`fnav`). It is defined as a 68K format (big endian) resource.

The term "hint" is appended to the name of those fields that are used only by some platforms. For example, the `bottomLeftObjectIDHint` is used only by those platforms that have their focus cycle from the top of the form to the bottom. If an application specifies a value for this field, it will run properly on platforms that do cycle and platforms that do not cycle.

The navigation resource has a header section and a list-of-objects section. The format of the header is described in the API guide.

PilRC navigation resource

The PilRC resource compiler supports a navigation resource format that is easier to create than the HEX resource just described. Many of the fields required by the HEX resource are optional in the navigation resource and a navigationmap that allows UI objects to be specified in row-column fashion is supported. This format is supported starting with PilRC 3.0 and is documented in the manual packaged with PilRC. PilRC can be obtained at http://pilrc.sourceforge.net.

Objects that become nonusable

The system skips over nonusable objects when a user moves through the tab order. Objects that reside in the same position in the form and are alternately shown should therefore be placed next to each other in the tab order and have the same above and below objects.

If an object in the vertical order becomes nonusable, and the user navigates up to it, the object above receives the focus instead. The down case works the same way, except that the object below the nonusable object receives the focus. If you want an application to behave differently, use the API calls to explicitly set which object should replace the nonusable object in the vertical order.

Handling navigation events

System navigation behavior is executed when `FrmHandleEvent()` receives a navigation event. Because `FrmDispatchEvent()` calls a form's custom event handler before calling `FrmHandleEvent()`, a form can easily override default navigation behavior by handling navigation events in its custom event handler.

As explained earlier, the system's internal focus structures are updated when a `frmObjectFocusTake` event is handled, not when it is sent. Therefore, an application must explicitly call `FrmSetFocus()` on the event's associated object if it handles a `frmObjectFocusTake` event.

A `frmObjectFocusLost` event is sent after an object has lost the focus. The internal focus structures will have already been updated when this event is sent. Therefore, an application does not have to do anything besides implement its desired custom behavior if it handles a `frmObjectFocusLost` event.

With tables and gadgets, applications must intercept navigation events. These UI object types have minimal default behavior, if any.

For example, when an application has a table that is included in the focus order, the application might perform the following actions in its custom handler:

- `frmObjectFocusTake` event for table:

  Call `FrmSetFocus()` on the table, highlight the first row of the table, and return `true`.

- `frmObjectFocusLost` event for table:

  Unhighlight row in table and return `true`.

- An Up `keyDown` event when table has the focus:

  If the highlight is not on the table's top row, move the highlight up a row, and return `true`.

- A Down `keyDown` event when table has the focus:

  If the highlight is not on the table's bottom row, move the highlight down a row, and return `true`.

In general, it is assumed that most modal forms do not alter the default navigation behavior, while most nonmodal forms do. That is why navigation is automatically enabled only for modal forms. Nonmodal forms are usually the main views of an application and therefore require a lot of custom behavior.

Focus treatment

The functions that draw UI objects are updated to know how to draw the new visual states introduced by navigation. The system uses a blue square ring, a blue rounded ring, or blue bars to indicate that an object has the focus:

`CtlDrawControl()` checks whether the control it is drawing has the navigation focus, and draws a focus ring around the control if it does.

`FldDrawField()` checks whether the field it is drawing has the navigation focus. The function then checks whether the field is in interaction mode. If it is not in interaction mode, the function draws focus bars above and below the field and does not draw the insertion point. If it is in interaction mode, the function draws the field normally and draws the insertion point.

If `LstDrawList()` is drawing an embedded list, it checks whether the list has the navigation focus. The function then checks whether the list is in interaction mode. If it is in interaction mode, the function draws a focus ring around the temporarily selected item. If it is not in interaction mode, the function draws a focus ring around the entire list.

API functions are provided if you want to draw focus rings around objects other than controls, fields, and lists in your application. On the Treo 600 smartphone, the functions are `HsNavDrawFocusRing()`, `HsNavRemoveFocusRing()`, and `HsNavGetFocusRingInfo()`. On the Treo 650 smartphone and the Tungsten T5 handheld, the functions are `FrmNavDrawFocusRing()`, `FrmNavRemoveFocusRing()`, and `FrmNavGetFocusRingInfo()`.

The system ensures that no more than one ring is ever drawn on a form. If a ring is being drawn and there is already a ring on the form, the system removes the ring already displayed on the form before drawing the new ring. A ring drawn with `HsNavDrawFocusRing()` or `FrmNavDrawFocusRing()` should never be directly erased. If you want your application to remove the ring, it should call `HsNavRemoveFocusRing()` or `FrmNavRemoveFocusRing()`.

On Tungsten T5 handhelds, when the Active Input Area is collapsed or expanded, the system automatically removes any focus ring before the area is collapsed or expanded and, after the area has been collapsed or expanded, sends a `frmObjectFocusTake` event with the ID of the object that had the focus ring.

Navigational API and behavioral differences between Treo™ 600 smartphones, Treo 650 smartphones, and Tungsten™ T5 handhelds

This section describes navigational differences between the Treo 600 smartphone, Treo 650 smartphone, and Tungsten T5 handheld.

Palm OS Features

Treo 600 smartphones, Treo 650 smartphones, and Tungsten T5 handhelds all set the `hsFtrIDNavigationSupported` feature. The feature's creator is `hsFtrCreator`. The value of the feature is the version number of the palmOne navigation API. On Treo 600 smartphones the version is 1, and on Treo 650 smartphones and Tungsten T5 handhelds the version is 2.

Treo 650 smartphones and Tungsten T5 handhelds also set the `sysFtrNumFiveWayNavVersion` feature. The feature's creator is `sysFileCSystem`. The value of the feature is the version number of the PalmSource navigation API. On both Treo 650 smartphones and Tungsten T5 handhelds, the version is 1. Version 2 of the palmOne navigation API and version 1 of the PalmSource navigation API are the same.

Treo 600 smartphones with software = 1.12, Treo 650 smartphones, and Tungsten T5 handhelds set the `sysFtrNumUIHardwareFlags` feature. The feature's creator is `sysFileCSystem`. The value of this feature is a bit field that describes what hardware is available on the device. The bit definitions used with the feature's value are as follows:

- sysFtrNumUIHardwareHard5Way:

  The device has a 5-way rocker.

- sysFtrNumUIHardwareHasThumbWheel:

  The device has a thumb wheel.

- sysFtrNumUIHardwareHasThumbWheelBack:

  The device has a thumb wheel with a Back button.

- sysFtrNumUIHardwareHasKbd:

  The device has a dedicated keyboard.

On Treo 600 smartphones with software = 1.12, Treo 650 smartphones, and Tungsten T5 handhelds, the `sysFtrNumUIHardwareHas5Way` is set. On Treo 600 smartphones with software = 1.12 and on Treo 650 smartphones, the `sysFtrNumUIHardwareHasKbd` is also set.

NOTE The feature not being set on the original Treo 600 smartphone software was an oversight. This problem was fixed with software update 1.12 available from the palmOne Customer Support download area.

The Tungsten T5 handheld no longer supports the `navFtrVersion` feature supported on Zire handhelds and earlier Tungsten handhelds.

Functions

Treo 600 smartphones and Treo 650 smartphones support `HsNavDrawFocusRing()`, `HsNavRemoveFocusRing()`, `HsNavGetFocusRingInfo()`, and `HsNavObjectTakeFocus()` calls. Treo 650 smartphones and Tungsten T5 handhelds support `FrmNavDrawFocusRing()`, `FrmNavRemoveFocusRing()`, `FrmNavGetFocusRingInfo()`, and `FrmNavObjectTakeFocus()` calls.

Except for the prefix differences in their names, these functions work exactly the same way. Applications should transition to using the `FrmNav` API calls, because the `HsNav` calls are deprecated and remain only on Treo 650 smartphones for Treo 600 smartphone backward compatibility.

IMPORTANT Do not make `HsNav` calls on Tungsten T5 handhelds. `HsNav` calls made on Tungsten T5 handhelds will fail, most likely with a Sys0505 error, which means that the module that exports the function is not on the handheld.

Because Treo 600 smartphones support only the `HsNav` version of these calls and Tungsten T5 handhelds support only the `FrmNav` version of these calls, applications intended to run on both devices must check their context before making these calls. The suggested method is to check the version number of the `hsFtrIDNavigationSupported` feature and decide whether to make an `HsNav` call or a `FrmNav` call based on the version value. Specifically, `HsNav` calls should be made if the version is `1`, and `FrmNav` calls should be made if the version is `2`. The decision about what call to make must be made at runtime. For example:

```
if (FtrGet (hsFtrCreator, hsFtrIDNavigationSupported, &version) == 0)
  {
    if (version == 1)
      HsNavObjectTakeFocus (formP, objID);
    else // if version >= 2
```

```
    FrmNavObjectTakeFocus (formP, objID);
}
```

Associating custom behavior with the Center button

Treo 600 smartphones, Treo 650 smartphones, and Tungsten T5 handhelds generate the following key events for Center button actions:

Action	Treo 600 smartphone	Treo 650 smartphone and Tungsten T5 handheld
Press	`keyDown` event with `chr=vchrRockerCenter`, `keycode=keyRockerCenter`, and `modifiers=commandKeyMask`.	`keyDown` event with `chr=vchrHardRockerCenter`, `keycode=keyRockerCenter`, and `modifiers=commandKeyMask`.
Continuous Press	`keyDown` event with `chr=vchrRockerCenter`, `keycode=keyRockerCenter`, and `modifiers=autoRepeatKeyMask | commandKeyMask`.	`keyDown` event with `chr=vchrHardRockerCenter`, `keycode=keyRockerCenter`, and `modifiers=autoRepeatKeyMask | commandKeyMask`.
Held for 1 second or longer	`keyHold` event with `chr=vchrRockerCenter`, `keycode=keyRockerCenter`, and `modifiers=commandKeyMask`.	`keyHold` event with `chr=vchrHardRockerCenter`, `keycode=keyRockerCenter`, and `modifiers=commandKeyMask`.
Release	`keyUp` event with `chr=vchrRockerCenter`, `keycode=keyRockerCenter`, and `modifiers=commandKeyMask`.	`keyUp` event with `chr=vchrHardRockerCenter`, `keycode=keyRockerCenter`, and `modifiers=commandKeyMask`. (Consumed by `SysHandleEvent` if the system handled `keyHold` of `vchrHardRockerCenter`.) `keyDown` event with `chr=vchrRockerCenter`, `keycode=0,` and `modifiers=commandKeyMask`. (If the `keyDown` event `vchr=vchrHardRockerCenter` is not handled and the system does not handle the `keyHold` event of `vchrHardRockerCenter`.)

On Treo 600 smartphones, associating custom behavior with the Center button simply entails handling the `vchrRockerCenter keyDown` event. Applications can handle the other events as well, although most applications will not need to. If an application handles the other events, it is responsible for making sure that multiple actions are not triggered by the Center button. For example, an application that performs an action on `keyUp` should ensure that no action is performed on `keyDown`.

On Treo 600 smartphones, by default, a Center button press and hold does nothing different from a Center button press. On Treo 650 smartphones and Tungsten T5

handhelds, however, if the Center button is pressed and held, the Attention dialog box is displayed.

As such, on Treo 650 smartphones and Tungsten T5 handhelds, an application cannot associate an action with the press of the Center button because it is not yet known whether the Center button is going to be pressed or pressed and held. If an application associates an action on press and then the button is pressed and held, two actions will be triggered by the Center button. On Treo 650 smartphones and Tungsten T5 handhelds, actions should occur on the *release* of the Center button and only if the Center button was not held.

To minimize the changes required to make Treo 600 smartphone applications work with Treo 650 smartphones and Tungsten T5 handhelds, on these devices a `vchrRockerCenter keyDown` event is generated on Center button release rather than on press, and it is only generated on release if the Center button was not held.

This means that an application can safely handle any `vchrRockerCenter keyDown` event it receives without the Center button triggering multiple actions. The application need not check to see if it is running on a Treo 600 smartphone, nor does it need to determine whether the Center button is held before handling the event. This also means that a continuous press `keyDown` event, a `keyHold` event, and a `keyUp` event for `vchrRockerCenter` are not generated on Treo 650 smartphones and Tungsten T5 handhelds. See the next section for what new events are generated on Treo 650 smartphones and Tungsten T5 handhelds.

New Center button events for Treo 650 smartphones and Tungsten T5 handhelds

`vchrHardRockerCenter` key events are generated on Treo 650 smartphones and Tungsten T5 handhelds in the same fashion that `vchrRockerCenter` key events are generated on Treo 600 smartphones. Only Treo 650 smartphone and Tungsten T5 handheld applications that need more information on the state of the Center button than what the `vchrRockerCenter keyDown` event provides need to handle `vchrHardRockerCenter` key events.

The system will not handle a `vchrHardRockerCenter keyHold` event if a form's custom handler, the handler associated with a form by `FrmSetEventHandler`, handles a `vchrRockerHardCenter keyDown` event. This prevents an action from occurring on both the press and hold of the Center button.

If an application did not handle the `vchrHardRockCenter keyDown` event and the system did not handle the `vchrHardRockerCenter keyHold` event, a `keyDown` event with `chr=vchrRockerCenter`, `keycode=0`, and `modifiers=commandKeyMask` is generated on release.

Even though an application can handle Center presses, doing so is not recommended because it prevents the Attention dialog box from being displayed when the Center button is held.

Paging

On Treo 650 smartphones and Tungsten T5 handhelds, paging through lists of records or through lines of text is easier than it is on Treo 600 smartphones. On Treo 600 smartphones, when the focus is on a multiline field that is not in interaction mode, Up and Down move the focus off the field and to the object above or below. On Treo 650 smartphones and Tungsten T5 handhelds, Up and Down still move the focus to the object above or below if the top or bottom line of text is showing. However, if the top or bottom line of text is not showing, then Up and Down pages the field's text up or down. Only after the top or bottom is reached and Up or Down is released and pressed again does the focus move to the object above or below. Users can still easily move the focus off the field by pressing the Center button if the field is in interaction mode, and then pressing Left or Right. Focus bars above and below a field convey that Up and Down behave in this manner.

Navigation behavior for tables is to be completely implemented by third-party applications. For tables in palmOne applications, the same paging behavior the system uses for multiline fields is used for Up and Down. Additionally, if a table uses Center to open a record, Left takes the table out of interaction mode. Without this extra functionality added to the Left button, there would be no easy way to move the focus off a table.

For consistency's sake and to ensure a good user experience, we recommend that third-party applications follow these navigation conventions in their tables as well.

Palm navigation macros

The navigation macros supported on Tungsten and Zire handhelds also work on Treo 650 smartphones and Tungsten T5 handhelds. They continue to be supported to help minimize the code paths that an application needs to take to run on multiple devices. The details of these macros are thoroughly documented in the `palmOneNavigator.h` header file.

Tips and pitfalls

Navigation Order

- Application developers must create a navigation resource any time they have a form whose initial navigation order is not the default navigation order. `FrmSetNavOrder()` and `FrmSetNavEntry()` are not intended to replace the use of navigation resources. A form that should initially have a custom navigation order should always have a navigation resource.

  Having the navigation information available at the time the system initializes the form is much cleaner than having the system initialize the form with the default navigation order and then having the order changed when the application performs its own form initialization. `FrmSetNavOrder()` and `FrmSetNavEntry()` are mainly for dynamically created forms or forms with navigation orders that change sometime after form initialization.

- For forms that are "navigation-aware" (that is., forms that have a navigation resource and/or call navigation API functions), we do not automatically update

the vertical navigation order as object attributes are changed, as objects are included in the order, or as objects are excluded from the order.

Application developers may have the expectation that the vertical order will automatically be updated because we do automatically update the vertical navigation order of forms that are not "navigation-aware". We are not, however, comfortable automatically determining the navigation order for all forms. We do so only when we have to—when a form does not know about navigation.

If a form knows about navigation, it is the developer's responsibility to specify the proper vertical order through a navigation resource and then update the vertical order as needed.

- Developers can enable basic navigation for existing applications by simply creating navigation resources for the application's forms and including the resources into the application's existing .prc.
- Popup lists do not technically receive the navigation focus. When they are not popped-up, they are not usable and therefore cannot receive the focus. When they are popped-up, the rocker keys have dedicated functionality—regardless of whether object focus mode is on or which object has focus. Therefore, there is no need to put popup lists in a custom navigation order (although no problems arise if they are placed in the order).

  If a form is using the default navigation order, the popup list will be included in the order but will most likely be marked as skipped (because the list will most likely not be usable when the form is initialized).

Focus

- Although `FrmSetFocus()` gives the focus to the specified object, it does not redraw the object. To give focus to a system-supported navigation object (controls, fields, or lists), an application should call `HsNavObjectTakeFocus()` on the object. `HsNavObjectTakeFocus()` sends a `frmObjectFocusTake` event for the object and `FrmHandleEvent()` processes the event by calling `FrmSetFocus()` on the object and redrawing the object.
- The effects of calling `FrmSetFocus()` to set the navigation focus will be lost if `FrmSetFocus()` is called in response to a `frmOpen` event. This is because a `frmObjectFocusTake` event that sets the form's initial navigation focus is sent just after the `frmOpen` event.

  To properly give an object the initial focus, a navigation resource with the object specified as the initial focus object should be provided.
- If the object with the navigation focus is hidden, the form will be in a state where there is no navigation focus. The application is responsible for setting the new focus after it hides the focused object.

  If the application fails to do this and the user presses a directional rocker key when there is no focus on the form, we will move the focus to the first object in the tab order.

Focus Rings and Redraw Problems

- An application may run into redraw problems when it controls the drawing and/or removal of focus rings (when they directly call `HsNavDrawFocusRing()` and/or `HsNavRemoveFocusRing()`).

  Drawing focus rings around an object and properly restoring an object when it loses the focus ring is very tricky to do. The rings can be drawn over an object's frame, over another object, and over pixels directly drawn to the screen. When the system draws and removes the ring, it takes these possibilities into account and also contends with clipping rectangles and objects that change appearance between when they receive the focus ring and lose the focus ring. The system manages these complications fairly well when it is controlling the drawing and removal of rings.

  For an application to properly handle these complications as well, application developers should have a basic understanding of the ring drawing and removal mechanism. Before a ring is drawn, the bits behind the ring are saved. When a ring is being removed, the bits behind the ring are restored, the object is redrawn, and the portion of any object that was behind the ring is redrawn.

  Therefore, it is important that an application always draw an object in its normal state BEFORE drawing a focus ring around it. If the appearance of an object with the focus ring needs to change (i.e., if an object's bounds needs to change) or if what's behind the focus ring needs to change (i.e., if the background color of the form needs to change), an application should remove the ring, make the changes, draw the changes, and then draw the ring again.

Fields

- To give a field the insertion point, `FldGrabFocus()` should be called. `FldGrabFocus()` will take care of enabling the insertion point and putting the field into interaction mode. Similarly, to take the insertion point away from a field, `FldReleaseFocus()` should be called. `FldReleaseFocus()` will take care of disabling the insertion point and taking the field out of interaction mode.

- Since navigation causes fields to constantly receive and lose the insertion point, we need a way to always set the proper shift state for a field when it receives the insertion point. We have therefore introduced the `hsNotifySetInitialFldStateEvent` notification. An application that has a field that should always have a particular shift state should register for the notification.

  When registering, it should pass the field's form pointer as the user data for the notification. When it receives the notification, it should compare the active form pointer with the form pointer passed as the user data. If the pointers match, it should then call `FrmGetFocus()` to see which object has the focus. If it is the field that it wants to set the shift state for, the application should set the shift state and then mark the notification as handled.

Handspring extensions

Available on:

- Treo 600 and Treo 650 smartphones

The Treo smartphone by palmOne was originally designed by Handspring, and Handspring created new extensions to the Palm OS to support Treo smartphones. You can find the details of the new APIs in the API guide available separately from palmOne on the palmOne web site, either in the palmOneAPIGuide.chm Microsoft Help file or the palmOneAPIGuide.html HTML Help file.

Tips and Tutorials

Available on:

- Treo 600 and Treo 650 smartphones
- Tungsten T5 handhelds

A framework for applications to display formatted help content is included in the SDK. This help content serves as an on-device training tool designed to provide customers with a quick introduction to a device, application, or accessory.

Terminology

Tips

Tips are a collection of one-screen HTML pages (`Lessons`) that provide succinct usage tips for an application. These Lessons are collected into one Topic.

Tips are accessed from a menu item in the application.

Tutorial

A Tutorial is a collection of several Topics.

Tutorials are accessed through an icon in the Application Launcher instead of directly from an application. A Tutorial consists of a menu, and a collection of ten or fewer Topics tied together in a logical manner. For example, the Tutorial that comes preinstalled on Treo smartphones contains Topics that provide customers with basic instruction on using their Treo smartphone.\

On the Treo 600 smartphone, the Launcher icon for the tutorial is labeled "Tutorial". On the Treo 650 smartphone, the Launcher icon for the tutorial is labeled "Quick Tour".

Topic

A Topic is a collection of Lessons on a common theme.

A Tips file generally has one Topic, while a Tutorial file usually contains multiple topics. Tutorial files generally have a menu page that lets the user select which Topic they want to see. For example, the first Topic in the tutorial that comes

preinstalled on the Treo smartphone, "Getting Started," covers basic familiarity with the Treo smartphone buttons and behavior. When you build an XML file, a Topic is defined by Sequence tags.

Lesson

A Lesson, sometimes referred to as a page, is a single screen within a Topic. It consists of text, images, or both. A single idea is usually conveyed by one Lesson, but more complex ideas may require multiple Lessons.

Content

Topic titles

Topic titles should fit on a single line—generally, 25 characters or less—and should clearly describe the logical connection between enclosed Lessons. Standard title capitalization rules should be used. When creating a Tips file, the Topic title should be the name of the application the Tips are about.

Lesson text

Most Lessons contain text. Standard manual writing guidelines should be used when writing text. Text should be limited to no more than 30 words per Lesson.

Lesson images

Whenever possible, images should be used to ground the text with specific examples of the device, application, or accessory in action. See the image creation guidelines for more information.

Tips and Tutorial structure

Before being converted into a Palm OS® PRC file, a Tutorial or Tips is a collection of HTML files and images organized in a directory. Components include:

- A folder to contain the complete contents of the Tips or Tutorial. This folder is referred to as the *root content folder*.
- A menu page that lists all the available Topics. This menu page is contained in an HTML document that is usually named index.html. Tips files that contain only one Topic do not need a menu page.
- An HTML page for each Lesson in each Topic:
  - If the HTML page references external content such as images or JavaScript, they must also be present within the root content folder, unless the external content is already in the device's ROM.
  - References to external content must be relative to the root content folder, not to any subfolders within the content. For this reason, it's frequently easiest to put all the content files into one folder and reference files by filename only.
  - HTML pages may also reference the shared content library in the device's ROM. See the shared content description document for more information.

  - A style sheet and JavaScript library that provide the standard palmOne Tips and Tutorial styles and formatting are included in the shared content library in the device's ROM. We strongly recommend that all Tips and Tutorial content use these files. See the shared content description document for more information.

- An XML document, usually named tips.xml, that defines the elements and structure of the Tutorial or Tips.

Menu document

Tutorials have a menu document written in HTML that let users select which topic they want to view. Tips files normally contain only one Topic, so they do not have menu documents. The HTML document that defines the menu is usually called index.html and contains the following elements.

Oscar-The HTML looks like it came from the Treo 600. All HTMl and screen shots need to be updated to the new Ace/Angus style.

Head

```
<html>
<head>
<meta name="HandheldFriendly" content="True">
<title>Tutorial Main Menu</title>
    <link rel="stylesheet" href="common/inc/style.css" type="text/css"
title="test_style">
    <script language="javascript" src="common/inc/tutorial.js"></script>
</head>
```

Where:

- `<meta name="HandheldFriendly" content="True">` prevents the rendering engine from transforming the content.

- `<title>Tutorial Main Menu</title>` is customizable.

  The content within the title tags should reflect the title of the Tutorial.

- `<link rel...>` and `<script language...>` tell the document which style sheet and JavaScript source file to use.

  These should refer to the shared style sheet and JavaScript files in the shared content library in the device's ROM as shown earlier.

Body

```
<body onLoad="setPage('menu','');">
```

Where:

- `setPage` is a JavaScript routine that performs cookie actions appropriate to the type of page.
  - The first parameter defines what actions should be taken. The first parameter for a menu page should always be titled "menu."
  - On a menu page, the second parameter should be empty.

Icon

```
<!-- icon -->
<div id="icon">
<img src="common/img/icon_menu.gif" height="42" width="35" border="0">
```

Where:

- `<div id="icon">` positions the icon.
- `<img src="common/img/icon_menu.gif">` points to the menu icon in the shared content library in the device's ROM. All menus should use this icon.

Header and Content

```
<!-- header -->
<div id="menu_frame">
<img src="common/img/spacer.gif" height="42" width="35" border="0" class="r_float">
<p class="title">Tutorial Main Menu</p>

<!-- content -->
<img src="common/img/spacer.gif" height="10" width="125" border="0"><br>
<a class="menu" href="get_start_1.html"><img src="common/img/bullet_empty.gif"
name="hsgs" width="16" height="8" border="0">Getting Started</a><br>
<a class="menu" href="keyboard_1.html"><img src="common/img/bullet_empty.gif"
name="hskb" width="16" height="8" border="0">Keyboard Basics</a><br>
<a class="menu" href="top_ten_1.html"><img src="common/img/bullet_empty.gif"
name="hstt" width="16" height="8" border="0">Top 10 Fun Features</a><br>
<a class="menu" href="upgrade_1.html"><img src="common/img/bullet_empty.gif"
name="hsup" width="16" height="8" border="0">Upgrading From a Palm OS Device</a><br>
```

Where:

- `<div id="menu_frame">` defines the position and background image for the menu frame. This should not be changed.
- `<p class="title">Tutorial Main Menu</p>` defines the color and position of the title that will appear on the Tutorial menu page. Replace "Tutorial Main Menu" with the title of your Tutorial. When testing your Tutorial, make sure your title fits in the allotted space.
- Each Topic that is to appear in your menu should be coded as follows:

  ```
  <a class="menu" href="get_start_1.html"><img src="common/img/
  bullet_empty.gif" id="ft01" width="16" height="8" border="0">Getting
  Started</a><br>
  ```

  - Replace `get_start_1.html` with the name of the HTML document that defines the first Lesson in the Topic.
  - Replace `hsgs` with a code uniquely identifying the Topic. This code is used on the last Lesson of a Topic, so make a note of it for future reference. This is used to update the bullet to a checkmark once the Topic is completed by the user. See the built-in Quick Tour for an example of this behavior.
  - Replace `Getting Started` with the name of the Topic.

Footer

```
<!-- footer -->
```

```
<span id="menu_footer">Scroll Up or Down and press Center to select a topic</span>
</body>
```

This footer text should appear at the bottom of every menu. Do not change this section.

Lesson document

To make it easier to understand the structure, Lesson documents should be given a consistent naming structure. For example, the Lessons in the "Getting Started" Topic are named `get_start_1.html`, `get_start_2.html`, `get_start_3.html` and so on.

Head

```
<html>
<head>
<meta name="HandheldFriendly" content="True">
   <title>Getting Started</title>
   <link rel="stylesheet" href="common/inc/style.css" type="text/css"
   title="test_style">
   <script language="javascript" src="common/inc/tutorial.js"></script>'
```

Where:

- `<meta name="HandheldFriendly" content="True">` prevents the rendering engine from transforming the content.
- `<title>Getting Started</title>` is customizable.

  The content within the title tags should reflect the title of this Topic. All Lessons in the same Topic should use the same title. Tips files should use the name of the application that the Tips are about as the title of all Lesson documents.
- `<link rel...>` and `<script language...>` tell the document which style sheet and JavaScript source file to use.

  These should refer to the shared style sheet and JavaScript files in the shared content library in the device's ROM as shown in the example. The `<script language...>` tags are necessary only on the last Lesson of a Topic.

Body

```
<body onLoad="setPage('end','hsgs');">
```

Where:

- The last Lesson of a Tutorial Topic should have a body tag like the one shown in the example.
- `setPage` is a JavaScript routine that performs cookie actions appropriate to the type of page.
  - The first parameter defines what actions should be taken. If this is the last Lesson in the Topic, the first parameter should be set to `end`.
  - The second parameter should be the same as the code used to uniquely identify this Topic on the menu page.

  - This is used to update the bullet to a checkmark once the Topic is completed by the user. See the built-in Quick Tour for an example of this behavior.

- All other Lessons should have a regular `<body>` tag.

Icon

```
<!-- icon -->
<div id="icon">
<img src="common/img/icon_gs.gif" height="42" width="35" border="0">
<div id="icon"> positions the icon.
```

`<img src="common/img/icon_gs.gif"...>` points to the icon used for this Topic. Replace `common/img/icon_gs.gif` with the path to your icon image relative to the root folder. All Lessons in the same Topic should use the same image. See the attached image guidelines for more information.

Header and Content

```
<!-- header -->
<img src="common/img/spacer.gif" height="42" width="35" border="0" class="r_float">
<p class="title">Getting Started</p>

<!-- content -->
<img src="common/img/spacer.gif" height="5" width="125" border="0"><br>
<P>
<B>Congratulations on your purchase!</B>
</P>

<P>
The following tips will help you get started.
</P>

<img src="common/img/key_fiveway.gif" width="51" height="34" class="cap_none" border="0"
style="margin: 26 5 0 5">

<P>
Press the Center navigation button to go to the next page.
</P>

<!-- arrows and text -->
<img src="common/img/arrow_down_long.gif" width="8" height="30" border="0"
style="position:absolute; top:83; left:126;">
</div>

</body>
```

Where:

- `<div id="content_frame">` defines the position and area in which the content will appear. This should not be changed.

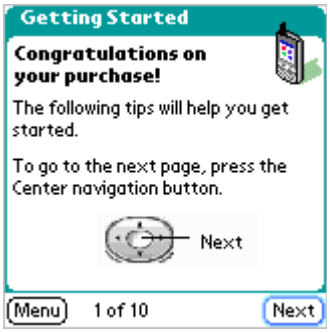
- `<p class="title">Getting Started</p>` defines the title for the Topic that will appear at the top of the screen. Replace “Getting Started” with your Topic title. The same title should be used for all Lessons in this Topic. Tips files should use the name of the application that the Tips are about as the title of all Lesson documents.

- All content should appear between `<!-- content --> <img src="common/img/spacer.gif" height="5" width="125" border="0"><br>` and `</div>`.

Lesson Content Formatting

There are several alternate ways of presenting content. In general, any Lesson may consist of the following:

- Text that flows vertically and horizontally in the main content frame
- Images that flow vertically within the main content frame and float right
- Images that flow vertically within the main content frame and float left
- Images that flow vertically within the main content frame and are centered
- Text or images that are removed from the flow of the main content frame and are absolutely positioned
- Text that appears in a graphic box on top of another image, otherwise known as a "callout."



In this example, all text but the word "Next" flows vertically and horizontally in the main content frame. The image of the navigation button flows vertically within the main content frame and floats right. The word "Next" and the arrow pointing to the center button are removed from the flow of the main content frame and are absolutely positioned. The HTML that defines this content is as follows:

```
<!-- content -->
<img src="img/spacer.gif" height="5" width="125" border="0"><br>
<p>
<b>Congratulations on your purchase!</b> The following tips will help you get started.
</p>

img src="img/fiveway.gif" width="51" height="34" class="capp_top" borders="0">

<p>
To go to the next page, press the center navigation button.
</p>

<!-- arrows and text -->
<img src="img/arrow_down_long.gif" width="8" height="30" border="0"
style="position:absolute; top:63; left:125;">
<span style="position:absolute; top:52; left:126;">Next</span>
```

Where:

- The text that will appear within the main flow of the content frame must be set off with paragraph tags (`<p>` and `</p>`).
- Because HTML aligns paragraphs and images at the top of their respective blocks, `<img src="img/fiveway.gif"...>` appears after the first paragraph to align with the top of the second paragraph. The path to the image file, as always, is defined from the root folder.
- `<img...class="cap_top">` defines the margins, which are extended—in this case on top—to make room for a caption. The following are predefined classes and their margins. If you need larger margins, you will need to define them within the `img` tag using the style attribute.
  - `cap_none: top:5; right:5; bottom:0; left:5; float: right`
  - `cap_top: top:30; right:5; bottom:0; left:5; float: right`
  - `cap_left: top:5; right:5; bottom:0; left:30; float: right`
- `<img src="img/arrow_down_long.gif"...>` defines the path from the root folder to the arrow image. You'll find predefined arrows in the attached template documentation; use those as a template if you create your own. Within the `img` tag is the style attribute: `<img...style="position:absolute; top:52; left:126;">`. Use this syntax to place elements exactly, always defining distance from the top-left corner at 0,0 pixels.
- `<span style="position:absolute; top: 52; left: 126;">` is used to absolutely position text. Always define distance from the top-left corner at 0,0 pixels.

This is an example of a callout. The following code generates the callout:

```
<span style="position: absolute; top: 40; left: 60;">
<table border="0" width="80" cellspacing="0" cellpadding="0" align="left" >
    <tr>
        <td><img src="common/img/pull_tl.gif" width="6" height="7" border="0"></td>
        <td><img src="common/img/pull_t.gif" width="68" height="7" border="0" ></td>
        <td><img src="common/img/pull_tr.gif" width="6" height="7" border="0" ></td>
    </tr>
    <tr bgcolor="#CCFFCC">
        <td><img src="common/img/pull_l.gif" width="6" height="55" border="0" ></td>
        <td>When on a call, press Center to Hang Up, or Right to choose another option.</
td>
        <td><img src="common/img/pull_r.gif" width="6" height="55" border="0" ></td>
```

```
    </tr>
    <tr>
        <td><img src="common/img/pull_bl.gif" width="6" height="7" border="0" ></td>
        <td><img src="common/img/pull_b.gif" width="68" height="7" border="0" ></td>
        <td><img src="common/img/pull_br.gif" width="6" height="7" border="0" ></td>
    </tr>
</table>
```

Where:

- `<span style="position: absolute; top: 40; left: 60;">` and `</span>` absolutely positions the entire callout on the screen.

  Always measure to the top and left of the block being positioned from the top-left corner of the screen at 0,0 pixels.

- `<table border="0" width="80" cellspacing="0" cellpadding="0" align="left">` starts the table that will draw the callout.

  The width can be adjusted to accommodate content, but should be no wider than 90 pixels.

- In the first row, `<td><img src="common/img/pull_tl.gif" width="6" height="7" border="0"></td>` defines the image of the top-left corner of the callout, `<td><img src="common/img/pull_t.gif" width="68" height="7" border="0"></td>` defines the image for the top bar of the callout, and `<td><img src=" common/img/pull_tr.gif" width="6" height="7" border="0">` defines the image for the top-right corner of the callout in the shared content library in the device's ROM.

  Always use these graphics when creating a callout. Adjust the width of `pull_t.gif` so that the width of all three images equals the width of the table.

- In the second row, `<tr bgcolor="#CCFFCC">` defines the background color of the callout.

  Always use #CCFFCC for the callout background color. `<td><img src=" common/img/pull_l.gif" width="6" height="45" border="0" ></td>` defines the left-side vertical bar of the callout, and `<td><img src=" common/img/pull_r.gif" width="6" height="45" border="0" ></td>` defines the right-side vertical bar of the callout in the shared content library in the device's ROM. The height of these two elements should be the same, and should be adjusted to encompass all of the text in the second cell: `<td>You'll see the number as you dial at the top of the screen.</td>`.

- In the third row, `<td><img src=" common/img/pull_bl.gif" width="6" height="7" border="0"></td>` defines the image of the bottom-left corner of the callout, `<td><img src=" common/img/pull_b.gif" width="68" height="7" border="0"></td>` defines the image for the bottom bar of the callout, and `<td><img src=" common/img/pull_br.gif" width="6" height="7" border="0">` defines the image for the bottom-right corner of the callout in the shared content library in the device's ROM.

  Always use these graphics when creating a callout. Adjust the width of `pull_b.gif` so that the width of all three equals the width of the table.

Some content may be in the form of bulleted lists. Because HTML doesn't support indented lists, use the following code to align numbered lists:

```
<p class="hanging_indent">
1. Press <img src="img/home_key.gif" width="21" height="23" border="0"><br></p>

<img src="img/keypad.gif" width="66" height="56" border="0" class="cap_none">

<p class="handing_indent">
2.Dial numbers directly from the keyboard dialpad.
</p>

<p class="hanging_indent">
3. Press Center to place the call.
```

Where:

- Each list item should be enclosed in a `<p class="hanging_indent">` and `</p>` tag.
- Each step is numbered individually.

XML document

An XML document, usually named tips.xml, catalogs the elements and structure of the Tutorial. It tells the PRC file creation utilities which content should be added into the Tips or Tutorial PRC file. It also defines the ordering of the lessons and the structure of the topics. It is an XML-formatted file that is processed by createTipsRsc.exe to create a standard Palm OS XRD resource file. For information about this process or clarifications about tips.xml syntax, please examine the source code for createTipsRsc.exe.

All XML tags and tag attributes are case sensitive.Filenames in the tips.xml file as well as within the HTML content are case sensitive. To make things simpler and easier, we strongly recommend that all filenames within the tips.xml file and within all tags in the content be in lowercase letters.

The header of the XML document is formatted as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Tutorial Name="Device_Tutorial " Base="Tutorial_Tips" Description="Device
Tutorial">
```

Where:

- `<?xml version="1.0" encoding="UTF-8"?>` defines the document type. This should not be changed.
- `<Tutorial Name="Device_Tutorial" Base="Tutorial_Tips" Description="Device Tutorial">` needs to be edited.

  Replace `Device_Tutorial` in both the Name and Base attributes with the name of the PRC database that will be created by the Tips and Tutorial creation utilities. Replace `Device Tutorial` with the human-readable name of the Tutorial file you want to create when the XML document is compiled.

```
<Page Base="1" Default="1" File="index.html"/>
```

- Tutorials may contain pages that are not part of a Topic, such as the main menu page. The syntax shown demonstrates how this is accomplished:
  - `Base="1"` tells the creation utilities that this is a page that is shown directly and may load other content, as opposed to included pages such as images or style sheets.
  - `Default="1"` tells the creation utilities that this is the page that should be displayed when this tutorial is first opened. Only one page in a tips.xml file should have the Default attribute set.
- Because the content in a Tips file is usually all in one Topic, Tips files usually do not have this form of the Page tag in their tips.xml files.
  - In a Tips file, the first page of the Topic should be flagged `Default="1"`

```
<Sequence Menu="index.html">
    <Page File="get_start_1.html"/>
    <Page File="get_start_2.html"/>
    <Page File="get_start_3.html"/>
    <Page File="get_start_5.html"/>
    <Page File="get_start_6.html"/>
    <Page File="get_start_7.html"/>
    <Page File="get_start_8.html"/>
    <Page File="get_start_9.html"/>
    <Page File="get_start_10.html"/>
    <Page File="get_start_11.html"/>
    <Page File="get_start_12.html"/>
    <Page File="get_start_13.html"/>
  </Sequence>
```

- The sequence tag defines one topic. Because a Tips file usually contains one Topic, it will usually contain one pair of Sequence tags. Tutorials usually contain multiple Topics, so they will usually contain multiple pairs of Sequence tags.
- For a Tutorial, the sequence tag looks like this: `<Sequence Menu="index.html">`. The Menu attribute tells the system where to navigate when the Menu button is tapped. If you've named the menu page something other than index.html, you will need to change it here.
- For a Tips file, the sequence tag looks like this: `<Sequence Done="1">`. The Done attribute tells the system to show a Done button instead of a Menu button. The Done button, when clicked, exits Tips and returns the user to the calling application.
- `<Page File="get_start_1.html"/>` tells the creation utilities what HTML pages contain the Lessons for this Topic. Order is important in this section; the Lessons will be shown in the same order in which the Page tags appear within the Sequence tag.

```
<Page File="inc\style.css"/>
<Page File="inc\tutorial.js"/>
<Page File="inc\active_call.gif"/>
```

- All other files used by the Tutorial or Tips file—other than shared content already in the device's ROM—need to be defined here using the relative path from the root folder. For instance, all images used by the Lesson pages must be included here. Order is not important in this section.

Converting Tips and Tutorial content in a PRC file

What you need

The following list of utilities and guidelines are what you will need to create Tips and Tutorials. The utilities are provided in the SDK under TipsTutorialsUtil.zip.

The contents of the TipsTutorialsUtil.zip file are as follows:

- Content developed using the guidelines in this chapter.
- The tips.xml file that defines the content in the shared content library in the device's ROM. This file should be copied into the root content folder.
- The createTipsRsc.exe program. The Perl source code for this program is also included for your reference.
- The Palm-BinTool.exe program.
- The PalmRC.exe program.
- A creator ID for the resulting PRC file. We highly recommend that the Tips and Tutorial PRC file have a different creator ID from that of the main application. This is the same type of creator ID as a standard Palm OS® application and can be obtained from the PalmSource web site.

Converting content into an XRD resource file

To convert the content into a Palm XRD file, open a command prompt and change directory (cd) to the directory with the tips.xml file. (The tips.xml file is described in the guidelines.) Then run `createTipsRsc`.

The syntax for the `createTipsRsc` command is as follows:

```
createTipsRsc -i tips.xml -o tips.xrd -c CREATORID -s
shared_content.xml -p c:\bin\palm-bintool.exe -f
```

Where:

- `-i` specifies the input XML file.
- `-o` specifies the output XRD file.
- `-c` specifies the creator ID for the final PRC file. `CREATORID` is a filler. You should obtain an authentic one from PalmSource.
- `-s` specifies the XML file that describes the shared content library in the devices ROM.

- `-p` specifies the full path to the palm-bintool.exe file.
- `-f` is an optional flag that sets the backup flag on the final PRC file. This tells the HotSync® to back up this file and to restore it automatically after a hard reset.
- `-l` is an optional flag that sets the current locale. For instance, `esES`.

Converting a Palm XRD resource file into a PRC

To convert the XRD file into a PRC, run PalmRC with the following syntax:

```
PalmRC.exe tips.xrd -p ARM -overlayFilter BASE -target 4.0 -o tips.prc
```

Where:

- `tips.xrd` is the name of the input XRD file.
- `tips.prc` is the name of the output PRC file.

Do not modify any of the other command-line parameters.

The resulting PRC file can then be placed onto the device through a normal HotSync operation.

Displaying Tips and Tutorial content

Use the methods described in this section to display Tips and Tutorial content.

Displaying Application Tips

To display tips within an application, launch the Blazer® web browser in Tutorial mode. The command block should be a pointer to a null-terminated string that is the name of the PRC database containing the application tips to be displayed. After the browser exits, the calling application is relaunched automatically.

```
#define myappTipsDbName                 "MyApp_Tips"
#define myappTipsDbNameLength           10

void DisplayTips() {
   Char* startPage;
   startPage = MemPtrNew( myappTipsDbNameLength + 1 );
   MemPtrSetOwner(startPage, 0);
   StrCopy( startPage, myappTipsDbName );
   AppLaunchWithCommand( hsFileCBlazer3,
sysAppLaunchWebBrowserTutorialMode, startPage );
}
```

Displaying a Tutorial

There is little programmatic difference between displaying Tips and displaying a Tutorial. Tips are displayed by launching the Blazer web browser in Tutorial mode from within an application, whereas a Tutorial is displayed by launching the Blazer web browser in Tutorial mode from a stub application that appears in the Launcher. This stub application has two functions:

- Provide a user-visible entry point into the Tutorial by displaying an icon in the Launcher.
- Launch the Blazer web browser in Tutorial mode with the command block pointing to a null-terminated string that is the name of the PRC database that contains the Tutorial to be displayed.

The following is an example of a stub application:

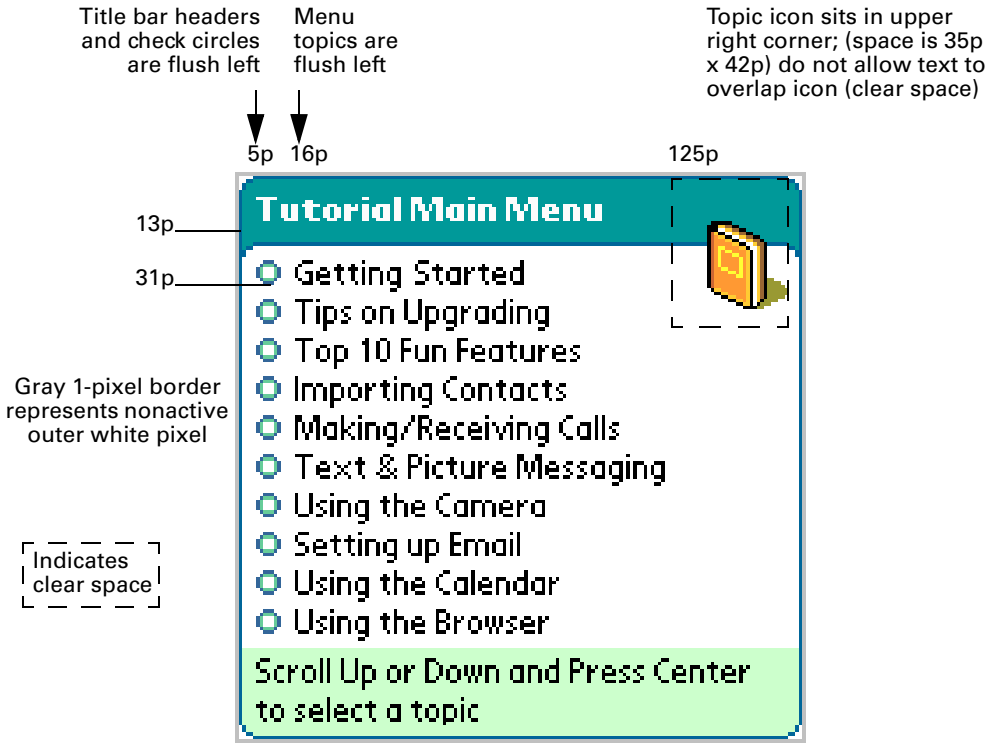
```
#define Tutorial_Start_Page "Tutorial_Tips"
#define Tutorial_Start_Page_Length 13

UInt32  PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
  if (cmd == sysAppLaunchCmdNormalLaunch) {
    Char* startPage;
    startPage = MemPtrNew(Tutorial_Start_Page_Length + 1);
    StrCopy(startPage, Tutorial_Start_Page);
    MemPtrSetOwner(startPage, 0);

    AppLaunchWithCommand(hsFileCBlazer3,
sysAppLaunchWebBrowserTutorialMode, startPage);
   }

  return 0;
}
```

Graphic Element Design Guidelines

This section details the graphic element design guidelines.

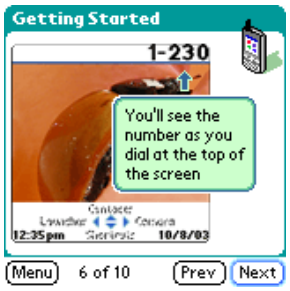
Tutorial Main Menu

The tutorial main menu is as follows:



Icons for upper right corners: Each icon used per chapter should be drawn in Photoshop at 30/60 degree increments using 2x1 stepping. Each icon should face left and have a cast shadow to the right.

Tutorial Content Pages



Content Page: large graphic & pull quotes
Full screens should be approximately 70% and placed on the left side of the screen. Pull quotes should be placed to the right as shown, with arrow pointing to area that text is referencing. Please use arrows (and circles) that exist in the template or request a set from palmOne.

Same clear space
(32p x 42p) applies
on content pages

Content Page: graphic (tall) & text
All graphics, except when they completely fill the screen, should be placed in the lower right corner. Use arrows with or without text to point out information and circles to call out specific details. Small graphics should be enlarged so that they are clear.

Same clear space
(35p x 42p) applies
on content pages

Content Page: graphic (wide) & text
All graphics, except when they completely fill the screen, should be placed in the lower right corner. Crop the images so that they are recognizable.

Content Page: Text only
Clear space for the icon still applies. For "scroll up or down" please use the same format from the Main Menu.

All button or key graphics should appear as shown in this screen.

Do not crop buttons like this:

Content Page: graphic (centered) & text
Certain graphics will look better when centered. This is the exception, not the rule. Note that elements such as buttons or keys from the device should be cropped as shown above.

Graphics: All icons used in the upper right corner should be drawn in Photoshop. All other placed images, such as screen shots or device images, must be taken from the original source files and reduced to fit into the space allowed per page. All images should be saved out of Photoshop using "save for web" at actual size in either jpeg or gif format, whichever is smaller. For JPEG, quality setting should be 20 or 30, depending on the image, with 0 blur. For GIF, preferences should be set to GIF 32 No Dither.

All device images are the property of palmOne, and may be used with permission.

Images that are in the shared content library in the device's ROM

The images available directly from the device's ROM for your own Tips and Tutorials can be found in the utility section of the SDK available at pluggedin.palmone.com.

Full-Screen Writing API

Available on:

- Tungsten T5 and Tungsten T3 handhelds
- Zire 31 and Zire 72 handhelds

This section provides reference information for the full-screen writing feature that is programmatically provided through the GoLCD (Graffiti® 2 writing on LCD) Manager API. You can use the functions in this API to enable or disable full-screen writing, enable and disable the Graffiti 2 shift indicator (GSI) as a control for full-screen writing, enable and disable Graffiti 2 inking, and even change the colors of Graffiti 2 inking and the GSI.

The GoLCD API is declared in the header file `PalmGoLCD.h`.

Overview of the full-screen writing feature

Full-screen writing allows users to enter Graffiti 2 characters in the application area of a handheld's display as well as in the Graffiti 2 writing area.

By setting Graffiti 2 Preferences, users can enable full-screen writing and choose whether to show, or *ink*, the Graffiti 2 strokes in the application area.

In each application the availability of full-screen writing is indicated by a shaded, rectangular Graffiti 2 shift indicator (GSI) in the lower-right corner of the display. The user can tap the GSI to turn full-screen writing off and on. The GSI appears as an outline when full-screen writing is off, and as a solid rectangle when it is on. In addition, the shift indicator, punctuation-mode indicator, and shift-lock indicator appear superimposed on the shaded rectangle when the user draws the appropriate Graffiti 2 strokes to activate those modes.

Graffiti 2 strokes in the application area are distinguished from taps on application controls, depending on the duration and direction of tap-and-hold events. GoLCD interprets pen events in the writing bounds of the application area, which you can set using `GoLCDSetBounds`. If the pen is held down for a certain length of time and travels significantly across the screen in the writing bounds area, GoLCD enters `goLcdGraffitiMode` and interprets all pen events as Graffiti 2 strokes. GoLCD exits `goLcdGraffitiMode` and stops interpreting pen events as Graffiti 2 strokes when the pen is lifted from the screen for a certain amount of time. For more information, see the API guide.

CHAPTER 8

Applications

This chapter details the application features and APIs available in some of the palmOne™ applications.

Web Browser API

Available on:

- Treo™ 600 and Treo™ 650 smartphones
- Tungsten™ T5 handhelds

The palmOne web browser is a powerful handheld web browser designed for the Palm™ operating system that can access multiple Internet content formats, including HTML, xHTML, cHTML and WML. With the web browser, a user can download a wide range of web pages from the Internet and view them on the compact screen of the user's device. The key features of the web browser are as follows:

- An intuitive graphical interface
- Support for multiple markup languages
- Support for secure web sites
- Two modes of display:
  - Optimized mode for optimization of content for Palm OS devices
  - Wide Page mode for the display of content similar to that of a desktop browser
- 5-way navigation support
- Ability to download applications, ring tones, images, and more
- A variety of new UI features including History, Saved Pages, and Beaming Bookmarks

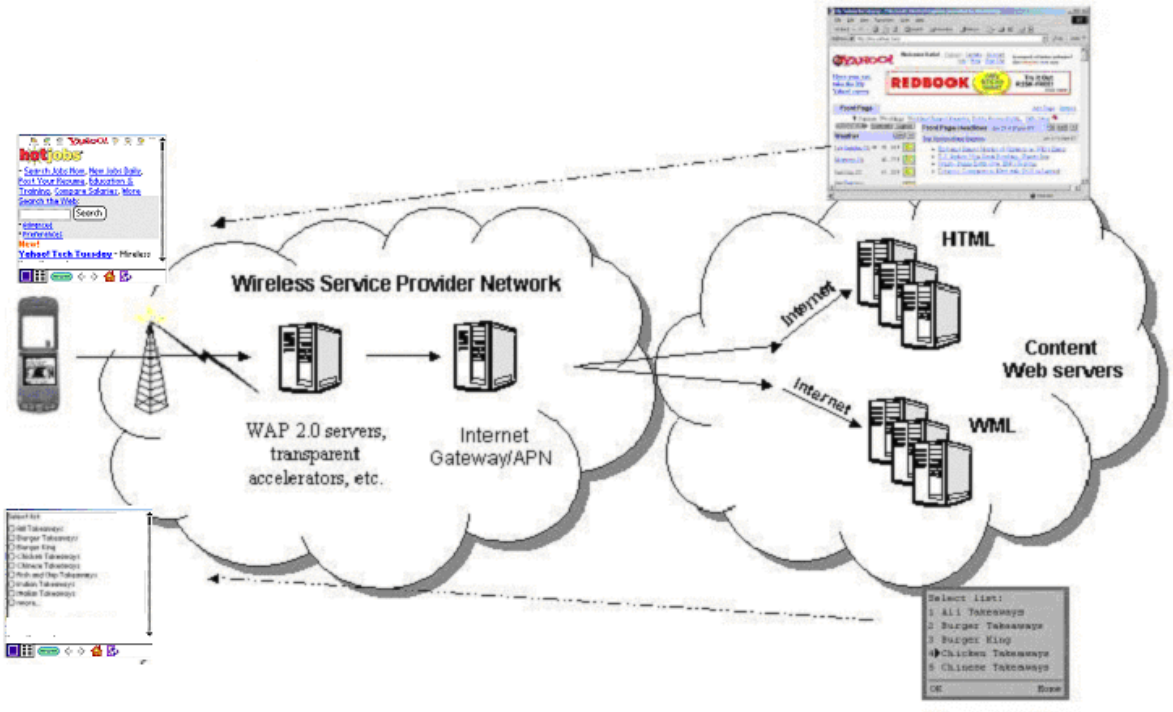
This section describes the various technical features of the palmOne web browser system that allow web site designers and programmers to deliver a better web experience.

How the web browser works

Versions of the web browser before Blazer version 3.0 were built in a proxy solution. The proxy server adapted pages from the remote web servers and streamed content to the client. The client application received the content from the proxy server and displayed the web page on the device's screen. Since Blazer 3.0 however, palmOne web browser is a proxyless, client-only browser. (It is capable of using a standard HTTP proxy in the same way a desktop browser would use a proxy.)

There are several reasons why this new architecture was chosen. For one, having a client-only solution allows devices to access more wireless service provider services like downloads and mobile content. Furthermore, the device processing power has reached the point where the content optimizations can be accomplished efficiently enough on the client.

The figure below shows how the proxy-less web browser accesses web pages.



The web browser uses the standard networking library (Netlib) of the Palm OS to make the connection to the wireless service provider infrastructure.

Web Browser Feature Overview

This section covers the various features of the web browser. This information is useful for understanding the software requirements, the technical features, and the various user interface elements.

Protocol stack support

The web browser uses the internet standard HTTP stack for communications. While the browser does support WML, there is no support for the WAP 1.x protocol stack.

Overview of key web browser features

The web browser incorporates a number of key technical features described in this section.

Intuitive graphical interface

The web browser's user interface takes advantage of the screen size on devices. Users have one-touch access to bookmarks, navigation, the home page, and new web pages. The web browser provides users with a familiar look and feel comparable to that of a desktop browser.

Support for multiple markup languages

The web browser supports a wide variety of markup languages. This allows users to access a wide variety of web and wireless content from a single browser. This also gives content providers some flexibility in determining what type of markup language to use. The various markup languages in the palmOne web browser are as follows. For special tags that the browser supports, please see "Palm OS Integration Tags."

Markup Language
HTML 4.01
XHTML 1.1
XHTML Mobile Profile
cHTML (iHTML)
WML 1.3
DHTML
DOM

In addition to the supported markup languages, the web browser also has extensive support for cascading style sheets and JavaScript.

See the latest "Browser Element Spec.pdf" for the tag-level support in the palmOne web browser available online.

Support for secure web sites

The web browser supports the following security features:

- End-to-end security using SSL 3.0
- 128-bit encryption using the RC-4 algorithm
- RSA-based key exchange

- RSA-based digital signature verification for verifying the authenticity of signed certificates, signed code, and so forth.
- MD5 and SHA-1 secure hash algorithms
- SSL 3.0 with server-side authentication only
- Support for X.509 certificates
- Indication of a secure connection by a lock icon in the toolbar or URL bar.

Optimization of content for device screens
The web browser optimizes the content to take advantage of device characteristics such as screen size, navigation, and so forth.

Download manager

The web browser incorporates new download management technology. This technology allows users to download files over the air using the web browser. There are a variety of applications for this new technology:

- Posting applications, ring tones, images, and so forth for your users to download
- Incorporating mechanisms in your application to either auto-check for updates and download them, or allow the user to manually download updates

Content support

When a user attempts to download a file, the download manager first checks with the Exchange Manager to see if a content handler has registered for that type of file. If there is a content handler registered, then the download proceeds as normal. If there isn't a content handler registered, Blazer 3.0 notifies the user that they must first install a content handler for that file type before they will be able to download the file. Blazer 4.0 allows all non-DRM-protected content to be saved to an expansion card, if one is present, even if there is no Exchange Manager handler for that content type.

For example, if the user tries to download a MIDI ring tone, because there is an application in the device's ROM that handles MIDI, the download proceeds. However, if the user attempts to download an Adobe Acrobat file, unless they have an application installed on their device that handles Acrobat files and that has registered with the Exchange Manager as a handler for Acrobat files, the user is not allowed to proceed with the download unless the user is running Blazer 4.0 and has an expansion card inserted in their device. If the user is running Blazer 4.0 and has an expansion card inserted in their device, the user could download the Acrobat file directly to the expansion card.

In order to avoid memory problems, Blazer 4.0 browser limits the size of downloaded files to 2MB. If a user tries to download a bigger file, an error message is displayed.

For your convenience, Blazer 4.0 lists every type currently registered in the Exchange Manager in the HTTP Accept header.

Download restrictions

The web browser includes the following download restrictions.

JPEG Images

Normal behavior for a browser is to render JPEG images within the browser, as opposed to downloading them and saving them on the device. Therefore, an image must be specially flagged if it is to be downloaded rather than rendered in the browser. There are two ways to flag an image for download:

- Use a descriptor file to precede the image.

  The download manager treats any images called by descriptor files as a download, and does not render them in the browser.

- Flag the image with a special MIMEtype.

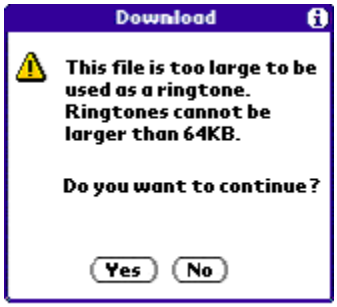
  If the web browser sees an image with the following MIME type, it will know the image is for download, not rendering: x-handspring-image/jpeg.

The only images officially supported for download are JPEG images. All other images are rendered in the browser.

In Blazer 4.0 web browser, the user can also tap-and-hold most images to bring up a Save As dialog box that allows the user to save the image to the Exchange Manager or to an expansion card (if present).

Ring tones

The palmOne Ring Tone manager can only play ring tones that are under 64k. If the user attempts to download a ring tone that is larger than 64k, they are presented with the following message:



Multiple file downloads

There are several schemes for supporting multiple file downloads. The most common is to combine the files in a ZIP file and then extract that file onto the device. In order for this to work, the user must first have an application on the device that can extract ZIP files and that registers for ZIP files with the Exchange Manager.

Alternatively, the Nutshell installer from Ecamm Network (www.ecamm.com) works well. You can use the installer to package multiple PRC and PDB files into a single PRC file. When the user runs the resulting installer PRC, the installer unpacks the PRC's and PDB's and installs them on the device.

Digital rights management
The download manager provides some digital rights management functionality. Specifically, wireless service providers can specify domains from which downloads will be forward locked which means that the file cannot be beamed, sent, and so forth. Forward lock applies to several multimedia types. It does not apply to PRC applications. PRC applications have built-in digital rights management, and those rights management schemes (for example, tying usage to HotSync® ID) should be employed at the content developer's discretion.

Launching the web browser on Treo smartphones

If you want your application to take the user directly to a web page, the web browser has the ability to launch a specific URL. The following code sample shows how to launch the web browser and go to a specific web page.

```
static void
LaunchBlazerWithURL(Char* urlP)
{
    Err err = 0;
    UInt16 cardNo;
    LocalID dbID;
    DmSearchStateType searchState;
    Char* url;

    // first check if web browser is installed
    err = DmGetNextDatabaseByTypeCreator(true, &searchState, sysFileTApplication,
        hsFileCBlazer3, true, &cardNo, &dbID);
    if (err)
        {
            // Display appropriate error dialog...
            return;
        }
    // ok, now let's call the web browser with the URL. Must first copy the URL,
    // because it will be disposed of by the system after the browser exits
    url = MemPtrNew(StrLen(urlP)+1);
    if (!url)
        {
            return;
        }
    StrCopy(url, urlP);
    // set the memory owner to zero, so it is not deleted
    // by the system when we switch apps
    MemPtrSetOwner(url, 0);
SysUIAppSwitch(cardNo, dbID, sysAppLaunchCmdGoToURL, url);
```

Launching the web browser in minimal mode

The web browser includes a minimal UI mode. This mode lets you display a web page with the simplest possible UI. The following code sample shows how to launch the web browser in minimal mode, and go to a specific web page.

```
static void
LaunchBlazerWithURL(Char* urlP)
```

```
{
   Err err = 0;
   UInt16 cardNo;
   LocalID dbID;
   DmSearchStateType searchState;
   Char* url;

   // first check if web browser is installed
   err = DmGetNextDatabaseByTypeCreator(true, &searchState, sysFileTApplication,
      hsFileCBlazer3, true, &cardNo, &dbID);
   if (err)
      {
         // Display appropriate error dialog...
         return;
      }
   // ok, now let's call the web browser with the URL. Must first copy the URL,
   // because it will be disposed of by the system after web browser exits
   url = MemPtrNew(StrLen(urlP)+1);
   if (!url)
      {
         return;
      }
   StrCopy(url, urlP);
   // set the memory owner to zero, so it is not deleted
   // by the system when we switch apps
   MemPtrSetOwner(url, 0);
SysUIAppSwitch(cardNo, dbID, sysAppLaunchWebBrowserMinimalMode, url);
```

VersaMail® application API

This section provides reference information for the VersaMail® application Device APIs. You can use these APIs to create attachment plug-ins for VersaMail attachments, add e-mail to VersaMail folders programmatically, create background network connections, and so forth.

Prerequisite knowledge required for using the VersaMail Device APIs

The VersaMail APIs assume a working knowledge of the following:

- VersaMail application itself
- palmOne™ programming
- palmOne 68k runtime environment

Overview of the VersaMail Device APIs

The VersaMail Device APIs consist of five main components that are documented in the remaining portions of this section:

- VersaMail Account Configuration

  This component allows you to distribute a PDB file to multiple users to set their initial VersaMail configuration automatically.

- Adding Outgoing E-mails to VersaMail Folders

  This component allows you to add e-mail messages to VersaMail folders programmatically. It can be used to distribute Welcome e-mail, corporate information e-mail, application information e-mail, and so forth.

- VersaMail Font library

  This component allows you to set the fonts displayed in the VersaMail Font Picker dialog box, as well as get and set information about fonts programmatically.

- VersaMail Attachment Plug-ins API

  This component allows you to create a plug-in for a type of attachment (for example, BMP attachments) which users can then use to send and view VersaMail e-mail attachments.

VersaMail Account Configuration

In an enterprise environment, it is often useful to set up all users with one or more baseline VersaMail account configurations. You can do so by distributing a database called __MMDevice.pdb. This feature will work in the VersaMail application, version 2.0 or later.

Overview of the MMDevice database

The MMDevice database consists of various records that set default values in VersaMail accounts. Each record follows the same basic syntax. When the VersaMail application is started on a handheld, it looks for the __MMDevice.pdb, sets account information based on the values therein, and then deletes __MMDevice.pdb. You should not confuse the file __MMDevice.pdb with _MMDevice.pdb (single underscore), which is usually created when an account configuration has changed.

__MMDevice.pdb is a case-sensitive name. It should have type and creator codes of `asc3` and a version number of 4.2 in hex (`0x0420`).

You can use the VMAccConfig application in the samples folder of the SDK to create an __MMDevice.pdb database automatically.

MMDevice database record syntax

The syntax for each record in the MMDevice database is as follows:

```
set <key> <account slot> <value>
```

with one or more spaces or tab characters between each field.

- `set`

  This string literal is required at the beginning of each record.

- `key`

  A string referencing the particular account parameter you want to set. For more information, see "MMDevice database record keys."

- `account slot`

  You should set this value to `0`. The account will be added to the end of the list of existing accounts. The VersaMail application accepts only a total of 8 accounts.

- `value`

  The value you want to set for the particular account parameter.

  The value is not parsed other than for keys that must be numeric. Trailing and leading whitespace are trimmed, however.

  To specify the default value for a record key, simply omit the entire record key line. If you include the record key line with a blank value, the key's value is explicitly set to an empty string that may or may not be valid for a particular record key.

For example, the following record would set the incoming mail server for the next account in the list to `mail.mac.com`:

```
set incomingServer 0 mail.mac.com
```

NOTE The `title` key is mandatory. Please see "title" for more information.

MMDevice database record keys

The following MMDevice record keys are valid. The keys are not case sensitive and, unless otherwise noted, default to an empty value.

apn
The string that specifies the default access point name (APN) to be used for the account. This refers to a service setting set in the Network preferences panel. If not specified, the default service is used.

connectionType
How the connection is made with the e-mail server. You can set it to `SyncOnly`, `PalmWireless`, or `ModemDialup`. For `ModemDialup`, the dial-up settings must be configured through the system's Network preferences panel.

emailAddress
The fully qualified address for the e-mail account.

incomingPort
The TCP/IP access port for the incoming e-mail server. The default value is blank, but the default TCP/IP access port for POP servers is `110`, and the default access port for IMAP servers is `143`. This value must be specified if the `serverType` value is specified. For more information, see "serverType."

incomingServer
The server for incoming e-mail. The incoming server is usually a POP or IMAP server.

outgoingPort
TCP/IP access port for the outgoing e-mail server. Typically, this is the SMTP port `25`.

outgoingServer
The server for outgoing e-mail. The outgoing server is usually an SMTP server.

password
The password for the account.

replyTo
The fully qualified e-mail address for the Reply-To header of outgoing mail.

rootMailbox
For an IMAP e-mail account, this specifies the root prefix of the account. This is typically not needed with most IMAP servers.

serverType
The protocol of the incoming e-mail server—`POP`, `PalmDotCom`, `Enterprise`, or `IMAP`. The default for this key is `POP`. If you specify this key, you must specify the `incomingPort`, as well. For more information, see "incomingPort."

title
The title of the account as shown in the VersaMail application. For example, "Personal Mail." You must enter a value for this record key.

useEncryptedPassword
Whether the account uses an encrypted password. Specify YES or NO. The default is NO.

useEsmtp
Whether the account requires authenticated SMTP. Specify YES or NO. The default is NO.

userName
The username for the account.

Adding Outgoing E-mail to VersaMail Folders

This section describes the various methods used to add outgoing e-mail to VersaMail folders. You can use the launch codes provided by the VersaMail application directly or use the Exchange Manager or Helper Notification methods documented in the *Palm OS Programmer's Companion, Volumes I and II* and the *Palm OS Programmer's API Reference*.

The direct and Exchange Manager methods work in the VersaMail application, version 2.0 or later. The Helper Notification method works in the VersaMail application, version 2.5 or later.

Overview of adding e-mail to the Outbox

There are three basic methods for adding e-mail to the VersaMail Outbox:

- Using the Exchange Manager

  An application can use the Exchange Manager Send command or the _send URL send scheme to sublaunch a Compose e-mail message form. The Compose form allows the user to add text to the message and save the message in the VersaMail Outbox or Drafts folder for later sending. Using the Send command displays a list box that gives the user a choice of Bluetooth® wireless technology, SMS, or the VersaMail application to send the e-mail. If the VersaMail application is not on the device, the list box does not contain the VersaMail option. Similarly, on devices that don't have the Bluetooth or SMS option, the VersaMail application is automatically launched.

  The following sample code shows how an application can use the Exchange Manager to sublaunch a Compose e-mail message form:

```
/* Use the Exchange Manager to create a New mail message */
/* Lets you add an attachment */

ExgSocketType exgSocket;
Err err = errNone;

Char      *textBuf = "test";
UInt32    size = StrLen(textBuf) + 1;

// it's important to initialize the structure to null values
```

```
MemSet(&exgSocket,sizeof(exgSocket),0);
exgSocket.description = "Testing";

/* A box pops up with an option to pick SMS, VersaMail application, etc.
 * You may not see the VersaMail option in the list if the VersaMail
 * application is not installed on the device.
 */

   exgSocket.name = "?_send:Sample.txt";

   /* send is important here */

   exgSocket.type = ".txt";
   err = ExgPut(&exgSocket);

 if (err == 0) {
       ExgSend(&exgSocket,textBuf,size,&err);
       ExgDisconnect(&exgSocket,err);
  }
```

For more information on using the Exchange Manager method, see the *Palm OS Programmer's Companion, Volume II* and the *Palm OS Programmer's API Reference*.

- Using Helper notifications

  This method supports a `SysUISwitch` from the launching application to a full VersaMail e-mail Compose form and back again. The full compose form is opened when the appropriate control is tapped in the launching application, and once the user has saved or sent the message, they are returned to the launching application.

  For more information on the Helper notifications method, see the *Palm OS Programmer's Companion, Volume I*, the *Palm OS Programmer's API Reference*, and the sample code `AddEmail` in the Samples folder of the SDK.

- Programmatically using the strategies documented in this chapter

  You must use the strategies documented in this chapter to allow a third-party application to add outgoing e-mail messages to a VersaMail folder.

  All of the strategies documented in this chapter use VersaMail launch commands.

VersaMail launch commands

There are three basic launch commands that you can use:

- `sysAppLaunchCmdAddRecord`

  This launch command creates a basic message without an attachment. You can use `MailAddRecordParamsType` to simply add the message to the VersaMail Outbox or `MailAddRecordsParamsTypePlus` to add the message to a different VersaMail category, such as Drafts.

  The attachment functionality of `MailAddRecordsParamsTypePlus` is ignored by the VersaMail application when it is used with `sysAppLaunchCmdAddRecord`. To use the attachment functionality of `MailAddRecordsParamsTypePlus`, you must use the next launch code, `MMPRO_ADD_MESSAGE_WITH_ATTACHMENT`, instead.

  `sysAppLaunchCmdAddRecord` is available in the `systemMgr.h` header file, `MailAddRecordsParamsType` is available in the `AppCmdLaunch.h` header file, and `MailAddRecordsParamsTypePlus` is available in the `PalmVMLaunch.h` header file.

- `MMPRO_ADD_MESSAGE_WITH_ATTACHMENT`

  This launch command works much like `sysAppLaunchCmdAddRecord`, except you can use `MailAddRecordsParamsTypePlus` to specify both a category and an attachment.

  `MMPRO_ADD_MESSAGE_WITH_ATTACHMENT` is available in the `PalmVMLaunch.h` header file.

- `MMPRO_LAUNCH_CODE`

  This launch command actually launches the VersaMail application itself and presents the user with a full e-mail compose form. The `MMPRO_LAUNCH_CODE` command uses the `MMProLaunchStruct` data structure to add attachments smaller than 64KB and `MMProLaunchStruct2` to add attachments larger than 64KB.

  `MMPRO_LAUNCH_CODE`, `MMProLaunchStruct`, and `MMProLaunchStruct2` are available in the `PalmVMLaunch.h` header file.

VersaMail Font library

This section provides information on how to use the VersaMail Font library. You can use the Font library to set the VersaMail Font Picker user interface and to work directly with font information on a device. More information on the Font library can be found in the API guide.

Checking whether the Font library is present

To check whether the Font library is present and loaded on a handheld, use `SysLibFind` to check for the library name `PalmSGFontLib` and a `0` error return value.

The VersaMail Font library is not included in the current versions of the VersaMail application. The library has to be retrieved, and you must perform a HotSync® operation to install it on a device. The library is available in the SDK.

Also, the Font library requires the font databases—palmOneSGHiResFonts.pdb for high-resolution devices and palmOneSGLowResFonts.pdb for low-resolution

devices. The font databases are available in the VersaMail application, version 2.5 or later and in the SDK.

Using the Font library

The font library is loaded during many handheld events, such as HotSync operations, but in order to use the functions in the Font library, you need to obtain and store the library reference number. You can use standard methods to maintain easy access to the reference number, such as globals with associated accessor routines or Palm OS® `FtrSet` and `FtrGet` calls. For more information, see `VMFontOpen`.

Also, before you use any of the Font Picker user interface functions, you must initialize the Font Picker user interface data structure using `InitFontUI`. You currently can apply only one style to a font. For example, you can apply bold or italic to a font, not bold and italic.

VersaMail Attachment Plug-ins API

This section provides information on how to create a plug-in so that users can view and send e-mail attachments using your plug-in in the VersaMail application. The VersaMail Attachment Plug-ins API is declared in the header file `PalmVMPlugin.h`.

This feature works with the VersaMail application, version 2.6 or later. The SDK includes a sample plug-in for TXT attachments.

Overview of how the VersaMail application handles plug-ins

The handling of e-mail attachments in the VersaMail application is controlled by the VersaMail Plug-in Manager.

When a user attempts to view a particular attachment in the VersaMail application, the VersaMail Plug-in Manager first tries to find the appropriate plug-in given the attachment's MIME type. If multiple compatible plug-ins are found, the VersaMail application displays a list so the user can choose the plug-in they want to use.

If no plug-in is found, the Plug-in Manager attempts to find an application registered with the Exchange Manager as the default application to handle the particular MIME type. If no Exchange Manager application is found, the message "No viewer for this attachment type" is displayed and the attachment can be viewed with the plain text viewer.

When a user attempts to send a particular attachment in the VersaMail application, all installed plug-ins are queried to find out what type of attachment each of them supports. When an appropriate plug-in is found, the VersaMail application queries the plug-in to display a list of possible attachments that can be sent. When a user selects an attachment from this list, the VersaMail application calls the appropriate plug-in to provide the data required to send the attachment.

Overview of plug-in design

A plug-in should be designed as a palmOne PRC file with the application type "mmpl" that supports the specific VersaMail launch commands detailed in this section. Because plug-ins are seamless to the user and should not appear to them to be separate programs, plug-ins should be created without an associated program icon. Furthermore, when a plug-in has finished viewing or sending an attachment, it should return control to the VersaMail application, not to the Home screen or to any other application.

The VersaMail application builds a dynamic list of all available plug-ins so that the user can install or remove them as desired. A separate plug-in should be provided for each type of file attachment. If there are multiple plug-ins that support the same type of file attachment, the user is provided with a list of possible plug-ins.

The VersaMail launch command that plug-ins must support are as follows:

- `MMPRO_PLUGIN_GET_INFO_LAUNCHCODE`

  This launch command is sent by the VersaMail application when it is trying to get information about what plug-ins are available to send a particular attachment.

- `MMPRO_PLUGIN_QUERY_LAUNCHCODE`

  This launch command is sent by the VersaMail application to get a list of possible attachments a user can send.

- `MMPRO_PLUGIN_EXTENDED_QUERY_LAUNCHCODE`

  This launch command is sent by the VersaMail application to get a more complete description of possible attachments a user can send. This launch command is provided for applications in which a simple, short description of a possible attachment is not enough information for a user to understand what the attachment is. For example, in Date Book, a simple date without the information associated with that date might not be enough information for a user to determine if they want to send the date as an attachment.

- `MMPRO_PLUGIN_SEND_LAUNCHCODE`

  This launch command is sent by the VersaMail application when a user has selected an attachment to send. The plug-in prepares the attachment so that it can be sent in response.

- `MMPRO_PLUGIN_RECEIVE_LAUNCHCODE`

  This launch command is sent by the VersaMail application when an attachment of the appropriate type is received. It allows the plug-in to do whatever you would like the plug-in to do with a received attachment. For example, you might want to display the attachment in a window that includes a Done button that users can tap when the have finished viewing the attachment.

PART III

Hardware Developers Kit

This part of the guide provides details on how to develop for hardware on Treo™ 650 smartphones and Tungsten™ T5 handhelds. For older devices, see the archive section of the pluggedin developer site from http://pluggedin.palmone.com.

CHAPTER 9

Multi-connector Specifications

This chapter defines the interfaces and interactions of the palmOne™ expansion multi-connector and its surrounding circuits and controlling software.

Overview

Available on:

- Treo™ 650 smartphones
- Tungsten™ T5 handhelds

This chapter specifies the electrical and software interface characteristics of the palmOne multi-connector. These characteristics include sets of various charging, cradle, and cable configurations, but from a handheld or smartphone perspective the electrical specification and systems software requirements are consistent across multiple devices. Specific devices may not, however, implement all of the features of the multi-connector.

The multi-connector supports the following features:

- Charging power from an external adapter with adapter detection ability
- Universal Serial Bus (USB)
- Serial communications (no flow control, logic levels)
- Dedicated HotSync® technology interrupt
- Power out
- Stereo headphone-level output
- Peripheral detection

The multi-connector interface supports interaction with the following devices:

- USB HotSync cables and cradles
- Other serial devices

  Due to serial peripheral detachment detection requirements, a device cannot be created that employs automatic serial peripheral detachment detection *and* utilizes the USB VBUS line for USB charging, signaling, or connections. Any device requiring both serial peripheral detection and USB functionality cannot automatically detect detachment of the peripheral through the normal serial peripheral detachment mechanism. Detachment for this situation must be handled through a nonstandard mechanism, such as serial error handling. For

more information on the peripheral detection mechanism, see "Peripheral detection."

- `Pass-through peripherals`

  These include peripherals that connect to a handheld or smartphone and have another connector to allow the peripheral to be connected to a USB HotSync cable or cradle.

Pinout of the multi-connector

The pinout, with or without mounts, is described in the following table. All pin references that follow in this chapter refer to the device connector pin-numbering scheme shown in this table.

Pin # on Device/ Multi-Connector	Pin # on Charger/ Adapter Connector	Pin# on Data/Cable Connector	Name	Direction with respect to the device	Default State with no attachment	Function
1	1		VDOCK	Power	CHRG_IN	DC charging voltage, 5V
2	2		ADAPTER_ID	Input	VCC, weak pull-up	Adapter identification
3	3		VDOCK_RTN	Power	GND	DC charging return
4	-	1	SHIELD	Shield	GND	Cable shield
5	-	2	VBUS	Power	VBUS_IN	USB charging voltage, 5V typical, 500 mA max
6	-	3	USB_DP	Input/output	Floating	USB Data +
7	-	4	USB_DN	Input/output	Floating	USB Data -
8	-	5	DGND	Power	GND	Digital ground, and VBUS return
9	-	6	Reserved	NA	NA	Do not connect
10	-	7	TXD	Input/output	VCC, weak pull-up	Transmit data, 3.3V logic level
11	-	8	RXD	Input	VCC, weak pull-up	Receive data, 3.3V logic level
12	-	9	HOTSYNC	Input	VCC, weak pull-up	HotSync input, active low, pulled up on device

Pin # on Device/ Multi-Connector	Pin # on Charger/ Adapter Connector	Pin# on Data/Cable Connector	Name	Direction with respect to the device	Default State with no attachment	Function
13	-	10	POWER_OUT	Output	High impedence	Power output to external devices
14	-	11	SPKR_L	Analog output	AC coupled	Speaker output left
15	-	12	SPKR_R	Analog output	AC coupled	Speaker output right
16	-	13	AGND	Power	GND	Analog ground
17	-	14	Reserved	NA	NA	Do not connect
18	-	15	SHIELD	Shield	GND	Cable shield

Shielding

On all peripheral devices, the shield pins 4 and 18 should be grounded with any available shielding system ground. For example, on a USB cable these pins should be connected to the USB cable outer shield, which in turn connects to the shield on the USB connector at the other end of the cable.

Where no external shielded ground is available peripherals should connect pins 4 and 18 to the peripheral's system ground.

USB

Pins 5, 6, 7, and 8 constitute the USB VBUS, D+, D-, and GND pins respectively.

The handheld is designed to accept the following parameters on pins 5, 6, 7, and 8:

Name	Description	Minimum	Average	Maximum	Units
(V)USB_VBUS_CHG	Input charging voltage	4.375	5.0	5.25	V
(V)USB_VBUS_CHG	Input serial peripheral detection voltage	2.97	3.3	3.63	V
(I)USB_VBUS_L	Input charging current, no negotiation, sunk from VBUS	-	-	100	mA
(I)IUSB_VBUS_H	Input charging current, with negotiation, sunk from VBUS	-	-	500	mA

Serial interface hardware

Pins 10 and 11 provide 3.3V logic-level serial connections with no dedicated hardware flow control pins. The direction of these pins with respect to the device is as follows:

- Pin 10 transmits from the handheld
- Pin 11 receives into the handheld.

The serial port connected to pins 10 and 11 supports the following bit rates and configuration options:

- 1,200 baud
- 2,400 baud
- 4,800 baud
- 9,600 baud
- 14,400 baud
- 19,200 baud
- 28,800 baud
- 38,400 baud
- 57,600 baud
- 115,200 baud
- 7 data bits
- 8 data bits
- No stop bits
- 1 stop bit
- Parity bit
- No parity bit

Both pins 10 and 11 are pulled high within the device by weak pull-ups. These GPIO are capable of waking the handheld from all operational modes, including sleep mode. For more information on how these weakly pulled, high input characteristics are used in the peripheral detection mechanism, see "Peripheral detection."

Pins 10 and 11 operate at 3.3V nominal voltage levels.

Treo 650 smartphones and Tungsten T5 handhelds are designed to accept the following parameters on pins 10 and 11:

Name	Description	Minimum	Average	Maximum	Units
(V)RXTX_INL	Input logic low voltage*	0	-	0.594	V
(V)RXTX_INH	Input logic high voltage*	2.904	-	3.63	V
(V)TX_OUTL	Output logic low voltage*	0	-	0.3	V
(V)TX_OUTH	Output logic high voltage*	2.67	-	3.63	V
(V)TX_OC	Open circuit TX line voltage*	-	3.3	-	V
(V)RX_OC	Open circuit RX line voltage*	-	3.3	-	V

*With respect to digital ground, pin 8

Serial interface software

The multi-connector serial pins interface with the Palm OS as a virtual serial port.

HotSync interrupt hardware

Pin 12 provides a HotSync Interrupt pin. The HotSync interrupt is weakly pulled high inside the device.

A HotSync interrupt is initiated when pin 12 is pulled to GND.

The HotSync interrupt is not used in the peripheral detection mechanism; it only initiates a HotSync operation.

Treo 650 smartphones and Tungsten T5 handhelds are designed to accept the following parameters on pin 12:

Name	Description	Minimum	Average	Maximum	Units
(V)HS_INL	Input logic low-voltage* triggering interrupt	0	-	0.594	V
(V)HS_OC	Open circuit line voltage*	-	3.3	-	V

*With respect to digital ground, pin 8

HotSync interrupt software

The HotSync interrupt is always set as an input. The default interrupt detection occurs on a falling edge.

Power output

Pin 13 provides a power output to power an external peripheral. This power output is limited to low-current capability only. The power output is normally driven LOW or floated as a high impedance signal to minimize the chances of a short to GND damaging the device.

Treo 650 smartphones and Tungsten T5 handhelds are designed to accept the following parameters on pin 13:

Name	Description	Minimum	Average	Maximum	Units
(V)POUT_L	Output inactive voltage*	0	-	0.363	V
(V)POUT_H	Output active voltage*	2.97	3.3	3.63	V
(I)POUT	Output current supplied	30	-	-	mA
(C)POUT	Minimum load series resistance to GND*	89	-	-	Ohms
(C)POUT	Maximum load capacitance	-	-	4.7	µF

*With respect to digital ground, pin 8

Audio detection

A Tungsten T5 handheld automatically detects and switches audio if it detects an attached audio peripheral and the audio peripheral indicates that a headset is inserted into the handheld.

Tungsten T5 handhelds should not, however, have a headset inserted into the headset jack and be attached to an audio peripheral at the same time. The audio signal would be shared between the two, resulting in a loss of volume on the headset and the multi-connector audio output channels.

Audio output

Pins 14 and 15 provide headphone-level stereo audio output. This output is intended to drive an external audio output device (such as a “boom box”), or to interface with a car kit.

Devices typically connect pins 14 and 15 directly to their headset stereo output signals.

Peripheral requirements

For information on the peripheral detection mechanism that drives many of the requirements listed in this section, see "Peripheral detection."

Peripherals are required to conform to the following specifications to ensure successful operation with Treo 650 smartphones and Tungsten T5 handhelds:

Name	Description	Minimum	Average	Maximum	Units
(C)PHL_LOAD	Capacitive load on POWER OUT pin	-	-	4.7	µF
(R)PHL_LOAD	Minimum load series resistance on POWER OUT pin	89	-	-	Ohms
(V)PHL_RXTX_INL	Serial line input logic low voltage*	0	-	0.594	V
(V)PHL_RXTX_INH	Serial line input logic high voltage*	2.904	-	3.63	V
(V)PHL_RX_OUTL	Serial handheld receive line output logic low voltage*	0	-	0.3	V
(V)PHL_RX_OUTH	Serial handheld receive line output logic high voltage*	2.67	-	3.63	V
(R)PHL_TX_DOWN	Pull-down to GND on TX*	1K	-	10K	Ohms
(R)PHL_RX_DOWN	Pull-down to GND on RX*	1K	-	10K	Ohms

*Pull-downs required for peripheral detection mechanism. May not be required on either or both of the RX and TX lines.

Audio peripherals

Audio peripherals should conform to the following requirements:

Name	Description	Minimum	Average	Maximum	Units
(R)PHL_A_DET	Maximum series resistance to ground on TX line for Audio Peripheral detection	-	-	1K	Ohms
(R)PHL_A_DET_HS	Maximum series resistance to ground on RX line for Audio Peripheral Headset Jack Insertion detection	-	-	1K	Ohms

Name	Description	Minimum	Average	Maximum	Units
(R)PHL_A_DET_NHS	Minimum series resistance to ground on RX line for Audio Peripheral Headset Jack Insertion Absence detection	10M	-	No maximum	Ohms
(R)AOUT_MIN	Minimum series resistance to ground on pins 14 and 15	8	-	-	Ohms

General serial peripherals

Serial peripherals should conform to the following requirements.

Name	Description	Minimum	Average	Maximum	Units
(T)PHL_SER_DLY == (T)DET_RXTX_DLY	Delay from POWER OUT applied to peripheral driving into Rx line	450	-	-	mS
RPHL_SER_DET	Series resistance to ground on TX and RX line before POWER_OUT applied for Serial Peripheral detection	-	-	100K	Ohms
VPHL_SER_DET	Minimum voltage on TX and RX line after POWER_OUT applied for Serial Peripheral detection	2.904	-	POWER_OUT	V
RPHL_SER_NDET	Maximum series resistance between VBUS and POWER_OUT for Serial Peripheral Detachment detection	-	-	10	Ohms

Peripheral detection

The multi-connector interface provides class-level peripheral detection. The class-level detection mechanism is performed using hardware detection with no software requirements on the peripheral.

Peripheral detection is initiated by triggering either the serial TX or serial RX lines in their GPIO states as falling edge interrupts. After detection of an interrupt, software in the device debounces the interrupted line for at least (T)DET_DBC milliseconds. If the condition causing the interrupt on the TX or RX lines still exists after (T)DET_DBC milliseconds, class-level detection initiates.

Class-level detection

Class-level detection involves sampling the serial TX and RX lines as GPIO both before and after the application of POWER_OUT. This provides four bits to define the attached peripheral, resulting in 12 possible attached configurations. (The situation in which both RX and TX stay high after attachment of the peripheral is invalid as no attachment interrupt is detected, thus invalidating four possible combinations.)

Class-level detection is currently supported by the Tungsten T5 handheld. The only class supported at this time is audio.

Peripheral attachment

Before attachment of any peripheral, the TX and RX lines are configured as GPIO and have weak pull-ups attached to each line.

When a peripheral is attached, at least one of the TX or RX lines must by definition be low. After debouncing the signal, the device samples the RX and TX lines and applies power to the peripheral through the POWER_OUT signal. The device then waits (T)DET_PWR_DLY milliseconds to allow the peripheral to power up, and then samples the TX and RX lines again.

If you want a peripheral to identify itself, it must have strong pull-downs on the appropriate serial communications lines. Some serial peripherals may not require such strong pull-downs because the impedance to ground of the unpowered serial input pins may provide small enough resistance to GND to allow detection to operate effectively. Also, the peripheral must not power the RX line until (T)DET_RXTX_DLY milliseconds after power is applied.

When a Tungsten T5 handheld detects that an audio peripheral is attached, the handheld automatically switches audio from the internal device speaker to the audio of the peripheral. As noted earlier, Tungsten T5 handhelds should not have a headset inserted into the headset jack and be attached to an audio cradle at the same time. The audio signal would be shared between the two, resulting in a loss of volume on the headset and the multi-connector audio output channels.

The truth table for class-level detection is as follows:

Before attachment		Peripheral attached, no POWER_OUT		Peripheral attached, POWER_OUT applied		Class
TX	RX	TX	RX	TX	RX	
1	1	0	0	0	0	Audio peripheral detected, headset not inserted
1	1	0	0	0	1	Reserved for future use
1	1	0	0	1	0	Reserved for future use
1	1	0	0	1	1	Reserved for future use

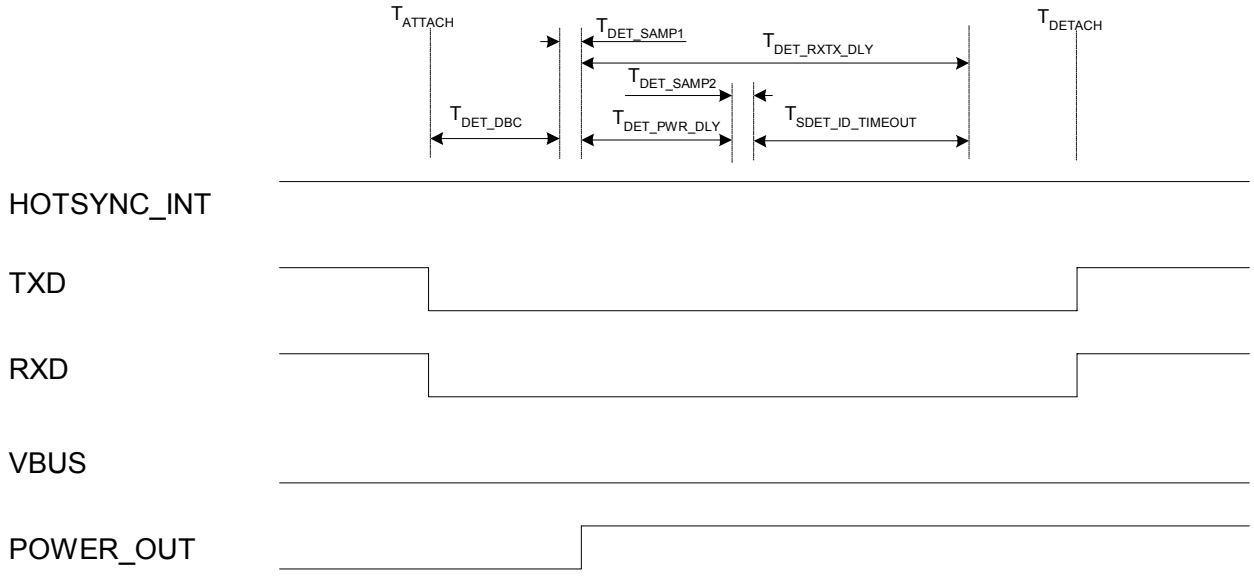
Before attachment		Peripheral attached, no POWER_OUT		Peripheral attached, POWER_OUT applied		Class
TX	RX	TX	RX	TX	RX	
1	1	0	1	0	0	Reserved for future use
1	1	0	1	0	1	Audio peripheral detected, headset inserted
1	1	0	1	1	0	Reserved for future use
1	1	0	1	1	1	Reserved for future use
1	1	1	0	0	0	Reserved for future use
1	1	1	0	0	1	Reserved for future use
1	1	1	0	1	0	Reserved for future use
1	1	1	0	1	1	Reserved for future use

Peripheral removal

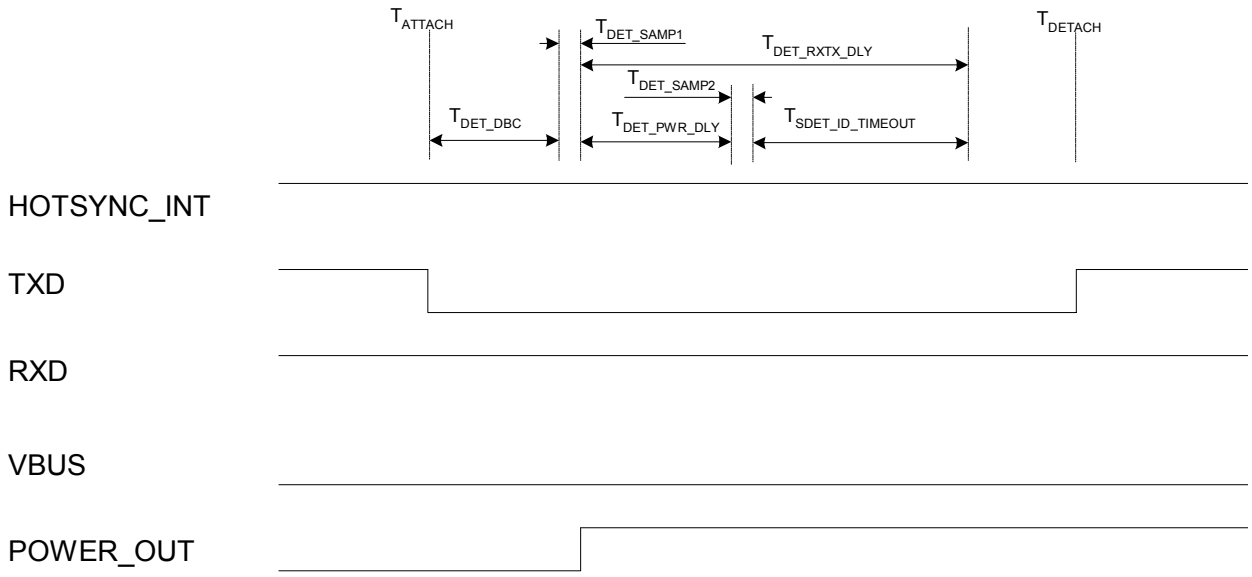
Detachment is detected by the TX and RX lines going high. The device detects these changing GPIO signals by detecting the rising edge, and initiates peripheral removal activities.

Audio peripheral detection timing diagrams

The following figure shows the timing diagram upon attachment of an audio peripheral with a headset not inserted.



The following figure shows the timing diagram upon attachment of an audio peripheral with a headset inserted.



Peripheral detection timing specifications

The following are the timing requirements for the peripheral detection mechanism:

Name	Description	Minimum	Average	Maximum	Units
(T)DET_DBC	Attachment interrupt debounce duration	150	-	-	mS
(T)DET_SAMP1	Time to read the state of the RX and TX GPIO before POWER OUT applied	-	-	50	mS
(T)DET_PWR_DLY	Delay from POWER OUT applied to reading the state of the RX and TX GPIO	200	-	200	mS
(T)DET_SAMP2	Window of time from maximum (T)DET_PWR_DLY to read the state of the Rx and Tx GPIO after POWER OUT applied	-	-	50	mS

Name	Description	Minimum	Average	Maximum	Units
(T)SDET_ID_TIMEOUT	Timeout from (T)DET_SAMP2 window finished until determination that peripheral is Serial Protocol Level detection peripheral	200	-	200	mS
(T)DET_RXTX_DLY	Delay from POWER OUT application to peripheral driving into RX line	450	-	-	mS

Interfacing with an audio peripheral

The method we recommend for interfacing with an audio peripheral is for the peripheral to be wired as described in the following table:

Name	Pin #	Recommended configuration
VDOCK	1	Charging source if charging device; otherwise no connect.
ADAPTER_ID	2	Connect to pin 3 if charging device with >=1Amp source; otherwise no connect.
VDOCK_RTN	3	Charging source ground if charging device; otherwise system ground.
SHIELD	4	Shield ground or system ground.
VBUS	5	No connect.
USB_DP	6	No connect.
USB_DN	7	No connect.
DGND	8	System digital ground.
USB_ID	9	No connect.
TXD	10	10K Ohm pull-down to DGND.
RXD	11	Insertion detection switch on headset jack, if any. Signal should connect to GND when no headset is inserted. Signal should be high impedance when headset is inserted.
HOTSYNC	12	No connect.
POWER_OUT	13	Connect to system power if required by peripheral.
SPKR_L	14	Audio left signal.

Name	Pin #	Recommended configuration
SPKR_R	15	Audio right signal.
AGND	16	System audio ground.
Reserved	17	No connect.
SHIELD	18	Shield ground or system ground.

PART IV

Debugging

This part of the guide provides details on how to debug problems with the code you create.

CHAPTER 10

Debugging

This chapter details how to debug problems with the APIs.

Debugging on Treo™ smartphones

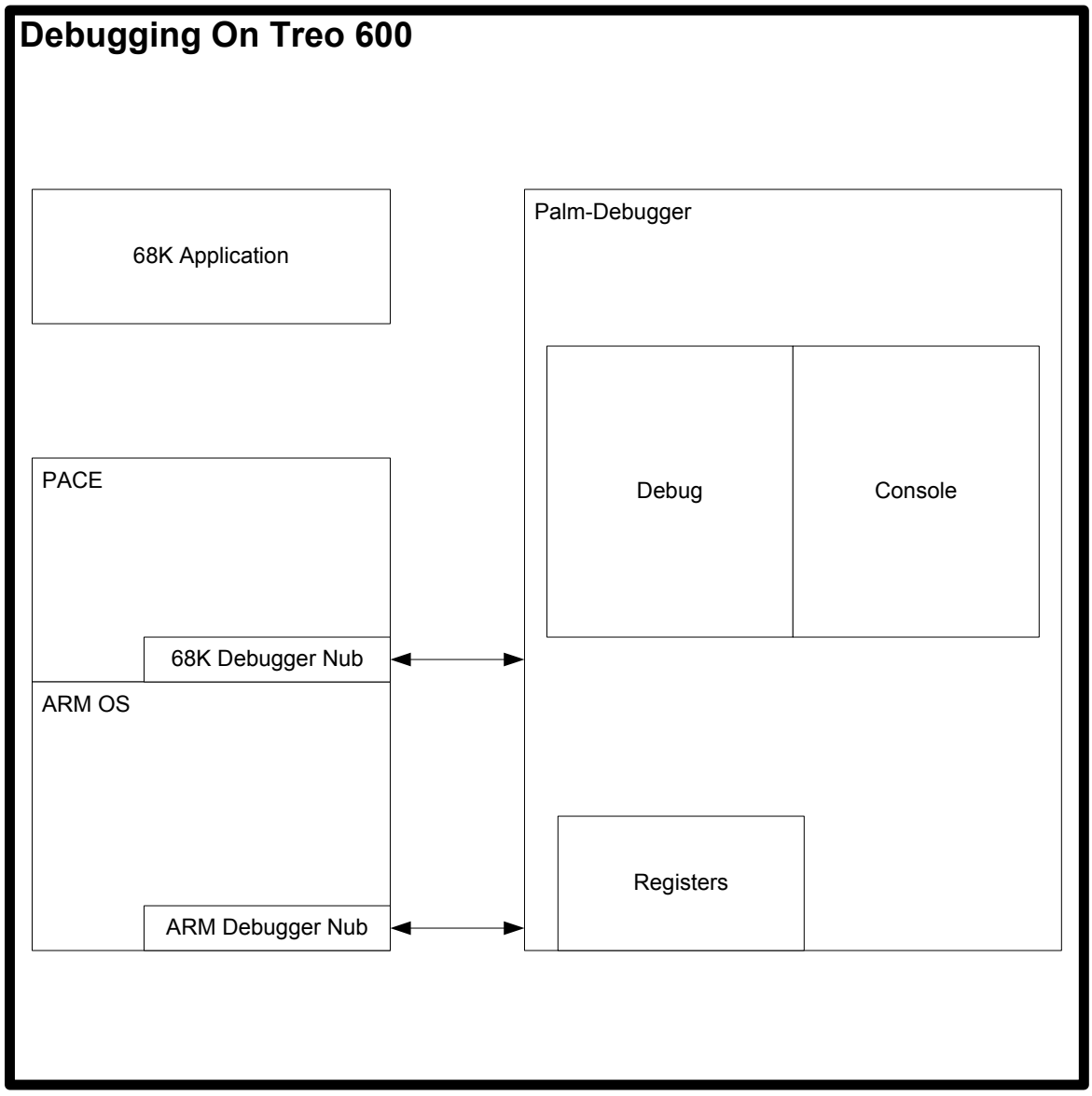
Available on:

- Treo™ 600 and Treo™ 650 smartphones

To support debugging on Treo™ smartphones, the Palm™ Debugger has been updated to communicate with the Palm OS® PACE *and* the ARM-based debugger nubs.

IMPORTANT There is no USB support for debugging on Treo 600 smartphones, so you'll need to use a serial cable if you want to do any debugging. The Treo 650 smartphone supports USB debugging except for project which requires you to use a serial cable.

The following figure shows the various Debugger nubs and where they are located within the Palm OS on a Treo smartphone.



Handling of fatal errors

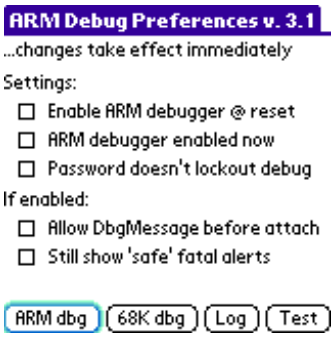
Fatal errors are handled differently on Treo smartphones. Fatal errors are not displayed to end users in the released product; if a fatal error occurs, the system performs a soft reset. However, before resetting, the smartphone saves a log of the error.

This process results in a better user experience. The user does not see fatal errors while the smartphone just keeps working. As the Treo smartphone is primarily a phone, its design is based on the belief that it is more important to keep the smartphone working than to show a fatal-error dialog box that asks the user to reset the smartphone.

Developers can access the error log in two ways:

- Type ##ERR (CDMA) or #*ERR (GSM) in the dial pad view and press dial (ERR = 377 on the dial pad).
- Press the Log button in the DebugPref application as described in the next section.

DebugPref for Treo smartphones



DebugPref is an application that allows you to configure how you want a Treo smartphone to behave when an error occurs. You can also trigger immediate debugging tests by tapping one of the buttons at the bottom of the DebugPref main dialog box. Debug Pref is available from the palmOne Developer web site at http://pluggedin.palmone.com.

A description of each setting and button is described in the following section.

Settings

- Enable ARM debugger @ Reset

  Puts the smartphone in debugger mode every time the system resets.

  One of the following actions will take place when a fatal error occurs:

  – ARM Debugger nub will start using the serial or USB port

  – 68K PACE Debugger Nub will start using the serial or USB port

- ARM debugger enabled now

  This is used to enable the ARM debugger. (For more information on ARM development, see the PalmSource™ web site.)

- Password does not lockout debug.

  Check this option to have the usual debug triggers enabled even if you have a password set for a smartphone.

  NOTE This option doesn't make a smartphone any less secure, because it is always possible to programmatically launch the Debugger if you have infrared (beaming) access to a smartphone.

- Allow DbgMsg before attach

  Works only with the Palm OS® Debugger, a PalmSource™ debugging tool currently not available to third parties.

- Still show "safe" fatal errors

  If the system determines that an error is not a major error, then it displays the error. If it determines it is a major error, the system either automatically resets the smartphone or launches the smartphone in debugger mode, depending on how the Enable ARM debugger @ Reset setting is set.

Buttons

The following options are available by tapping the buttons at the bottom of the screen:

- 68K dbg

  Immediately launch the ARM debugger nub.

- ARM dbg

  Immediately launch the 68K PACE debugger nub.

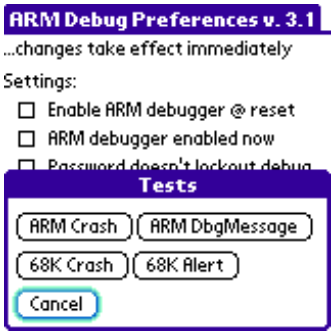
- Log

  Shows the last fatal error dialog box.

  NOTE This is particularly useful because, as mentioned earlier, a Treo smartphone does not display fatal errors when they occur; it simply performs a soft reset.

- Test

  Simulates one of the following errors:



Which debugger nub is currently active?

If you see a blinking block in the lower-left corner of the screen, the 68K PACE debugger nub is active.

If you see a blinking block in the lower-right corner of the screen, the 68K PACE console mode is active.

If you see a blinking line at the top of the screen, the ARM debugger nub is active.

NOTE The PACE debugger nub stops the PACE environment, not the ARM.The ARM debugger nub stops everything and shows the current 68K state as accurately as possible. It might not always be valid, because there is a state for each thread or task.

How to connect to a Treo smartphone for debugging

To switch to debug or console mode:

1. Press Option+Shift+Find to open the Find dialog box.
2. Press S and then Alt.

   The shortcut character appears as an option, usually at the bottom of the screen.
3. Select the shortcut character, enter a period (.), and then press Option-1 or Option-2 to switch to Debug or Console mode, respectively.

Treo smartphone version of the Palm OS® simulator

This section describes the Treo smartphone version of the Palm OS® simulator.

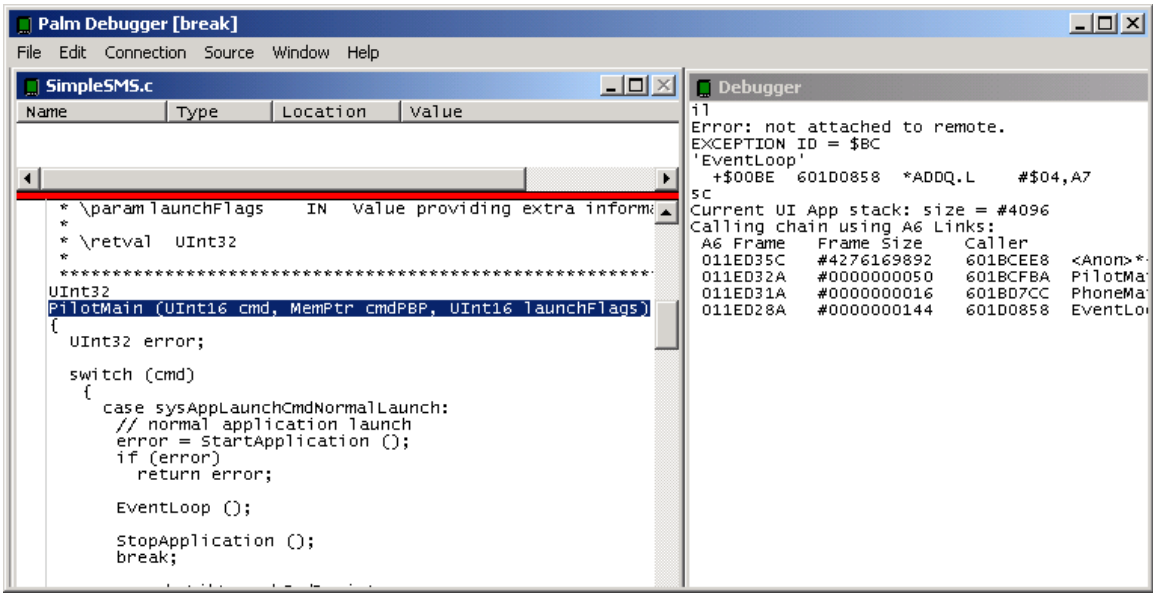
5-way button keystroke equivalents

The 5-way button keystrokes are simulated in the Treo smartphone version of the Palm OS simulator as follows:

```
Right-Shift             Option
Left-Shift              Shift
Right-Ctrl              Menu
Left-Ctrl               various vchrs, see below
Left-Ctrl-A             vchrMenu
Left-Ctrl-B             vchrLowBattery
Left-Ctrl-C             vchrCommand
Left-Ctrl-D             vchrConfirm
Left-Ctrl-E             vchrLaunch
Left-Ctrl-F             vchrKeyboard
Left-Ctrl-I             vchrFind
Left-Ctrl-K             vchrCalc
Left-Ctrl-N             vchrNextField
Left-Ctrl-P             vchrPrevField
Left-Ctrl-L             chrCarriageReturn

Esc                     vchrHardPower (radio power)
F1                      vchrHard1 (Phone)
F2                      vchrHard2 (Calendar)
F3                      vchrHard3 (Messaging)
F4                      vchrHard4 (Screen power)

End                     hsChrSymbol (Alt)
Home                    vchrRockerCenter
Clear                   vchrRockerCenter
Left-Arrow              vchrRockerLeft
Right-Arrow             vchrRockerRight
Up-Arrow                vchrRockerUp/vchrPageUp
                        (depending on focus mode)
Down-Arrow              vchrRockerDown/vchrPageDown
                        (depending on focus mode)

Page-Up                 hsChrVolumeUp
Page-Down               hsChrVolumeDown

F5                      Toggle coordinate display in title bar
F7                      Sticky-shift (toggles press/release
                        of shift key)
F8                      Sticky-Option (toggles press/release
                        of option key)
F9                      Plug-in/Unplug a simulated charger
F11                     Increase simulated battery charge by 1%
F12                     Decrease simulated battery charge by 1%

Left-Ctrl-R             Soft Reset
Shift-Left-Ctrl-R       Hard Reset
Left-Ctrl-S             Power off (simulates down/up of Esc key)
```

Source Level Debugging

Using Palm Debugger



Palm Debugger is the palmOne modified version of the Palm Debugger.exe application. It allows for debugging of 68K code in Palm OS 5. You can get the latest version from the palmOne developer web site.

To debug using Palm Debugger, follow this procedure:

1. Build your project object code to generate symbolic information (filename.sym).

   You can find an example of how to build your project object code using this method in the sample code available on the palmOne developer web site at http://pluggedin.palmone.com. You must compile your code with something resembling the following:

   ```
   m68k-palmos-gcc -O2 -g -Wall SimpleSMS.o -o Obj/SimpleSMS.sym
   ```

2. Do one of the following:

   – To connect to the Palm OS simulator, launch Palm Debugger and select Connection as Emulator, and then launch the Palm OS Simulator.

   – To connect to a Treo smartphone, launch Palm Debugger and select Connection as Serial (USB connection is *not* supported), and then put the Treo smartphone into Debug or Console mode using the method described in "How to connect to a Treo smartphone for debugging."

3. Install the database or databases that represent your application, and load its symbols from the Source menu or by using the F8 hot key.

4. Open or load the appropriate .prc, .sym, and .c files.

   Skip the gdbstub.c file by selecting Cancel.

5. Set any breakpoints you need, and begin your debugging session.

Using Metrowerks Code Warrior

Code Warrior for Palm OS v9 also allows you to do source debugging on a Palm OS simulator or Treo smartphone:

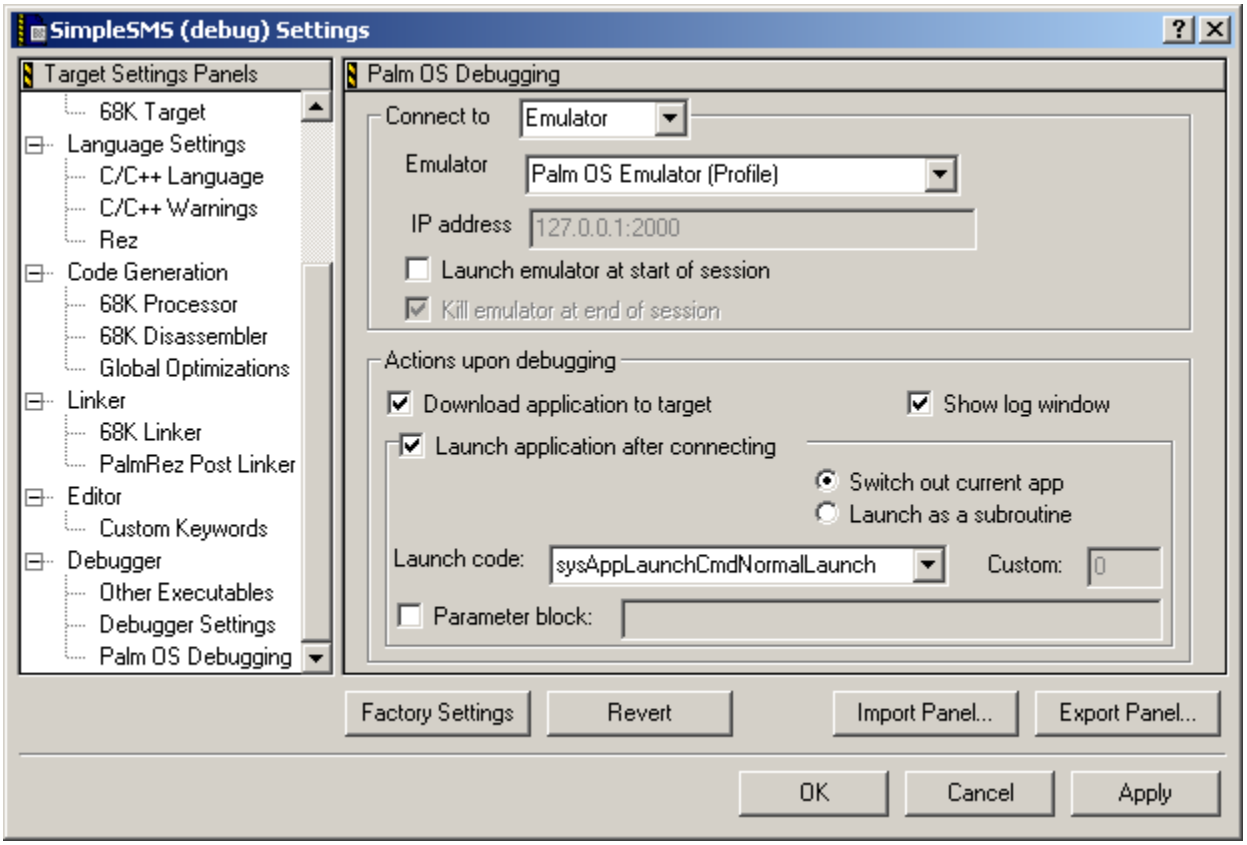
1. Select the project settings to build your project object code with symbolic information:

   Settings -> 68K Linker -> Debugger Info -> Generate SYM File

2. Do one of the following:

   – To debug using the Palm OS Simulator, configure the project settings as follows and launch the Palm OS Simulator.

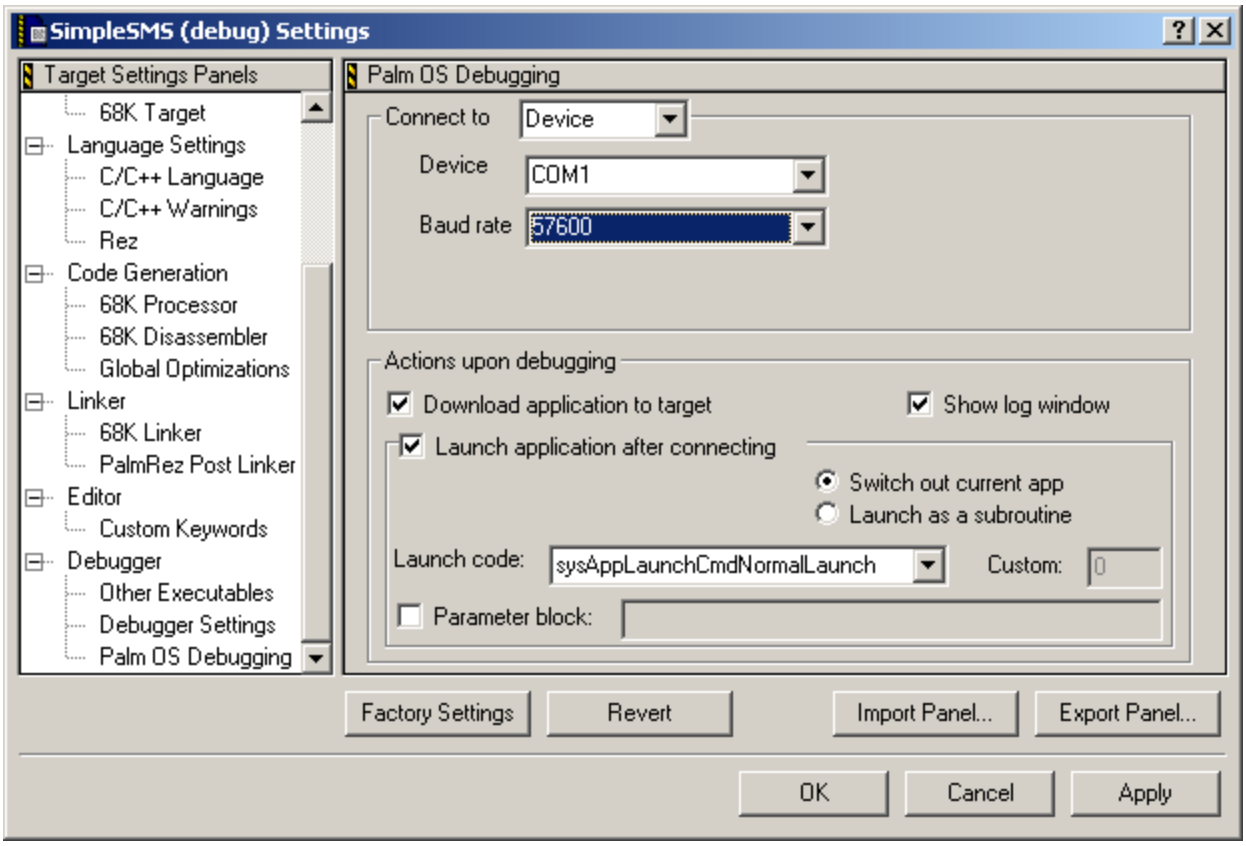
     Settings -> Palm OS Debugging -> Connect to ->Emulator



– To debug on a Treo smartphone, configure the following project settings and place the Treo smartphone into Debug or Console mode using the method described in "How to connect to a Treo smartphone for debugging."

  Settings -> Palm OS Debugging -> Connect to -> Device

  Device -> Com1



TIP Make sure the following are true:
The right COM port and baud rate (57600) are selected.
The COM port is not currently being used by other applications.
The Serial cable is connected to the right COM port.

3. Load your application and its symbols and run it by selecting Source -> menu -> Run (or F5).

   Code Warrior displays a dialog box notifying you that the application is being loaded to the target.

4. Set any breakpoints you need, and begin your debugging session.

PART V

Style Guide

This part of the guide provides guidelines for the “look and feel” of applications that use software components in the SDK.

CHAPTER 11

Style Guide

This chapter outlines the style guidelines that you should follow when designing certain palmOne™ SDK features.

Designing pages for the Blazer® web browser

Available on:

- Treo™ 600 and Treo™ 650 smartphones
- Tungsten™ T5 handhelds

Although the Blazer® web browser's table unrolling technology does a good job of adapting web sites for mobile devices, advance planning can reduce the translation and download time and ensure that results are as expected. This section covers the various factors that web site designers and programmers should take into account when designing web pages.

General rules for web page design

Here are some general rules that apply when designing web sites for mobile devices:

- Make content accessible within one or two links.

  Because the user is typically using a slow, costly connection, it is important that the information the user is trying to access be easily available within one or two navigational moves.

- Keep relevant content and links within the viewable area.
- Personalize and prefill forms whenever possible.
- Use as much screen space as possible without cluttering the screen.
- Keep graphics use to a minimum. When you have to use graphics, keep them simple and small.
- Keep the page size small, preferably under 4KB to 6KB.
- Keep the page simple. Limit the use of JavaScript, and don't use frames or plug-ins. While JavaScript is supported, most JavaScript runs slower on the Blazer web browser than on other handheld browsers due to memory and CPU limitations. For simple JavaScript this is not noticeable, but the use of more elaborate JavaScript may create noticeable delays.

Due to the various restrictions on designing web pages for mobile devices, we recommend that a web site contain a parallel "mobile" version. The full web site can take advantage of all the web technologies, while the mobile site is a slimmed-down version designed to support mobile devices. For example, the palmOne web site contains a page that is designed for mobile devices: http://mobile.palmone.com/ .

Web developers can have their web server automatically load the appropriate page based on the user agent of the browser. For information about the user agent for the Blazer web browser, please see "Browser identification."

Screen resolution

Mobile devices have a screen size that is much smaller than a desktop screen's resolution. On low resolution Treo™ smartphones, the full-screen resolution is 160 x 160 pixels, and the available space in the browser, due to the toolbar and scroll bar dimensions, is 155 x 145 pixels. On high-resolution Treo smartphones, the full screen resolution is 320x320, and on high-resolution Tungsten™ handhelds the resolution is 320x320, 320x480, or 320x480, depending on the current state of the Graffiti area. Blazer 4.0 web browser also has a URL bar that users can choose to display or hide. When it is displayed, it takes space away from the current web page display area. Designers should take these dimensions into account when creating images, tables, lists, and other components of web pages. Developers who want to alter their page based on the current screen resolution should note that the current screen resolution is included in the User Agent. See "Browser identification" for more information.

Connection speed

Users can connect to the Internet at several speeds. The speeds available depend on the type of network the user is accessing and the device they are using to connect (CSD, 1xRTT, GPRS, EDGE). Field-tested speeds can range from 10kbps (CSD dial-up) to over 150kbps (EDGE). Web site designers should keep in mind that different users can connect using a wide range of speeds and should optimize their web pages to load appropriately.

Content

In Optimized mode, the Blazer web browser typically reformats web pages into a small, screen-friendly format. Sites that have a simple layout with minimal tables and graphics have a good chance of being displayed properly after the reformatting. The sections details the factors to keep in mind when designing your web pages.

Page titles

The Blazer web browser does not display web page titles in web page view. To view the title of a web page, users must open the Page Properties dialog box, as shown in the following figure. The page title is used to prepopulate the bookmark description or title field when adding a bookmark.

Content optimized for the Blazer web browser

You can also insert the following tag into the HEAD section of a web page to minimize the amount of reformatting that the Blazer web browser does:

```
<META name="HandheldFriendly" content="True">
```

This tag tells the Blazer web browser to render tables without any special reformatting. If this tag is not present, tables may be unrolled or reformatted. Also, if this tag is present, Blazer 4.0 displays the content in one pass instead of the usual two-pass rendering method.

Embedded Audio Playback

Blazer 3.0 and 4.0 web browser support embedded audio playback using the `<OBJECT>` tag. The `<BGSOUND>` and `<EMBED>` tags are not supported. For instance, `"<object data="oscar.mid" type="audio/midi" width="0" height="0" ></object>"` loads and plays the MIDI file oscar.mid while the current page is displayed.

All devices with Blazer 3.0 or Blazer 4.0 loaded in ROM support embedded MIDI. Some devices may also support embedded AMR and/or embedded QCELP.

File upload

Blazer 4.0 web browser has limited file upload support. Blazer 4.0 only supports uploading files from an expansion card; it does not currently support any uploading from Palm OS main memory. When Blazer 4.0 is rendering a page with a form containing the file upload tag, Blazer displays a single-line text field. To upload a file, the user must type in the fully-qualified path to the file on the expansion card to be uploaded. (In the future, a Browse button may be added that allows the user to browse for the file to be uploaded.)

mailto command

The Blazer web browser does support the `mailto` command to send e-mail.

Typically on desktop systems, the web browser launches an e-mail application to send mail when a `mailto` command is encountered. On devices, it is not guaranteed that a mail application is installed. If an e-mail application is installed, a `mailto` command launches the default e-mail application as defined in the Defaults panel in Prefs.

For dialing telephone numbers, see the `tel:` and `phoneto:` tag discussions in the "Palm OS integration tags" section.

Multipass rendering

Unless the `HandheldFriendly` tag is present, Blazer 4.0 web browser renders each page in two passes. In the first pass, all the text on the page is displayed. In the second pass, images are downloaded and JavaScript is downloaded and executed. By making the text immediately available to the user, the site appears more responsive and the user can read content and select links before the entire page has downloaded. There are several things a content developer can do to ensure that a page is more usable during the first pass:

- Minimize the use of images, especially images with text.

  Because images are not displayed until the second pass, if a site's content is image-based the user has to wait longer before interacting with the page.

- When images are used, include descriptive ALT tags.

  During the first pass the text of the ALT tag is displayed in place of the image.

- Do not use JavaScript in links.

  Usually, when a user selects a link during the first pass, the link is immediately activated. If the link includes JavaScript (and JavaScript is enabled), the browser can not follow the link until the second pass because the JavaScript engine is not available until the second pass is complete. Thus, if the user clicks on a link with JavaScript during the first pass, an error message is displayed and the link is not activated.

Forms

The Blazer web browser supports standard HTML and WML forms, including text boxes, radio buttons, check boxes, text inputs, select lists, multiple selects, and drop-down menus. However, there are some guidelines to follow when you design forms for mobile devices:

- Make sure that the form's open and close tags (`<form>` and `</form>`) are not contained inside a table. Form input may safely be placed inside table cells.
- Text input is supported, but the maximum length of the text is the length of the text input dialog box.

Tables

In Optimized mode, the Blazer web browser supports tables by reformatting or unrolling the table so that it fits on the small screen of a mobile device. The table-rendering engine is optimized for displaying simple text-based tables. Web page designers should avoid using one-pixel spacer images for precise content control, because the images and subsequent reformatting may not look ideal on a mobile device.

Table width attributes are not supported. However, table cell width attributes are supported. To widen tables, change the width attribute of the table cells. To preserve the width on the client display, use the `HandheldFriendly` tag. For more information, see "Content optimized for the Blazer web browser."

For example, the following HTML code displays a table with a table width of 140:

```
<table border="1">
<tr>
   <td width="140" colspan=5>abcdef</td>
</tr>
<tr>
   <td>1</td>
   <td>2</td>
   <td>3</td>
   <td>4</td>
   <td>5</td>
```

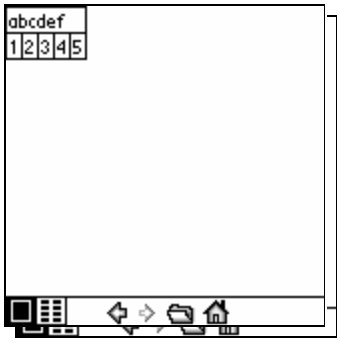
```
</tr>
</table>
```

The table appears on a desktop web browser as follows:

a b c d e f

1 2 3 4 5

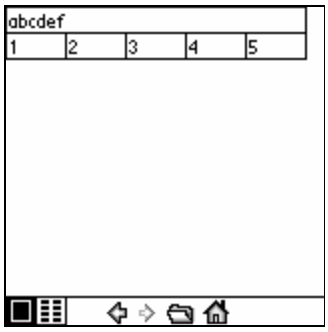
This same HTML code is displayed in the Blazer web browser without the extra spacing if the `HandheldFriendly` tag is not included:



To have the Blazer web browser display the table as intended, include the `HandheldFriendly` tag within the HEAD section of the web page, as follows:

```
<META name="HandheldFriendly" content="True">
```

This causes the device to display the table with the width tag enabled, as shown in the following figure:



Images

Using images on a web site can enhance the presentation of content and the user experience. At the same time, the use of images slows down the web browsing experience. Therefore, it's important to carefully consider which images to display on mobile devices.

All images pass through an imaging processor in the Blazer web browser client that scales down any image that exceeds the current screen size. The bit-depth of the image is also adjusted to match the device's image display settings.

When designing images for mobile devices, keep the following in mind:

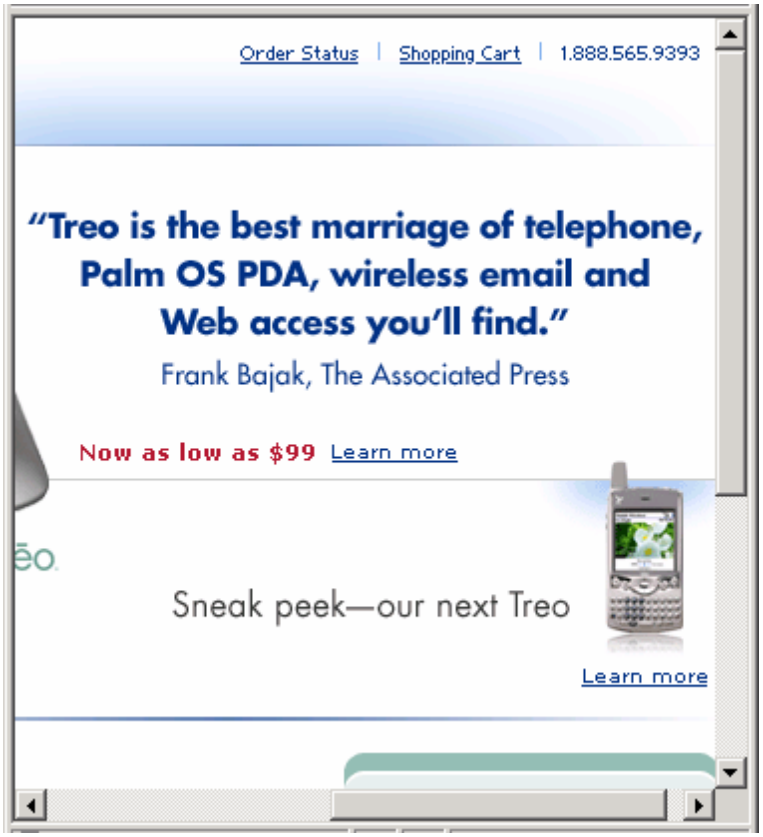
- Image size should be very small (under 4KB).
- Use only a few graphics per page to reduce the load time.
- Use images that are high contrast for easy readability.
- Use ALT tags to display text content while images download.

  Here is what is displayed while the image is loading:

  And here is what appears after the image has finished loading:

Images with text

Avoid using images in place of text. Unnecessary images add to the total download time for each page. If the Blazer web browser needs to scale the image, the text may become unreadable. The following images show how a text image can look fine on a desktop browser but can become unreadable when scaled and displayed in the Blazer web browser.



Horizontal header images

Many web sites use a horizontal header image for navigation purposes. Typically, this image consists of many smaller images that are formatted in a table with zero-length borders.

On a desktop browser, such an image is displayed appropriately. The Blazer web browser, however, typically reformats such a page so that the images are stacked vertically. The following images demonstrate this behavior.

Header images on a desktop browser:

The same images on the Blazer web browser:

You can clean up the web page by removing spacer images and inserting the `HandheldFriendly` tag at the beginning of the page. The next figure shows the updated page without spacer images in the table. The second figure shows the page in the Blazer web browser without the `HandheldFriendly` tag. Because the spacer images are not present, the browser displays the image in a more readable format. The third figure shows the same page with the `HandheldFriendly` tag present. This tag instructs the Blazer web browser not to reformat the tables. As a result, the table is displayed as intended. The Blazer web browser provides horizontal scroll bars to allow the user to view the entire image.

Supported image formats

The Blazer 3.0 web browser supports the following image formats:

- GIF
- JPEG
- PNG
- Animated GIF
- WBMP
- BMP (Blazer 4.0 web browser)

Unsupported content

While the Blazer web browser supports most of the HTML 3.2 specification, there are certain unsupported elements. Some of the additional web technologies that are not supported in the Blazer web browser include:

- Java applets
- WMLScript
- Animations (Macromedia Flash)

- Audio, although the Blazer web browser can download an audio file to be played by an application that can handle the audio format
- Browser Plug-Ins

When the Blazer web browser encounters an unsupported element, it ignores the associated code when displaying the web page. In most cases, the user experiences the web page with reduced functionality. If the web site requires the unsupported technologies to view the information, the Blazer web browser users will not be able to use the web site.

Working with the Blazer web browser

This section discusses topics that are specific to the Blazer web browser and Palm OS. Web site designers and you should make note of these issues when implementing web pages that you have optimized for use with the Blazer web browser.

Palm OS integration tags

Blazer Web Browser 3.0 and 4.0 do not support date picker and time picker. Support may be added in a later release of the Blazer web browser.

Browser identification

As part of communication to a web server, web browsers send out a string indicating what type of browser is accessing the server. This is referred to as the user agent.

The the Blazer 3.0 web browser user agent string is:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 95; PalmSource;
Blazer 3.0) 16;160x160
```

The Blazer 4.0 web browser user agent string is:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98; PalmSource/hspr-
H102; Blazer/4.0) 16;320x320
```

`hspr-H102` indicates which palmOne device the browser is running on; different devices have different strings. Also, the current screen size is included, so if the user changes the state of the Graffiti area, the user agent string is updated to match the screen size. For these reasons, you should not search for an exact match when checking for the presence of the Blazer 4.0 web browser. Instead, simply check to see if the user agent string contains the substring `Blazer/4.0`.

Web developers can use this header to send the version of a web page optimized for display by the Blazer web browser.

Cookies

The Blazer web browser contains support for cookies. Typically, web site developers use cookies to store a small amount of information about a user's ID or profile, web site personalization, and so forth. As with any web browser, users can

have the Blazer web browser not accept cookies or clear out the cookie store. Cookies are stored on the local Palm OS device.

The Blazer web browser contains full support for the `SET-COOKIE` header. The only restriction is that the value attribute must not exceed 2000 bytes. Cookies with a value attribute larger than 2000 bytes may be rejected. Other types of cookie headers are not supported.

If a cookie has an expiration date, then it is a persistent cookie. If there is no expiration date, it is a session cookie and the cookie is purged when the session is terminated.

Session handling

On a desktop browser, a session cookie persists until the user quits the web browser. On a Palm OS® device, only one application can be active at any given time. To take into account that a user may temporarily switch to another application while using the Blazer web browser, the system ends a session 20 minutes after the Blazer web browser exits. (This time may vary based on the wireless service provider.)

Security

The Blazer web browser uses the standard Palm OS security libraries—the same libraries that are shipped with every version of Palm OS 5.2.1 and later. The Blazer web browser incorporates 128-bit SSL 3.0 encryption technology to ensure that visiting any site with a device is as secure as browsing from a desktop. The 128-bit encryption uses the RC4 algorithm and RSA-based key exchange. It also incorporates RSA-based digital signature verification, MD5 and SHA-1 secure hash algorithms, support for X.509 certificates, and an intuitive indication of a secure connection by a lock icon in the toolbar or URL bar.

Caching

The Blazer web browser includes a web page cache to improve performance. If a page requires a regular refresh, or should not be cached, the web server must send the appropriate standard HTTP cache control headers in the HTTP response.

Downloads

Blazer 4.0 web browser limits downloads to 2MB.

Testing your web site

Designing your web pages will most likely be an iterative process. We recommend that you test your web sites early and frequently to ensure that you are able to achieve the intended design and layout. The following sections contain a few pointers to help you successfully test your web site's content.

Multiple devices

If you intend your content to be accessible through a variety of devices, be sure to test your web site with as many of those devices as possible during the

development process. Areas you should look at in particular are device memory and screen size.

Refreshing content

Because the Blazer web browser uses cached pages on a device to provide quick access to frequently viewed content, constant updates to a web page during development may not be visible on the device. We recommend that you set the cache size to zero during testing. During development, you should use the Clear button in Preferences to clear out the cache and use the Page -> Refresh menu command to make sure the current page is loaded. Images are typically refreshed when the page is refreshed. If images are not being updated in the Blazer web browser, the cache should be cleared out or the image filename should be changed.

International support

The Blazer web browser client supports the Palm OS character set. This character set is similar to the ISO-8859-1 character set. The Palm OS character set allows support for most Western European characters. The Blazer web browser application has also been localized into the following European languages:

- English
- French
- Italian
- German
- Spanish

HTML encoding

The Blazer web browser supports web pages that are encoded using the ISO-8859-1 and UTF-8 character formats. This allows Western European languages to be displayed properly on the Blazer web browser client. Web pages that are encoded in UTF-8 must indicate so by including one of the following tags:

Content-type HTTP Header:

```
Content-Type: text/html; charset=utf-8
```

HTTP-equiv META Tag:

```
<META http-equiv="content-type" content="text/html; charset=utf-8"\
```

WML Encoding

WML content is typically encoded in UTF-8 format. WML pages that are encoded in the ISO-8859-1 format must indicate so by including one of the following tags:

Content-type HTTP Header:

```
Content-Type: text/vnd.wap.wml; charset=iso-8859-1
```

HTTP-equiv META Tag:

```
<META http-equiv="content-type" content="text/vnd.wap.wml; charset=iso-
8859-1"\
```

XML Tag:

```
<?xml version="1.0" encoding="iso-8859-1"\
```

Accept headers

The Blazer web browser client supports Accept HTTP headers. The client sends this header to the web server based on the languages the client supports. You can use this header to determine what type of content to deliver to the device.

The Blazer Web Browser 3.0 Accept headers for an English ROM appear as follows:

```
Accept: text/html, application/vnd.wap.xhtml+xml, application/xhtml+xml;
profile="http://www.wapforum.org/xhtml", image/gif, image/jpeg, image/
pjpeg, */*
Accept-Language: en, *;q=0.8
```

The Blazer 4.0 web browser Accept header is similar to the Blazer 3.0 header, except that it also includes every type currently registered with the Palm OS Exchange Manager.

(The Accept-Language header varies based on the current language selected upon setup of the device.)

```
Accept-Encoding: deflate, gzip
```

List of Acronyms

The following table lists web browser and internet acronyms.

Acronym	Definition
cHTML	Compact HTML. A subset of HTML for mobile devices. Primarily used in i-Mode devices.
ECMA	European Computer Manufacturers Association.
GIF	Graphics Interchange Format.
HDML	Handheld Device Markup Language. An old format used for web-enabled phones. This is no longer used.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
ISO	International Organization for Standards. The name is derived from the Greek word *iso*, meaning “equal.”
JPEG	Joint Photographic Experts Group.
PNG	Portable Network Graphics.

Acronym	Definition
TCP/IP	Transmission Control Protocol/Internet Protocol.
UCS	Universal Character Set.
UTF	UCS Transformation Format.
WAP	Wireless Application Protocol.
WBMP	Wireless Bitmap. A graphic format optimized for wireless devices.
WML	Wireless Markup Language.
XHTML	Extensible HTML. A cross between HTML and XML.
XML	Extensible Markup Language.

Palm OS Integration Tags

The Blazer web browser supports several tag attributes that extend HTML support for Palm OS devices, as shown in the following table.

Keyword	Description
HandheldFriendly	This attribute for the META tag tells the proxy server that the web page has been specifically designed for small screens. The proxy server tries to render the tables as close to specification as possible. Example: `<META name="HandheldFriendly" content="True">`
Palm™	This keyword is used in the HREF attribute to launch a specific application on the device. Blazer web browser exits and the specified application becomes the active application. Example: `<A HREF="palm:memo.appl">Memo Pad</A>`
Palmcall	This keyword is similar to the Palm keyword except that Blazer web browser sublaunches the specified application. Once the application exits, the user returns to Blazer web browser. For example: `<A HREF="palmcall:flip.appl">Flipper</A>` Parameters can also be passed into the application using the `sysAppLaunchCmdURLParams` launch code. For more information on this and the `palm` and `palmcall` tags, please refer to the following tutorial on the Palm OS Developer web site: www.palmos.com/dev/tech/webclipping/tutorials/tutorial_palm.html **TLI-Is this link still valid?**
phoneto: and tel:	These keywords are equivalent and are used in the HREF attribute to dial a specific phone number. Blazer web browser exits and the phone application becomes the active application when these keywords are used. Examples: `<A HREF="tel:555-1212">Jenny</A> and <A HREF="phoneto:588-2300">Empire</A>`

Keyword	Description

file:	This keyword is used in the `HREF` attribute, in the URL bar, and in the Open URL dialog box to access browser content stored locally on an expansion card. A network connection is not required when accessing content using `file:///`.
	Note that the syntax always includes three slashes after `file:`
	Blazer 3.0 web browser supports the following syntax:
	`file://dir1/dir2/file.txt`
	In this example, Blazer 3.0 web browser would attempt to open the file file.txt located on the expansion card in directory /dir1/dir2.
	In addition to the Blazer 3.0 syntax, Blazer 4.0 web browser includes support for an additional syntax that handles devices with multiple expansion cards better, such as the Tungsten T5 handheld. T
	An example of the additional syntax is:
	`file:///volname/dir1/dir2/file.txt`
	In this example, Blazer 4.0 would locate the expansion card named volname, and then open file file.txt in directory /dir1/dir2 on that expansion card.

Gadgets

In the Phone application on Treo smartphones, there is a fixed position for the system gadgets. For third-party applications to be consistent with the Phone application, they should position the system gadgets in the same location. We have included suggestions for placement of the gadgets based on the Phone application.

Required headers and libraries

The following headers and libraries are required to use the gadgets:

- `PmSysGadgetLibrary`
  - `68K/Libraries/PmSysGadgetLib/PmSysGadgetLib.h`
  - `Common/Libraries/PmSysGadgetLib/PmSysGadgetLibCommon.h`

  This is the library that contains the system Battery, Signal, and Bluetooth gadget implementation.

How to include the Battery gadget

To include the Treo smartphone Battery gadget in an application form, do the following:

1. Add a gadget to the application form at position (304, 0) for 320 x 320 resolution for a Treo 650 smartphone. Use half these dimensions for a Treo 600 smartphone.

   Because the system determines the height and width of the gadget, the width and height specified in the rcp/rsrc file will not matter.

2. In the application form event handler for `frmOpenEvent`, add the following code:

```
PmSysGadgetStatusGadgetTypeSet ( gPmSysGadgetLibRefNum
                                 frmP,
                                 <YOUR_BATTERY_GADGET_ID>,
                                 pmSysGadgetStatusGadgetBattery);
```

Updates and events associated with the Battery gadget are handled automatically.

How to include the Signal gadget

To include the Treo 650 smartphone Signal gadget in an application form, do the following:

1. Add a gadget to the application form at position (278, 0) for 320 x 320 resolution for a Treo 650 smartphone. Use half these dimensions for a Treo 600 smartphone. Because the system determines the height and width of the gadget, the width and height specified in the rcp/rsrc file will not matter.
2. In the application form event handler for `frmOpenEvent`, add the following code:

```
PmSysGadgetStatusGadgetTypeSet ( gPmSysGadgetLibRefNum
                                 frmP,
                                 <YOUR_SIGNAL_GADGET_ID>,
                                 pmSysGadgetStatusGadgetSignal);
```

Updates and events associated with the Signal gadget are handled automatically.

How to include the Bluetooth® gadget

To include the Treo 650 smartphone Bluetooth® gadget in an application form, do the following. (Note that the Treo 600 smartphone does not include Bluetooth technology):

1. Add a gadget to the application form at position (260, 0) for 320 x 320 resolution for a Treo 650 smartphone. Use half these dimensions for a Treo 600 smartphone. Because the system determines the height and width of the gadget, the width and height specified in the rcp/rsrc file will not matter.
2. In the application form event handler for `frmOpenEvent`, add the following code:

```
PmSysGadgetStatusGadgetTypeSet ( gPmSysGadgetLibRefNum
                                 frmP,
                                 <YOUR_BLUETOOTH_GADGET_ID>,
                                 pmSysGadgetStatusGadgetBt);
```

Updates and events associated with the Bluetooth gadget are handled automatically.

Index

F

G

I