API Manual

Software Development Kit

80-B6006-1 Rev. -

QUALCOMM Incorporated
6455 Lusk Blvd.
San Diego, CA 92121-2779

80-B6006-1 Rev. -
Document Version 1.0
April 2, 1999

Contents

About This Document

Welcome to the pdQ™ API Manual. The purpose of this document is to describe the Application Program Interface (API) for the pdQ smartphone phone. It is written for developers interested in writing Palm applications for the pdQ smartphone phone. Knowledge of how to write a Palm application is assumed. *Developing Palm OS 3.0 Applications,* parts I-III, is a useful reference and can be found at the following Web site: http://www.palm.com/devzone.

About the pdQ SDK

The pdQ SDK consists of the following:

- API Manual (this document)
- Header files
- Sample code

Requirements

The following are required for development of an application for the pdQ smartphone:

- The pdQ SDK, as defined above.
- A development environment.
- A pdQ smartphone, Palm III, or emulator.

The pdQ smartphone development team uses CodeWarrior Release 5 for Palm™ OS. Other development environments, such as gcc, should work. Please report development environment problems by writing to the following e-mail address: pdQDeveloper@qualcomm.com. Developers can also discuss development issues on the unmoderated mailing list, pdQDevTalk. Visit http://www.qualcomm.com/pdQ/developer.html for more information.

Developers who do not have access to a pdQ smartphone or to CDMA coverage may wish to begin development by stubbing out library calls for the pdQ smartphone and simulating launch codes.

About the pdQ Smartphone

About the Operating System

The pdQ smartphone incorporates the 3.0 version of Palm OS. All functions of this operating system and version are supported. Thus, all applications that are compatible with the Palm Computing® platform including, applications that take advantage of Palm's HotSync® technology, access the TCP/IP stack, and are IrDA-enabled, are supported and should function exactly as they do on other Palm Computing platform devices.

About the Phone

The pdQ smartphone is a CDMA digital wireless phone. It will be available in two models: a digital PCS pdQ smartphone (1900 MHz), and a dual mode (CDMA/Analog) cellular pdQ smartphone (800 MHz). Developers interested in learning more about CDMA digital technology can find information at QUALCOMM's web site: http://www.qualcomm.com

Hardware

Memory

The pdQ smartphone contains 2MB RAM. Memory expansions are not supported.

Similarities To Palm Computing Connected Organizers

The pdQ smartphone includes all the familiar Palm Computing platform interfaces. This includes a silk-screened Graffiti® handwriting recognition area with four silk-screened icons, four integrated hard buttons, up and down scroll keys, and an IrDA port. The stylus is housed in a slide slot. Figure 1 shows the pdQ smartphone interfaces.

Differences With Palm Computing Connected Organizers

The most obvious physical difference between the pdQ smartphone and other Palm Computing devices is the addition of a phone keypad flip. With the keypad flip in an upright ("closed") position as shown in Figure 2, the pdQ smartphone functions like a traditional wireless phone. Third party applications can be "sub-launched" (see the following pages) while the flip is closed, but there will be no display of any third-party application user interface. For example, third party applications may have awareness of incoming alerts and may perform some management functions on them, but the screen display will not be available to that application until the keypad flip is opened.

When the keypad flip is opened, users are presented with the familiar Palm Computing platform interface with minor differences. A silk-screened strip down the right side of the screen contains five additional rectangular icon areas. These icons have default settings as follows (from top to bottom): contrast adjust, turn phone on/off, call history, launch pdQmail™, and enable HotSync. Of these, users may assign, through the Preferences option, different functionality for the icons devoted to call history and pdQmail™.

The top right icon in the silk-screened Graffiti area (which has the default setting to launch the calculator on other Palm Computing devices) has been replaced with this image.

The default setting for this icon will be to launch the Dialer application, which permits users to tap on the screen to dial a phone number.

Standard Applications

All standard Palm applications are included in the pdQ smartphone and function exactly as they do on other Palm Computing platform devices with the following exceptions:

Address Book – The Address Book has been modified to permit direct launch of the telephone dial dialog, pdQmail™, pdQbrowser™, or other registered applications.

Preferences – Preferences settings have been added to support some of the additional functionality of the pdQ smartphone.

In addition to the other standard Palm Computing applications, the following applications will be embedded in the ROM of the pdQ smartphone:

Call History – An application that logs and permits the management of a user's call log.

Speed Dial – An application that permits users to assign shortcuts to frequently accessed numbers.

pdQmail™ – A full-featured, over-the-air enabled email client application.

pdQbrowser™ – A full-featured over-the-air enabled HTML pdQ smartphone browser.

pdQalert™ – An application that enables users to manage and view incoming voice and text alert messages (SMS types), and launch applications associated with those alerts.

Dialer – An application that permits the user to input a telephone number and initiate a call.

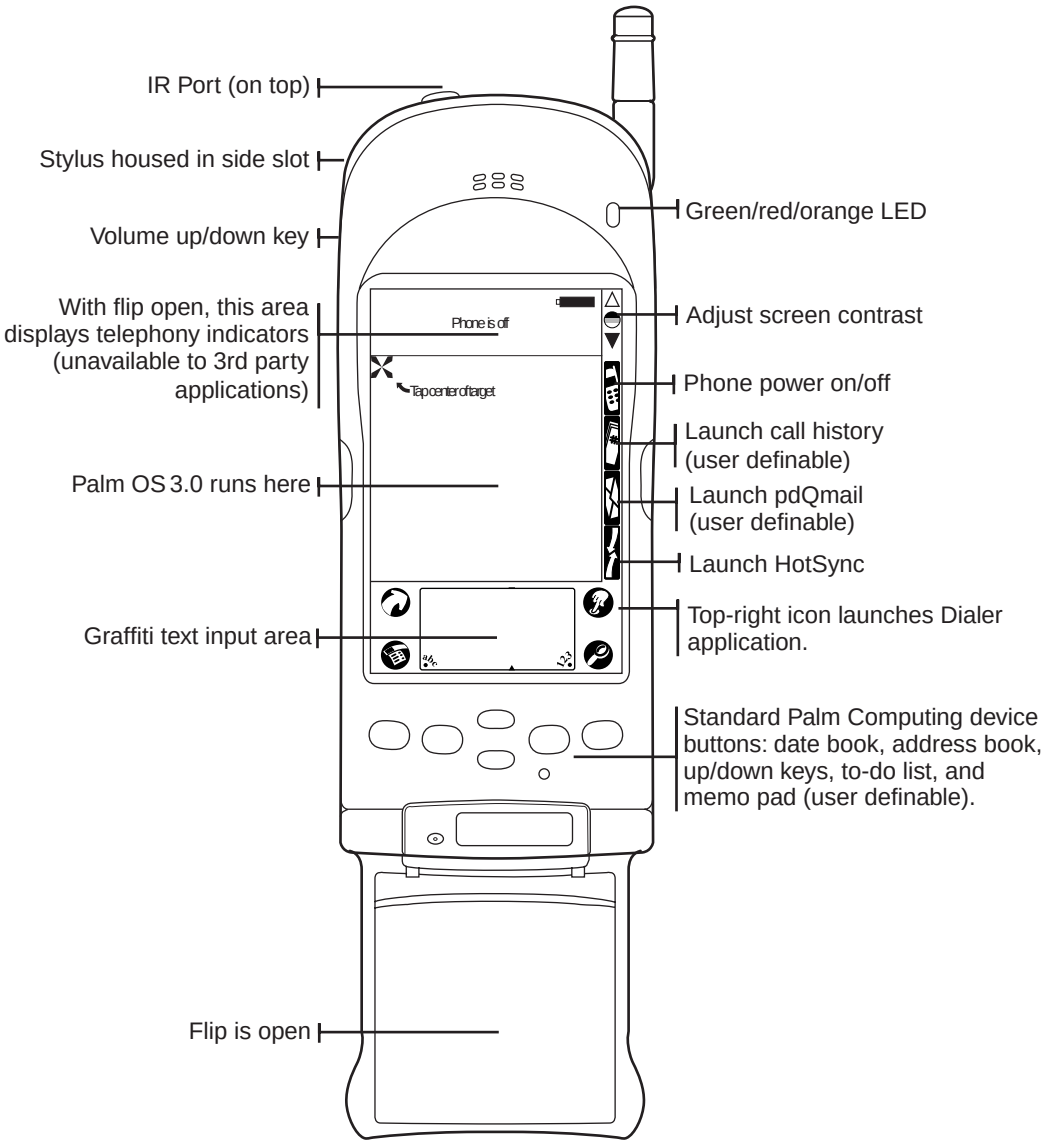

Figure 1 Flip Open

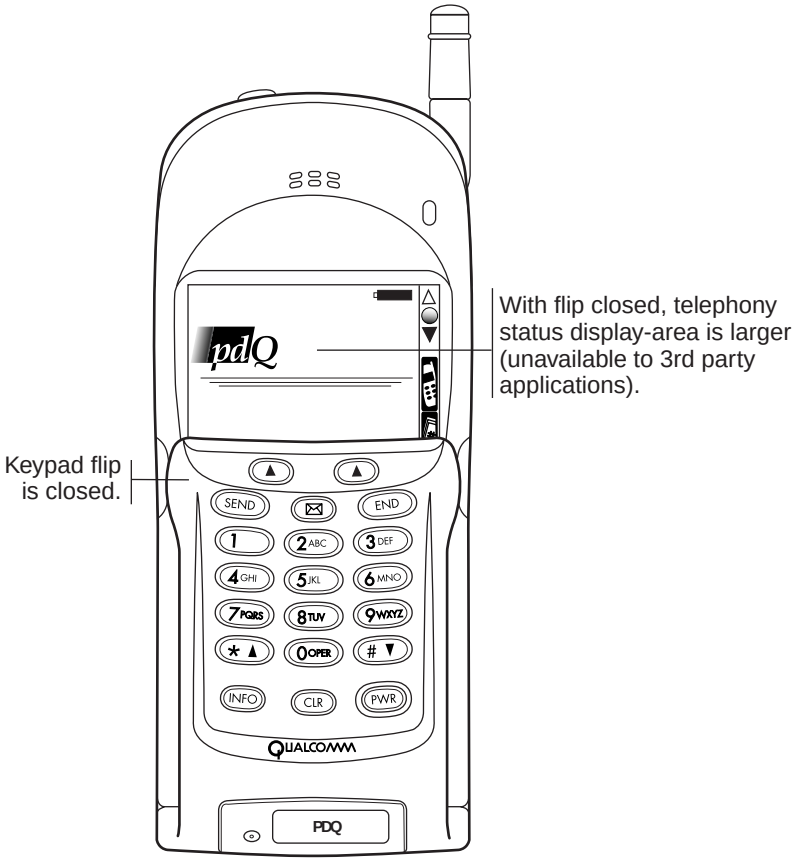

Figure 2. With Flip Closed

CDMA Data

The pdQ smartphone's native TCP/IP stack is enabled with CDMA wireless data interfaces. This means that in CDMA data-enabled markets, the pdQ smartphone functions just like a Palm Computing device with a wireless modem. This functionality is available only in markets enabled with CDMA data by the service provider.

CDMA wireless data currently supports a rate of 14.4 kbps. The following CDMA wireless data capabilities are enabled on the pdQ smartphone:

IS-99 (Async/Fax)

This permits the pdQ smartphone to function like a wireless data and fax modem. With the appropriate settings in Network Preferences, a dial-up session to a corporate or ISP modem may be initiated from an application.

Quick Net Connect

This is the default data service or Internet connection. It establishes a PPP connection to the PCS service provider within 3-5 seconds. Once connected, the application has access

via TCP/IP to hosts and services on the Internet. Configuration parameters, enabling or disabling, are set via a Network Preferences setting.

Note: The availability of these and other CDMA data implementations on commercial systems is entirely dependent upon individual cellular and PCS service providers. QUALCOMM does not provide any information on these markets or their implementation schedules. Developers with questions about when and how a particular market will be enabled for CDMA data are directed to contact the service provider in the appropriate market.

Getting Started

User Interface Guidelines

The pdQ smartphone is a Palm Computing platform device. As such, developers are encouraged to adhere to Palm's user interface guidelines. To find out more about certifying an application through Palm, visit http://www.palm.com/devzone. In addition to Palm's guidelines, phone use should be factored into the application design. Developers should consider:

- Airtime Cost - Carriers typically charge per minute. Applications should avoid making or extending calls without the user's knowledge.
- Etiquette – Activity that disrupts voice conversations should be avoided. For instance, applications should not end phone calls or play DTMFs without the user's consent.

The pdQ Phone Libraries

The pdQ API uses standard Palm shared libraries. The three libraries are shown in Table 1.

Table 1. Summary of pdQ Libraries

Library	Name	Functions
PdQ Core	PDQCoreLibName	Signaling Telephony
PdQ Registry	PDQRegistryLibName	URL support Macro support
PdQ Alert	PDQAlertLibName	LED control Backlight control

The pdQCore library contains the signaling and telephony API. Applications that wish to communicate with one another or to be notified of state changes should use the signaling API. Applications that need to control and access the phone should use the telephony API.

The pdQRegistry library supports URLs and macros. An application can register as a handler for a URL scheme or process a URL. Applications may also want to create or list macros. A macro is simply a URL with a name and description.

The pdQalert library provides access to the LED and backlight.

How to Use the Libraries

In order to call a library function, an application must do the following:

- Include the appropriate pdQ header file (pdqCore.h, pdqRegistry.h, pdqAlert.h)
- Retrieve a valid reference number to the library using Palm's SysLibFind function with the library name.
- Use this reference number as the first parameter to every library function call.

See the code snippet in Appendix A for an example of how to use a library. Palm's white paper Shared Libraries and Other Advanced Project Types, which can be found on their developer pdQ smartphone site http://www.palm.com/devzone provides some good background information on shared libraries.

Note: The library reference number is not valid across hard resets. pdQCore, pdQRegistry, and pdQalert libraries are stored in ROM and are automatically loaded on startup. Do not call SysLibInstall, SysLibLoad, or SysLibRemove on any of these libraries. Do not call the open, close, sleep or wake library traps.

pdQ Signal API

This section introduces the pdQ smartphone signal API. Developers should use this API for applications that need to communicate with one another or to receive notification of pdQ smartphone state changes. For example, an application that wants to be notified whenever there is an incoming call will need to use the pdQ smartphone signal API.

Signaling is a generic inter-application communication mechanism. A signal is an event, much like a Palm OS event, that typically generates when a state changes. Applications dynamically register or unregister for one or more signals. When a signal is sent, all applications registered for the signal are immediately notified by being sublaunched. Developers might want to think of this as analogous to setting up a callback function.

Signals of similar function are grouped into a signal class. Applications can register for specific signals or for an entire class. To register, an application calls **PDQSigRegister** along with a signal class and a mask of one or more signal codes. When an application no longer cares about a signal, it calls **PDQSigUnregister**. An application registers at anytime, but if it wants to be signaled at start up, a good place to do this is when it receives the sysAppLaunchCmdSystemReset launch code.

The signal mechanism is a sublaunch. Applications should handle the **sysAppLaunchCmd_PDQSignal** launch code within PilotMain. See the sample code snippet in Appendix A.

Note: Applications should check the sysAppLaunchFlagSubCall launch flag before accessing globals. See the section, Application Control Flow (pp. 66-84) in *Developing Palm OS 3.0 Applications, Part 1* for more information.

Applications are also free to create and send their own signals. To create a signal class, an application should call **PDQSigAddClass**. To send a signal, an application calls **PDQSignal**. An application can also determine what other applications are registered for one or more signals by calling **PDQSigEnumerate**.

The following declarations and function prototypes are be found in the header file pdqCore.h.

Summary of Signal Functions

PDQCoreLibGetVersion

PDQSigRegister

PDQSigUnregister

PDQSigEnumerate

PDQSignal

PDQSigAddClass

Description of Signals

SGN_CLASS_TAPI Signal Class

This signal class contains signals relevant to the phone's major calling states. Listed below are the data structures used with these signals.

```
typedef enum {
   CT_NORMAL = 0,                  // Two parties
   CT_CONFERENCE,                  // Conference call
   CT_WAITING,                     // Call waiting
   CT_DATA                         // Data call.
} CallFlagsType;
typedef enum {
   PI_ALLOWED = 0,
   PI_RESTRICTED,
   PI_NOTAVAILABLE,
   PI_RESERVED
} CallerIDStatusType;
typedef struct _CallInfoType {
   Dword       dwSignalHist;            // History of all bits set on
call
   CallFlagsType callFlags;             // Data/Voice, etc.
   CallerIDStatusType nCallerIDStatus;       // Reserved, restricted,
unavailable...
   Boolean     bDialingPaused;              // Set to TRUE when we
hard-pause
   Char        szNumber[MAX_DIGITS + 1];     // Dialed or Incoming
phone number
   Ulong       lTimeOfCall;        // When the call began (in seconds)
   Ulong       lCallDuration;           // How long call lasted (in
seconds)
   Char        szWaiting[MAX_DIGITS + 1];    // Last call waiting
number (if available)
   Char        szExt[MAX_DIGITS + 1];        // Extension or extra
DTMF codes...
   Err nErr; // Error code set when call fails, etc.
} CallInfoType, *CallInfoPtr;
typedef enum {
   PAUSE_NONE = 0,
```

```
    PAUSE_SOFT,                 // Timed pause
    PAUSE_HARD                  // Hard/Resume pause
} PauseType;
```

Table 2. SGN_CLASS_TAPI Signals

Signal	Parameter	Description
SGN_TAPI_IDLE	NULL	Phone has acquired service and is idle.
SGN_TAPI_INCOMING	CallInfoPtr	A call has just arrived. szNumber may not be available until SGN_TAPI_CALLERID received. Valid callFlags are CT_NORMAL and CT_WAITING.
SGN_TAPI_LOST	CallInfoPtr	The call was lost.
SGN_TAPI_MISSED	CallInfoPtr	The call was not answered.
SGN_TAPI_FAILED	CallInfoPtr	The call failed.
SGN_TAPI_CALLING	CallInfoPtr	Attempting to originate a call.
SGN_TAPI_CONVERSATION	CallInfoPtr	A call has just been established.
SGN_TAPI_ENDED	CallInfoPtr	A call has just ended.
SGN_TAPI_DIALPAUSED	PauseType	Hard pause encountered. Phone is awaiting user release of pause before continuing to generate DTMFs.
SGN_TAPI_CALLERID	CallInfoPtr	Caller ID of incoming or call waiting call.
SGN_TAPI_DIALEDDTMF	szArg	DTMF has just been sent. szArg contains remaining digits.

SGN_CLASS_MSG Signal Class

This signal class contains signals relevant to the phone's short messaging services. The following list shows the data structures used with these signals:

```
typedef enum {
   MSG_NONE,
   MSG_NORMAL,
   MSG_URGENT
} MsgStatusType;
 typedef struct _MsgType {
   UShort                nNumUnread;         // if 0, other params
ignored
   MsgStatusType         status;             // status of latest mesg
   CharPtr               szMessage;
} MsgSigType, * MsgSigPtr;
```

Table 3. SGN_CLASS_MSG Signals

Signal	Parameter	Description
SGN_MSG_RAW_SMS	pVoidArg	An SMS message has just arrived from the base station.
SGN_MSG_CARRIER_VOICE_MAIL	MsgSigPtr	A voice mail message has just arrived.
SGN_MSG_TEXT_MESSAGE	MsgSigPtr	A text message has just arrived.

Function Descriptions

PDQCoreLibGetVersion

Purpose Retrieve the version number of the Core library.

Prototype Err PDQCoreLibGetVersion (UInt refNum, DWordPtr dwVerP)

Parameters ->refNum Reference number of the Core library.

->dwVerP Pointer to version

Result PDQErrNoError

PDQSigRegister

Purpose Register an application for signals.

Prototype Err PDQSigRegister (UInt refNum,
ULong uClass,
ULong uMask,
SigPriorityType nPrio,
ULong uCreator,
ULong uType)

Parameters ->refNum Reference number of the Core library.

->uClass Signal Class

->uMask Mask of signal codes

->nPrio Priority

->uCreator Unique creator ID of application

->uType Creator type (typically sysFileTApplication)

Result PDQErrNoError

PDQErrNoMemory

Comments To register for all the signals in a class, pass SGN_ALL in as uMask. Avoid registering for signals that are never processed. An easy mistake is to pass in an incorrect uCreator. If an application cannot be found when signaled, it will automatically be unregistered.

See also PDQSigUnregister

PDQSigUnregister

Purpose Unregister an application for given signals.

Prototype Err PDQSigUnregister (UInt refNum,
ULong uClass,
ULong uMask,
ULong uCreator,
ULong uType)

Parameters ->refNum Reference number of the Core library.
->uClass Signal Class
->uMask Mask of signal codes
->uCreator Unique creator ID of application
->uType Creator type (typically sysFileTApplication)

Result PDQErrNoError
PDQErrNotRegistered

Comments If the application does not need to be notified of a signal, unregister it. This will improve performance.

See also PDQSigRegister

PDQSigEnumerate

Purpose List all of the registered applications for the given signals.

Prototype Boolean PDQSigEnumerate (UInt refNum,
ULong uClass,
ULong uMask,
CharPtr szName,
VoidPtr pContext,
PFNENUMSIGNALS pCB)

Parameters ->refNum Reference number of the Core library.
->uClass Signal Class (0 if szName used)
->uMask Mask of signal codes
->szName Name of class (ignored if uClass used)
->pContext Context passed to callback
->pCB Callback function

Callback Prototype Boolean (*PFNENUMSIGNALS) (VoidPtr pContext,
ULong uClass,
ULong uMask,
SigPriorityType priority,
ULong uCreator,
ULong uType)

Result True if any registered applications found.

Comments This function searches by class or class name. The callback function gets called for every match. The pContext can be NULL, but will typically be a pointer to some data. To terminate the enumeration, return false in the callback.

PDQSignal

Purpose Send a signal.

Prototype
```
Err PDQSignal (UInt refNum,
               ULong uClass,
               ULong uSig,
               SigParamType params)
```

Parameters

->refNum	Reference number of the Core library.	
->uClass	Signal Class	
->uSig	Signal code	
->params	Signal parameters	

Result PDQErrNoError

PDQErrInvalidSignaling	(Signal must have exactly one bit set.)
PDQErrSignalReentered	(Signal is already being called.)

Comments Make sure the correct parameters are passed in. If the wrong parameters are passed in, the Palm OS can crash.

PDQSigAddClass

Purpose Create a new signal class or find the ID of the class.

Prototype
```
DWord PDQSigAddClass (UInt refNum, CharPtr pszClass)
```

Parameters

->refNum	Reference number of the Core library.
->pszClass	Class name

Result Unique class ID

Comments Classes (and associated IDs) created with this call are not persistent across hard resets.

pdQ Telephony API

Applications can use the telephony functions to control and access pdQ smartphone capabilities. To originate a voice call, an application can call **PDQTelMakeCall**. This function is capable of parsing a long string of digits in order to originate a call, pause, and continue sending DTMFs. To end a call, an application should call **PDQTelEndCall**. An application can answer an incoming call by calling **PDQTelAnswerCall**.

When the phone is in a voice call, an application can generate a DTMF via **PDQGenerateDTMF**. Depending upon the capabilities of your carrier's service, DTMFs can be used to answer incoming call waiting calls and establish three-way conference calls. They may also be used to navigate through PBX systems.

An application queries the phone to get the current phone state by using **PDQTelGetCallInfo**. Applications may want to use this function instead of registering for phone signals.

When **PDQTelMakeCall** encounters a hard pause, an application (which can be registered for the signal **SGN_TAPI_DIALPAUSED**), can resume the dialing via **PDQTelResumeDialing**, or cancel the pause and dialing of the remaining digits via **PDQTelCancelPause**.

This library also contains two utility functions. Use **PDQTelFormatNumber** to format a string for comparison or display. Use **PDQTelGetDigit** to convert a character to its keypad equivalent.

The following function prototypes are found in the header file pdqCore.h.

Summary of Functions

PDQCoreLibGetVersion

PDQTelMakeCall

PDQTelEndCall

PDQTelAnswerCall

PDQTelGenerateDTMF

PDQTelGetCallInfo

PDQTelResumeDialing

PDQTelCancelPause

PDQTelFormatNumber

PDQTelGetDigit

Function Descriptions

PDQCoreLibGetVersion

Purpose Retrieve the version number of the Core library.

Prototype Err PDQCoreLibGetVersion (UInt refNum, DWordPtr dwVerP)

Parameters ->refNum Reference number of the Registry library.

->dwVerP Pointer to version.

Result PDQErrNoError

PDQTelMakeCall

Purpose Originate a voice call.

Prototype Err PDQTelMakeCall (UInt refNum, CharPtr pszNumber)

Parameters ->refNum Reference number of the Core library.

->pszNumber Phone number to call. Can include digits, letters, and pause characters. Pass NULL to redial last number.

Result PDQErrNoError

PDQErrBadNumber

PDQErrNoSignal

PDQErrNoService

PDQErrAlready

PDQErrNoMemory

Comments This function converts letters to keypad digits, strips punctuation and white space out, and handles soft (timed) and hard pauses. Use enums DIG_SOFTPAUSE and DIG_HARDPAUSE for this. The first pause begins immediately upon entering conversation.

See also PDQTelEndCall

PDQTelEndCall

Purpose End a call.

Prototype Err PDQTelEndCall (UInt refNum)

Parameters ->refNum Reference number of the Core library.

Result PDQErrNoError

PDQErrAlready

PDQErrNoMemory

PDQErrPhoneOff

See also PDQTelMakeCall

PDQTelAnswerCall

Purpose Answer an incoming voice call.

Prototype Err PDQTelAnswerCall (UInt refNum)

Parameters ->refNum Reference number of the Core library.

Result PDQErrNoError

PDQErrPhoneOff

PDQTelGenerateDTMF

Purpose Send a DTMF.

Prototype Err PDQTelGenerateDTMF (UInt refNum,
Byte nKey,
Boolean bLong)

Parameters ->refNum Reference number of the Core library.
->nKey Key to send
->bLong Long or short DTMF.

Result PDQErrNoError

PDQErrPhoneOff

PDQTelGetCallInfo

Purpose Get current call information.

Prototype Err PDQTelGetCallInfo (UInt refNum, CallInfoPtr pi)

Parameters ->refNum Reference number of the Core library.
<-pi Call information.

Result PDQErrNoError

PDQErrNoCallInfo

PDQTelResumeDialing

Purpose Resume sending DTMFs after a hard pause.

Prototype Err PDQTelResumeDialing (UInt refNum)

Parameters ->refNum Reference number of the Core library.

Result PDQErrNoError

See also PDQTelCancelPause

PDQTelCancelPause

Purpose Cancels out of pause state and remains in conversation state.

Prototype Err PDQTelCancelPause (UInt refNum)

Parameters ->refNum Reference number of the Core library.

Result PDQErrNoError

See also PDQTelResumeDialing

PDQTelFormatNumber

Purpose Format a phone number.

Prototype
```
CharPtr PDQTelFormatNumber (UInt refNum,
                            CharPtr pszSrc,
                            CharPtr pszDest,
                            Int nSize,
                            Word wFlags)
```

Parameters ->refNum Reference number of the Core library.

	->pszSrc	Phone number
	<-pszDest	Formatted phone number
	->nSize	Size (including NULL) of pszDest
	->wFlags	Format flags (see comments below)

Result Returns pointer to end or first invalid digit.

Comments Use these defines in pdQCore.h:

```
#define TFN_RAW           0    // Unformatted: XXXXXXXXXXX
#define TFN_FORMATTED     0x0001// Format:  X(XXX) XXX-XXXX
#define TFN_DEFAULTAREA   0x0002// Fully-qualified 11 digit number
#define TFN_ADDAREA       0x0004// Add area to 7 digit US number
#define TFN_PREPEND       0x0008// Prepend user preferred local/long
distance string
#define TFN_APPEND        0x0010// Append user preferred local/long
distance string
```

PDQTelGetDigit

Purpose Convert a character to a phone keypad digit.

Prototype
```
Char PDQTelGetDigit (UInt refNum,
                     Char ch,
                     Boolean * bValid)
```

Parameters ->refNum Reference number of the Core library.

	->ch	Character
	<-bValid	Is it a valid phone keypad digit?

Result Converted digit if valid.

pdQ Registry API

This section includes APIs for URL (Universal Resource Locator) and macro handling. URLs are part of an Internet standard that specifies a mechanism for universally accessing objects across the Internet. See RFC 1738 for information on URIs and URLs. One of the primary advantages of URLs is that it defines a universal way for different computing devices using different operating systems to access the same object. For example, "<http://www.qualcomm.com>" is a URL that can be used to launch a pdQ smartphone browser and go to QUALCOMM's pdQ smartphone-site. Users can use this URL to view QUALCOMM's site on their desktop computer as well as their pdQ smartphone.

The basic format of a URL string consists of a scheme, followed by a colon, followed by a path. It is the responsibility of the scheme handler to interpret the path. The pdQ smartphone handles a number of schemes: http:, tel:, and mailto:. For a detailed description of how the applications handle these schemes, see Appendices C-D. Third party applications can handle these schemes by replacing pdQ smartphone's default handlers.

In addition to URL handling, the pdQ smartphone phone supports macro handling. A macro is simply a URL bundled with a name and description. Applications can take advantage of the macro facility to record and execute frequently used URLs.

An application can dynamically register and unregister for a URL scheme by calling **PDQRegAddScheme** or **PDQRegRemoveScheme**. Applications can also enable or disable registered scheme handlers by calling **PDQRegEnableScheme**. To list registered schemes, an application should call **PDQRegEnumSchemes**. To determine which application is registered for a particular scheme, applications can use **PDQGetSchemeHandler**. An application should call **PDQRegProcessURL** to dispatch a URL to its registered handler. To create an e-mail message with the registered e-mail application, an application can use **PDQRegProcessMailAddress**.

Applications can add and remove macros with **PDQRegAddMacro** and **PDQRegRemoveMacro**. To list registered macros, applications can call **PDQRegEnumMacros**.

The following function prototypes can be found in the header file pdqRegistry.h.

Summary of functions:

PDQRegGetVersion

PDQRegAddScheme

PDQRegRemoveScheme

PDQRegEnableScheme

PDQRegEnumSchemes

PDQRegGetSchemeHandler

PDQRegProcessURL

PDQRegProcessMailAddress

PDQRegAddMacro

PDQRegRemoveMacro

PDQRegEnumMacros

PDQRegSetHandlerName

Function Descriptions

PDQRegGetVersion

Purpose Retrieve the version number of the Registry library.

Prototype Err PDQRegGetVersion (UInt refNum, DWordPtr dwVerP)

Parameters ->refNum Reference number of the Registry library

->dwVerP Pointer to version

Result PDQErrNoError

PDQRegAddScheme

Purpose Add scheme to registry.

Prototype Err PDQRegAddScheme (UInt refNum,
CharPtr pszScheme,
ULong uCreator,
ULong uCapFlags)

Parameters ->refNum Reference number of the Registry library

->pszScheme Scheme

->uCreator Unique creator ID

->uCapFlags Capability flags

Result PDQErrNoError

PDQErrBadParam

PDQErrDBNotFound

PDQErrDBWriteFailed

Comments For uCapFlags, use these defines in pdQRegistry.h:

REGCAP_LAUNCHURL Switch to and launch application

REGCAP_SUBLAUNCH Sub-launch application

See also PDQRegRemoveScheme

PDQRegRemoveScheme

Purpose Remove scheme from registry.

Prototype Err PDQRegRemoveScheme (UInt refNum,
CharPtr pszScheme,
ULong uCreator)

Parameters ->refNum Reference number of the Registry library

->pszScheme Scheme

->uCreator Unique creator ID

Result PDQErrNoError

PDQErrBadParam

PDQErrDBNotFound

See also PDQRegAddScheme

PDQRegEnableScheme

Purpose Enable or disable registry of URL handler.

Prototype Err PDQRegEnableScheme (UInt refNum,
CharPtr pszScheme,
ULong uCreator,
Boolean bEnable)

Parameters ->refNum Reference number of the Registry library
->pszScheme Scheme
->uCreator Unique creator ID
->bEnable Enable or disable handler

Result PDQErrNoError

PDQRegEnumSchemes

Purpose List registered URL scheme handlers.

Prototype Err PDQRegEnumSchemes (UInt refNum,
PFNREGENUM pfn,
VoidPtr pUser)

Parameters ->refNum Reference number of the Registry library
->pfn Callback function
->pUser Pointer to user data

Callback Prototype Boolean (*PFNREGENUM) (VoidPtr pUser,
CharPtr pszScheme,
CharPtr pszName,
CharPtr pszShortName,
ULong uCreator,
ULong dwFlags,
Boolean bEnabled)

Result PDQErrNoError

Comments This function will call the callback function for each registered scheme handler. The pUser can be NULL, but will typically be a pointer to some data. To terminate the enumeration, return false in the callback.

PDQRegGetHandler

Purpose Get the creator ID of the application registered for this scheme

Prototype ULong PDQRegGetHandler (UInt iRefNum, CharPtr pszScheme)

Parameters ->refNum Reference number of the Registry library

->pszScheme Scheme

Result Creator ID of registered handler.

PDQRegProcessURL

Purpose Launch application that is registered to handle this URL scheme. A handler must be registered for this function to work.

Prototype Err PDQRegProcessURL (UInt refNum, CharPtr pszURL)

Parameters ->refNum Reference number of the Registry library

->pszURL URL to parse

Result PDQErrNoError

PDQErrBadParam

PDQErrNoHandler

PDQErrNotRegistered

See also PDQRegAddScheme

PDQRegProcessMailAddress

Purpose Utility function that constructs and executes a valid mailto URL of the form: mailto:pszFirstName pszLastName <pszAddress>. The default mailto scheme handler, **pdQmail™**, handles this by creating a message. This is provided as a convenient way to properly encode mailto links that include the full name. You may bypass this function and generate mailto links yourself in order to generate more headers and different formats.

Prototype Err PDQRegProcessMailAddress (UInt refNum,

CharPtr pszFirstName,

CharPtr pszLastName,

CharPtr pszAddress)

Parameters ->refNum Reference number of the Registry library

->pszFirstName First name

->pszLastName Last name

pszFirstName and pszLastName constitute the display name used when only one address is specified. May be NULL or empty. Control characters should not be used. 8-bit chars will be mostly interpreted as ISO-8859-1

->pszAddress Email address or (comma-delimited) addresses to send to.

Result PDQErrNoError
PDQErrBadAddress
PDQErrBadParam
PDQErrNoHandler
PDQErrNotRegistered

PDQRegAddMacro

Purpose Add macro to registry.

Prototype Err PDQRegAddMacro (UInt refNum,
CharPtr pszURL,
CharPtr pszShortName,
CharPtr pszDesc)

Parameters ->refNum Reference number of the Registry library
->pszURL URL
->pszShortName Name of macro
->pszDesc Description of macro

Result PDQErrNoError

See also PDQRegRemoveMacro

PDQRegRemoveMacro

Purpose Remove macro from registry.

Prototype Err PDQRegRemoveMacro (UInt refNum, CharPtr pszURL)

Parameters ->refNum Reference number of the Registry library
->pszURL URL

Result PDQErrNoError

See also PDQRegAddMacro

PDQRegEnumMacros

Purpose List registered macros.

Prototype Err PDQRegEnumMacros (UInt refNum,
PFNMACROENUM pfn,
VoidPtr pUser)

Parameters ->refNum Reference number of the Registry library
->pfn Callback function
->pUser Pointer to user data

Callback Prototype Boolean (*PFNMACROENUM) (VoidPtr pUser,
CharPtr pszURL,
CharPtr pszShortName,
CharPtr pszDesc)

Result PDQErrNoError

Comments This function will call the callback function for each registered macro. The pUser can be NULL but will typically be a pointer to some data. To terminate the enumeration, return false in the callback.

PDQRegSetHandlerName

Purpose Set display name for registered macro or scheme handler. The registry preference panel uses these names when displaying registered schemes.

Prototype Err PDQRegSetHandlerName (UInt refNum,
CharPtr pszScheme,
CharPtr pszName,
CharPtr pszShortName)

Parameters
->refNum Reference number of the Registry library
->pszScheme Scheme or URL
->pszName Application name
->pszShortName Short version of application name

Result PDQErrNoError
PDQErrBadParam
PDQErrDBNotFound

See also PDQRegAddScheme
PDQRegAddMacro

pdQ Alert API

Applications control the pdQ smartphone's various alert devices. These devices include the backlight and the LED located on the front of the pdQ smartphone. Applications should use **PDQAlertSwitch** to turn on, off, toggle, or retrieve the state of the alert devices. They may also use **PDQAlertFlash** to "flash" or repeat a cyclic pattern. Turning the backlight on will illuminate the entire screen as well as the flip keys (when the flip is closed). The LED can emit three colors: green, red, and orange.

Note: Using these functions will consume more battery power. Other applications (built-in or third party) may also control these alert devices. For instance, when the phone is on, an incoming call can flash the green LED and backlight.

The following function prototypes are found in the header file pdqAlert.h.

Summary of functions:

PDQAlertGetLibAPIVersion

PDQAlertSwitch

PDQAlertFlash

PDQAlertGetLibAPIVersion

Purpose Retrieve the version number of the Alert library.

Prototype Err PDQAlertGetLibAPIVersion (UInt refNum, DWordPtr dwVerP)

Parameters ->refNum Reference number of the Alert library

->dwVerP Pointer to version

Result PDQErrNoError

PDQAlertSwitch

Purpose Get state of alert device or switch on, off, or toggle.

Prototype Err PDQAlertSwitch (UInt refNum,
AlertDeviceType type,
AlertSwitchType turn,
Boolean *prevStateP)

Parameters ->refNum Reference number of the Alert library

->type LED_RED,
LED_ORANGE,
LED_GREEN,
VIBRATOR,
BACKLIGHT

->turn ALERT_SWITCH_OFF – switch off
ALERT_SWITCH_ON – switch on
ALERT_REVERSE_STATE – toggle on/off
LERT_GET_STATE – get state

<->prevState Previous state

Result PDQErrNoError

PDQAlertErrNotOpen

PDQAlertNotImplemented

Comments The vibrator is not implemented in version 1.0. The function will return PDQAlertNotImplemented when called with the VIBRATOR enum. Any color LED can be used to switch off or toggle the LED.

PDQAlertFlash

Purpose Turn alert device on and off repeatedly.

Prototype Err PDQAlertVibratorFlash (UInt refNum,
AlertDeviceType type,
UInt numberOfFlashes,
UInt onDuration,
UInt offDuration,

```
                        UInt periodOfSilence,
                        UInt timeout)
```

Parameters ->refNum Reference number of the Alert library

->type	LED_RED,
	LED_ORANGE,
	LED_GREEN,
	VIBRATOR,
	BACKLIGHT
->numberOfFlashes	Number of on/off times
->onDuration	Duration to remain on (in ms)
->offDuration	Duration to remain off (in ms)
->periodOfSilence	The silence after numberOfFlashes times (in ms)
->timeout	When to stop flashing (in secs)

Result PDQErrNoError

PDQAlertNotOpen

PDQAlertNotImplemented

Comments The vibrator is not implemented in version 1.0. The function will return PDQAlertNotImplemented when called with the VIBRATOR enum.

Appendix A: Sample Code

```
//
// Code snippet for an application that registers for all
incoming and missed calls.
//

#include <Pilot.h>
#include "pdqCore.h"

//  declarations and other includes ommitted...

DWord PilotMain( Word cmd, Ptr cmdPBP, Word launchFlags)
{
   switch (cmd)
   {
        case sysAppLaunchCmdSystemReset:
             pdQAppInit();
             break;
        case sysAppLaunchCmd_PDQSignal:
             pdQAppDispatchSignal((SignalParamsPtr )cmdPBP);
             break;
        // other launch codes ommitted...
   }
   return 0;
}


static void pdQAppInit(void)
{
   UInt coreRefNum = sysInvalidRefNum;
   Err err = 0;

   // get core library reference number
   // and register for missed and incoming calls

   err = SysLibFind(PDQCoreLibName, &coreRefNum);
   if (!err && (coreRefNum!= sysInvaldRefNum))
        err = PDQSigRegister(coreRefNum,SGN_CLASS_TAPI,
                        SGN_TAPI_MISSED | SGN_TAPI_INCOMING,
                        PRIORITY_0, myAppCreator,
sysFileTApplication);
   if (err)
        myAppHandleError(err);
}


static void pdqDispatchSignal(SignalParamsPtr signalParams)
{
   if(!signalParams)
```

```
        return;

   switch (signalParams->signalClass)
   {
        case SGN_CLASS_TAPI:
        {
             switch(signalParams->signal)
             {
                  case SGN_TAPI_MISSED:
                  case SGN_TAPI_INCOMING:
                  {
                       CallInfoPtr        callInfoP;
                       callInfoP = (CallInfoPtr)signalParams-
>params.pVoidArg;
                       // do something with this info...
                  }
                  break;
             }
        }
        break;
   }
}
```

Appendix B: System Patches

Here is a list of Palm OS™ III system traps that the pdQ smartphone extension software has modified. **Patching these traps may interfere with the phone functionality**.

sysTrapFrmCustomAlert
sysTrapPenCalibrate
sysTrapPenGetRawPen
sysTrapPenRawToScreen
sysTrapPenResetCalibration
sysTrapPenScreenToRaw
sysTrapScrDisplayMode
sysTrapScrInit
sysTrapHwrBacklight
sysTrapHwrBatteryLevel
sysTrapHwrIRQ2Handler
sysTrapHwrIRQ3Handler
sysTrapHwrPluggedIn
sysTrapHwrLCDSleep
sysTrapHwrLCDWake
sysTrapHwrWake
sysTrapKeyHandleInterrupt
sysTrapKeyInit
sysTrapKeySleep
sysTrapKeyWake
sysTrapSysBatteryDialog
sysTrapSysBatteryInfo
sysTrapSysGetOSVersionString
sysTrapSysHandleEvent
sysTrapSerReceiveISP
sysTrapWinInitializeWindow
sysTrapWinValidateHandle

Appendix C: tel: URL

The tel: URL is used to initiate voice calls. PdQ smartphone pages or e-mail messages might contain tel: URLs that, when tapped on, display the pdQ smartphone's dialer screen and dial the specified phone number. Note that users can specify in the dialer preferences whether to dial immediately or to only open the dialer and input the phone number.

The pdQ smartphone's support of the tel: URL is based on the Internet draft, URLs for Telephone Calls <draft-antti-telephony-url-07.txt>, (dated Feb. 3, 1999 and expiring Aug. 8, 1999) which can be found at <http://info.internet.isi.edu:80/in-drafts/files/draft-antti-telephony-url-07.txt>. Please note that this draft is still a work in progress. The pdQ smartphone's implementation may change to reflect changes in the draft as it makes its way towards becoming a standard. Listed below are a few notes about the implementation:

- We do not support "+" for international dialing. It is ignored.
- We do not support ISDN-subaddress.
- "p" is interpreted as a one-second pause.
- "w" is interpreted as a hard pause; the user must press the "resume" key to continue.
- Post-dial is implemented as a hard pause.
- We do not support DTMF-digits "A","B","C","D"-- We only support 0-9, "*", and "#". Note that the "#" DTMF must be encoded as "%23".
- Our maximum number is 32 dial digits and 32 DTMF digits.

Listed below are some sample tel: URLs:

```
<tel:1(800)349-4478>
<tel://1(800)349-4478>
<tel:18003494478>
<tel:+18003494478>
<tel:1.800.349.4478>
```

These will all correctly dial 1 800 349-4478 (QUALCOMM's technical support for CDMA products). Note that spaces are not permitted.

```
<tel:1(619)658-JOBS>
```

Alpha characters are ignored. This will dial 619658.

```
<tel:1(800)349-4478pp1>
```

This will dial 1 (800) 349-4478, wait for two seconds, and then dial DTMF digit 1.

```
<tel:1(800)349-4478w1>
<tel:1(800)349-4478;postd=1>
```

This will dial 1 800 349-4478, wait for the user to press “resume,” and then dial DTMF digit 1.

Appendix D: pdQsuite Specifications for Developers (mailto: and http: URLs)

The pdQmail™ and pdQbrowser™ applications both can be used as part of other applications and, in some cases, can be used to build simple applications.

pdQmail mailto: URL Handling

The mailto: URL is a way to start up the composition of a mail message with the headers and body initialized. For example, it enables you to pre-address a message or to create a simple form by inserting text in the body or subject of a message.

Mailto: URLs may be placed as anchors for links on pdQ smartphone pages or in HTML mail messages where the user can tap them. A mailto: link also may be created by a program and then sent to pdQmail. For example, an application could have a button labeled "suggestion" that starts the composition of a pre-addressed mail message when tapped.

Note**:** The Internet standard for mailto: URLs requires that they always be processed interactively. That is, when you launch pdQmail with a mailto: URL it will bring up the mail composition screen for the user to finish the composition and tap the send button. There is no way to non-interactively put a message in the send queue with a mailto: URL, nor will this control return to the application that called pdQmail to handle the link. (Batch sending is planned for a future release of pdQmail.)

The mailto: URL handling in pdQmail conforms to *RFC 2368, The mailto URL scheme*. The requisite hex and HTML encoding specified therein is required. Though the non-US-ASCII characters are forbidden in the standard, they may be passed to pdQmail. These are interpreted as Windows Code Page 1252 characters, a super set of ISO-8859-1. Certain characters in URLs are deemed "unsafe" in *RFC 1738, Uniform Resource Locators*. These must be hex encoded.

Only the following fields are supported:

To:, Cc:

Full RFC-822 syntax is supported. This includes specification of the full name, multiple address, and even full group syntax. Note that spaces and commas must be hex encoded. Following are some examples of encoded mailto: URLs with address:

mailto:KarlS@SD.COM - Sends mail to KarlS@SD.COM

mailto:?to=Joe%20Smith%20<jsmith@xx.com> - Sends mail to the fully formatted address `'Joe Smith <jsmith@xx.com>'`

mailto:?to=xx@yy.com%2Caa@bb.com%2Cgg@hh.com?cc=rr@ss.com - Sends mail with xx@yy.com, aa@bb.com and gg@hh.com in the To: field and rr@ss.com in the Cc: field. (%2C is hex for a comma.)

mailto:?to=My%20Friends%3Ajoe@z.com%2Cjane@z.com%2Cjeff@z.com%3B - Uses RFC-822 group syntax to send mail to joe@z.com, Jane@z.com and jeff@z.com. (%3A is hex for a colon, and %3B hex for a semicolon.)

Subject:

No formatting requirements are placed on the subject. Hex encoding of space, comma, percent, and other unsafe characters is required by RFC 1738.

X-Priority:

The value for this must be a single character, “1”, ”2”, ”3”, ”4”, or “5”. “1” is highest, “3” is normal, and “5” is lowest.

Body:

There are no restrictions other than proper hex encoding of special characters. Line breaks should be encoded as ‘%0d%0a’. The size of the body is limited by available memory. A 1Kb body should be acceptable in any case.

pdQbrowser HTML Capabilities

HTTP Cookies

- Maximum cookie size is 1Kbyte.
- Maximum size for all cookies is 6 Kbytes.
- When maximum size is exceeded, prejudice is shown against larger cookies. This rule is applied: Any cookies larger than 140 bytes will be deleted first, followed by the least-recently-used cookie.
- "Expire at end of session" cookies are expired in eight hours because a "session" in PalmOS is different from a "session" in desktop operating systems.

HTTP Basic Authentication

- Passwords are displayed literally, not as "*" characters.
- HTTP keeps a single-entry cache of user name and password values; this cache does not outlive the application instance (i.e., switching applications will erase this information).

HTTP Miscellaneous Features

- A proxy server can be configured. Host name, port, and list of "exception" domain suffixes can be specified. Only HTTP URLs are proxied.
- Redirection is supported.
- The "Refresh:" header is supported as long as no delay (or a delay of 0) is specified. Note: HTML <META HTTP-EQUIV=...> tags are not supported. Only "Refresh" headers actually in the HTTP response will be understood.

HTML Capabilities

Note: HTML capabilities apply to viewing HTML in pdQmail and pdQbrowser.

HTML v3.2 syntax is fully supported. Most capabilities implied by HTML v3.2 are also supported, but some are not, such as text styles that are beyond the capability of the device (e.g., color).

The following features are NOT supported:

- Images, image maps, TYPE=FILE input fields, scripting (e.g., JavaScript)
- Java (APPLET), Netscape plug-ins (EMBED), ActiveX, Cascading Style Sheets (CSS)
- META HTTP_EQUIV=..., and FRAMESET (not supported the way frames are normally displayed)

The following features ARE supported:

- Forms
- Lists
- Blockquotes
- Rudimentary FRAMESET navigation capability

Text Styles and Character-Level Elements

- Two different text sizes are supported: (1) PalmOS regular font and bold font mapped to HTML sizes 1 through 3 and (2) Large font mapped to sizes 4 through 7. Some HTML elements to which this applies are FONT SIZE=..., BIG, SMALL, and H1 through H6.
- Normal and boldface text are supported. (To synthesize large+bold, overstrike the large font.)
- Linked text is displayed with a dashed underline.
- <U> results in solid underline.
- <I> (and similar tags) result in dashed underline.

- Fixed-width text and color are NOT displayed.

Paragraph Styles and Block-Level Elements

- Lists are supported.
- BLOCKQUOTEs are supported.
- BLOCKQUOTE CITE or TYPE=CITE text is represented by a vertical bar in the left margin. ("TYPE=CITE" is a Netscape-ism. A "CITE" attribute, or "CITE=..url..", is a preferred method.)
- While fixed-width fonts are not supported, XMP, PRE, LISTING, and PLAINTEXT are parsed properly (spaces and new lines are significant).
- Tables are not supported, but table tags (TABLE, TD, TR, etc.) are recognized and used to generate line breaks and spaces for readability.
- COMMENT, SCRIPT, and STYLE elements are treated the same: Their contents are ignored.

Character Set Handling

pdQbrowser supports Unicode character references for characters in the PalmOS character set (e.g., "♠" for spades, "•" for bullet), and ISO standardized names for ANSI CP1252 as well as Latin-1 characters (e.g., "•" and "Ÿ"). It also understands entity names "spades", "hearts", "diams", and "clubs" for the characters Palm added to CP1252. An exhaustive description of the supported character set is in "charset.html".

Forms Support

The following control types are supported (everything in HTML v2.0):

INPUT TYPE=TEXT

INPUT TYPE=PASSWORD (*)

INPUT TYPE=CHECKBOX

INPUT TYPE=RADIO

INPUT TYPE=SUBMIT

INPUT TYPE=RESET

INPUT TYPE=HIDDEN

INPUT TYPE=IMAGE (*)

SELECT MULTIPLE (*)

SELECT single selection, single line - displayed as a popup trigger

SELECT single selection, multiple line - displayed as a list

TEXTAREA (*)

Notes:

- INPUT TYPE=PASSWORD input fields don't show asterisks for the entered text; instead they show exactly the text entered, due to the uncertain nature of Graffiti input, and because it is easy to keep the device private.
- INPUT TYPE=IMAGE is handled as a regular SUBMIT button, as is standard for text-based browsers.
- SELECT MULTIPLE is shown by Windows browsers as a list box which allows selection of zero or more items using the Control or Shift key. PalmOS has no such native mutiple-selection list object. PdQbrowser represents SELECT options as separate check boxes. The resulting set of controls has identical behavior to, but a very different appearance from what a desktop browser would show.
- TEXTAREA doesn't display scroll bars. To scroll, move the pen down inside the field, then drag it above or below the field. Also, TEXTAREA doesn't pay much attention to the sizing attributes specified in the HTML source; these will require some tweaking and tuning.
- When you tap on a control that is partially scrolled off the top or bottom of the document windows, the control will be scrolled into view before it is activated.

Known Bugs and Issues in Forms

- You can't enter text into fields while the document is loading.
- There are some known drawing errors that occur during activation of list objects. These are rare and very hard to duplicate.
- A form can be submitted even if the document did not finish loading, leaving the form potentially incomplete.

FRAME Support

Each FRAME element is displayed as a link to the framed document. The links are named "Frame 1", "Frame 2", etc. The title of the document is displayed before the first FRAME link. This allows pdQbrowser to navigate sites which otherwise would be dead ends.

Miscellaneous Features

- "http:\\host\a\b\c" will be silently changed to http://host/a/b/c. Authors should not rely on this because it is simply to reduce the number of bogus support calls.
- Caching: Most recently-retrieved documents are retained in a PalmOS database. Maximum cache size can be specified by the user.
- Maximum size of URL + form data is 1600 bytes.

- If unsupported features are present in a pdQ smartphone page, a message to that effect will be displayed. If a page is blank and the only unsupported feature is scripting, a message to that effect will be displayed.

Debugging Features

A URL scheme named "x-memo" accesses the contents of Memo Pad files. The format is as follows:

```
x-memo:CategoryName/Record;Params
```

CategoryName is the name of a category in Memo Pad. If "Category/" is omitted, all categories are searched.

Record is (1) a numeric position within that category (indices start at 1) or (2) a record 'name', represented as the initial paragraph of the record.

Params is optional. If given, Params can consist of one or more of the following parameters, each separated by semicolons (";").

- "slow" or "slow=<x>" : limits the speed at which data is delivered to x bytes per second (default is 100).
- "x=<n>" : causes the memo contents to be repeated n times (1<=n<=100).
- "p" : force type to "text/plain" (default is "text/html").
- "type=<mime type>": forces type to <mime type>, which may include semicolon characters (but not spaces for now). Due to its syntax, if "type=..." is given, it must be the last parameter.

An example URL is "`x-memo:HTML/1;x=2;slow`", which means load the first memo in the HTML category as text/html, duplicate its contents, and limit the loading speed to 100 bytes per second.

Another example URL is "`x-memo:Flowed;type=text/plain;format=flowed`", which will look for the first memo whose first paragraph consists entirely of "Flowed", and then process the *rest* of the memo as data of type "text/plain; format=flowed".

If the document's base URL is "`x-memo:category/name`", a relative link to "xxx" will evaluate to the URL "`x-memo:xxx`".

Index