palmOne™

Developing SDIO Peripherals for palmOne Handhelds

Copyright

palmOne, Zire, Tungsten, Treo, Blazer, 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.

Contents

About This Document

This document is intended to assist you in writing Palm OS® applications that interact with SDIO hardware. Because there is a wide range of possible SDIO devices, it focuses solely on those aspects of program design that are specific to the Palm OS, palmOne handhelds, and to the SDIO slot driver.

SDIO SDK

The latest palmOne SDIO SDK is available for download from **http://pluggedin.palmone.com**. It contains the SDIO headers, API Guide, and sample code.

Developing SDIO Applications for palmOne Handhelds

Much of an SDIO application is dictated by the hardware with which it interacts. However, because SDIO is a standard, and because these SDIO applications run on the Palm OS® handheld, all such applications have a number of traits in common. This commonality is the subject of this chapter.

This chapter begins by ensuring that you have all of the software, hardware, and documentation that you'll need to create your application. It next talks about the various aspects of the Palm OS that you'll use when writing your application, and then provides some programming guidelines specific to SDIO applications. It ends with a few pointers relative to creating the SDIO card itself.

Useful information and tools

This document is by no means an exhaustive source of information with regard to creating SDIO applications. In addition, you'll want to have a copy of the SDIO Specification and an up-to-date copy of the *Palm OS Programmer's API Reference* and *Companion* **(http://www.palmos.com/dev/support/docs/palmos/**).

If you are developing SDIO hardware, you will also want to know about palmOne's HDK (Hardware Development Kit) and EDK (Expansion Development Kit). The HDK contains mechanical specifications, drawings, and documentation that assist with the design of peripherals. The EDK is a set of parts or items available for purchase at the palmOne Expansion Parts Store (**http://pluggedin.palmone.com/regac/pluggedin/auth/PalmPartsStore**). Information on all of these items can be found at the PluggedIn Program website (**http://pluggedin.palmone.com**).

SD, SDIO, and MMC specifications

The SD Card Association (SDA) publishes the *SDIO Card Specification*, which is based on and refers to the SDA document titled *SD Memory Card Specifications, Part 1, PHYSICAL LAYER SPECIFICATION*. Both of these documents provide essential foundation material for the contents of this document. You should be familiar with the *SDIO Card Specification* and with those parts of the *SD Memory Card Specifications* that document card modes, card initialization, interrupts, registers, and card reading and writing. Depending on the SDIO hardware with which you are working, additional sections of the *SD Memory Card Specifications* document may be of interest.

The SD Card Association's website can be found at **http://www.sdcard.org/**. You'll need to be a member in order to obtain the specifications from the SD Card Association.

NOTE Creating Palm OS applications that can use and exchange data from other products via SD Memory cards is outside the scope of this document. However, to make sure that data can be interchanged with present and future SD products, please refer to the appropriate SD Association specification depending on the type of application.

For developers working with MultiMediaCards (MMC), the MultiMediaCard Association's website can be found at **http://www.mmca.org/**. The MMC specifications are available from the MultiMediaCard Association to MMCA members.

The SDIO slot driver has been written to accommodate the following specifications:

- MMC memory cards, V1.4 to V3.0
- SD memory cards, Part 1, V1.0 (and the supplement to part 1)
- SDIO V1.0

Palm OS SDK

General Palm OS programming concepts are documented in the *Palm OS Programmer's Companion*. Reference documentation for the APIs made public by the Palm OS can be found in the *Palm OS Programmer's API Reference*. Both of these documents are installed as part of the Palm OS Software Developer's Kit (SDK), which can be found at **http://www.palmos.com/dev/tools/core.html**. SDIO applications are not supported on versions of the Palm OS prior to 4.0.

Although you'll want to be familiar with a number of different aspects of Palm OS programming, pay particular attention to the portions of the *Companion* and *Reference* that cover the Expansion and VFS Managers; these chapters show you how to read and write expansion media, including SD memory cards.

In addition to the Palm OS SDK, you should also have the header files for the SDIO slot driver and copies of the SDIO sample applications provided by palmOne. These are included with the palmOne SDIO SDK. The header files included with the SDIO SDK are compatible with the Palm OS SDK and must be copied into a folder in your project's "include" path.

Software Architecture of an SDIO Application

Palm OS applications that interact with SDIO cards make use of the functions provided by the Expansion Manager, the VFS Manager, and the SDIO slot driver. Before you can write such a Palm OS application, you should have an understanding of how your application will interact with these and other features of the Palm OS.

Figure 1 presents a simplified view of how the SDIO slot driver relates to your applications, the Expansion Manager, and the VFS Manager. Unlike other Expansion Manager slot drivers, the SDIO slot driver exposes its APIs to applications. Because it also lies beneath the Expansion and VFS managers, you access SDIO hardware through a combination of Expansion Manager, VFS Manager, and SDIO slot driver calls. Note that you use the VFS Manager with a given SDIO card only if there is an SD or SDIO file system present on that card.

The VFS Manager APIs are used for all file system access on an expansion card. When inserted, SD memory and SDIO CSA memory is mounted as file system

memory. Therefore, access to these memory areas is done using the VFS Manager APIs. Details of accessing data on file systems can be found in the standard Palm OS documentation on Expansion Manager and VFS APIs.

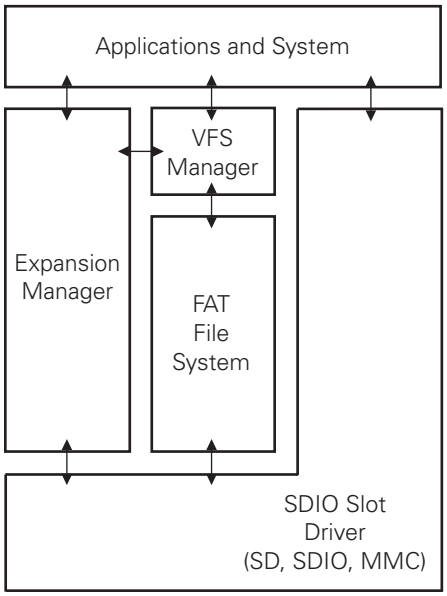

Figure 1 Relationship between SDIO application, SDIO slot driver, and other key OS components

Expansion Manager

The Expansion Manager is a software layer that manages slot drivers on Palm OS handhelds. The Expansion Manager is not solely responsible for support of expansion cards; rather, it provides an architecture and higher-level set of APIs that, with the help of low-level slot drivers and file system libraries, support various types of media.

The Expansion Manager:

- broadcasts notification of card insertion and removal
- plays sounds to signify card insertion and removal
- mounts and unmounts card-resident volumes

NOTE Some of the functions provided by the Expansion Manager are designed to be used by slot drivers and file systems and are not generally used by third-party applications.

For a detailed explanation of the functions that make up the Expansion Manager, see the "Expansion Manager" chapter in the *Palm OS Programmer's API Reference* (**http://www.palmos.com/dev/support/docs/palmos/PalmOSReference/ReferenceTOC.html**).

VFS Manager

The VFS (Virtual File System) Manager provides a unified API that gives applications access to many different file systems on many different media types, including SD media. The VFS Manager is used for all file system access on an expansion card. In the case of an SDIO card, the VFS Manager is typically used to access any function CSA memory. The data stored in CSA memory is structured as a FAT12/16 file system (FAT 12/16/32 for LifeDrive™ mobile manager devices) and is therefore ideally suited for access by the VFS Manager.

Combo cards may contain SD memory that is also accessed through the VFS Manager APIs.

For a detailed explanation of the functions that make up the VFS Manager, see the "Virtual File System Manager" chapter in the *Palm OS Programmer's API Reference* (**http://www.palmos.com/dev/support/docs/palmos/PalmOSReference/ReferenceTOC.html**).

SDIO Slot Driver

To simplify the interaction with the SDIO hardware, palmOne has created an SDIO slot driver. It replaces the Palm OS SD/MMC slot driver, which isn't SDIO-aware, and consists of data structures and functions that allow you to easily manage power, interrupts, and data on the SDIO card.

The SDIO slot driver controls all media supported by an SD expansion slot, including SD media, MMC media, and SDIO media.

An examination of the functions provided by the SDIO slot driver shows that it implements most of the software functionality outlined in the SDIO Card Specification. It does not, however, support the following:

- SDIO Suspend/Resume Operation
- SDIO Read Wait Operation
- SDIO RW Extended Block Operation in "forever" mode

Notification Manager

The Palm OS Notification Manager allows applications to receive notification when certain system-level or application-level events occur. Although the Notification Manager has many uses, developers of SDIO applications should particularly take note of the fact that you use it to detect card removal by registering for a `sysNotifyCardRemovedEvent`

Guidelines for SDIO Applications

All SDIO applications need to be aware of the power needs of the SDIO card. As well, they need to be able to handle interrupts generated by the card, and must be aware of when an SDIO card is inserted or removed from the handheld's SD slot. The following sections discuss these and other SDIO-application–specific topics.

Power Management

When the handheld awakes from sleep mode, it doesn't turn the card on. Only when there is a request to access the card does it turn the card on.

Turning on Card Functions

You can turn on a given SDIO card function with `SDIOSetPower`. Be aware that you, as an application developer, are responsible for managing card power.

You must ensure that the total of all function hardware that is active does not draw in excess of the SDIO-specified maximum of 200ma.

Perform the following steps to explicitly turn on an SDIO card function:

1. Disable SDIO interrupts with `SDIODisableHandheldInterrupt`—even if your application doesn't use interrupts.
2. Verify that there is sufficient current available to power the card function. To aid in the power management process, the SDIO slot driver provides three functions: `SDIOGetCurrentLimit`, `SDIOSetCurrentLimit`, and `SDIORemainingCurrentLimit`.

   NOTE These three functions do not detect or limit current draw, check the battery level, or reflect how much energy the battery has left.

   The current limit for each function can be obtained by calling `SDIOGetCurrentLimit` or changed by calling `SDIOSetCurrentLimit`. Prior to enabling power to a given function, call `SDIOGetCurrentLimit` to determine how much power it will draw, and compare it to the value returned from `SDIORemainingCurrentLimit`, which indicates how much current can be spared.
3. Turn the function on using `SDIOSetPower`.
4. Reenable interrupts by calling `SDIOEnableHandheldInterrupt`.

After turning off an SDIO card function (with `SDIOSetPower`), be sure to call `SDIOSetCurrentLimit` and set its current limit to zero.

When a card is removed, all of the in-memory current limits are automatically set to zero.

Auto Power Off

The `SDIOSetAutoPowerOff` function allows you to specify an amount of time after which the power and data signals to a given function on an SDIO card should be turned off. You specify this time interval in system ticks; there are `SysTicksPerSecond` ticks per second. To disable the auto-power-off feature, simply call this function and supply a tick count of zero.

To obtain the current auto-power-off settings for a given SDIO card function, use `SDIOGetAutoPowerOff`.

Callbacks

The SDIO slot driver allows your application to register callback functions that will be invoked whenever the corresponding event occurs on the SDIO card. Several of these callbacks relate to power management.

Whenever the handheld is about to be put to sleep, the callback function corresponding to `sdioCallbackSelectSleep` is called. Just after the handheld wakes, the function corresponding to `sdioCallbackSelectAwake` is called. These callback functions can be called from either an interrupt routine or a non-interrupt routine; as a result, interrupts may be disabled or enabled. In either case, they should always be as fast as possible.

Whenever SDIO card power is turned on or is about to be turned off, the callback function corresponding to `sdioCallbackSelectPowerOn` or `sdioCallbackSelectPowerOff`, respectively, is called. While processing these functions, never call `SDIOSetPower` in order to turn an SDIO card's power on or off. These functions can be called from within an interrupt handler, so they should be as fast as possible.

Interrupt Handling

An SDIO card is capable of interrupting the host device into which it is inserted—in this case, the palmOne handheld. The SDIO slot driver allows you to register a callback function that is called whenever the card interrupts the handheld.

Register for the interrupt callback by calling `SDIOSetCallback` and specifying that you are registering for `sdioCallbackSelectInterruptSdCard`. In your callback function, be sure to reset the interrupt source to prevent the interrupt callback from being called again inadvertently.

Whether or not you have registered an interrupt callback function, you can enable or disable the SDIO interrupt on the handheld by calling `SDIOEnableHandheldInterrupt` or `SDIODisableHandheldInterrupt`. Note that these functions only affect interrupts on the handheld; they do not turn on or off interrupts on the SDIO card itself.

These functions are implemented as an incrementing counter, making them re-entrant. For instance, for every call to `SDIODisableHandheldInterrupt` there must be an equal number (or more) of calls to `SDIOEnableHandheldInterrupt` in order to re-enable interrupts.

By default, when the card is inserted interrupts on the handheld are enabled, but are disabled internally until an interrupt callback is set with `SDIOSetCallback`. Note that in order to receive the SDIO interrupt, power to the card must be on, even if the handheld is asleep.

Detecting Card Insertion and Removal

Applications that depend on the presence of the SDIO card in the slot should register for a `sysNotifyCardRemovedEvent`, which is broadcast when the user removes the card from the SD slot.

Be sure to unregister for the `sysNotifyCardRemovedEvent` notification and any SDIO callbacks when your application terminates.

For more information on registering and unregistering for notifications, see the Notification Manager chapter in the *Palm OS API Reference Guide* (**http://www.palmos.com/dev/support/docs/palmos/PalmOSReference/ReferenceTOC.html**). The "Expansion" chapter of the *Palm OS Programmer's Companion*, vol. I (**http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/CompanionTOC.html**) discusses, among other things, the

various notifications that are issued when a card is inserted or removed, or when a volume is mounted or unmounted.

Auto Run

When a card is inserted into the SD slot, after it has been initialized any file system memory present on the card is mounted by the Expansion Manager. This includes all SD memory, in the case of a standard SD card or SDIO combo card, and all SDIO Function CSA memory for functions 0-7.

After mounting of the file systems, the SDIO slot driver broadcasts a series of Auto Run (`sysNotifyDriverSearch`) notifications. These notifications are sent in an attempt to locate function- or card-specific drivers, and allow those drivers that are already on the handheld to launch themselves.

The typical sequence of events after a card is inserted is as follows:

1. Power is applied to the card.
2. The card is initialized according to the SDIO, SD, or MMC specification, as appropriate.
3. Information about the card (tuples, clock speed, CSD, CID, etc.) is read.
4. Any recognized file systems are mounted.
5. `sysAppLaunchCmdCardLaunch` is sent to `start.prc` on each mounted file system.
6. The Auto Run notifications (`sysNotifyDriverSearch`) are sent.
7. `sysAppLaunchCmdNormalLaunch` is sent to `start.prc` on each mounted file system.

For SDIO cards, one Auto Run notification is broadcast for the SD memory portion of a combo card, and an additional notification is broadcast for each card function (up to 7). For SD memory and MMC memory cards, only one such notification is sent. The notifications are sent starting with SD memory, followed by function 7 (if there is one) and proceeding to function 1 as appropriate.

The `notifyDetailsP` field of the `SysNotifyParamType` structure that accompanies the Auto Run notification points to an `AutoRunInfoType` structure. Each driver that has registered for `sysNotifyDriverSearch` should examine the contents of the `AutoRunInfoType` structure to determine if it is the driver that should control the inserted card. If so, the driver should then check the `SysNotifyParamType` structure's `handled` field. If `handled` is set to `true`, another driver has received the broadcast and will control the card. If `handled` is set to `false`, the driver should set it to `true` to indicate that it will control the device.

Developing the SDIO Peripheral

An SDIO application is only as good as the hardware with which it interacts. The following sections provide some tips for the creation of an SDIO peripheral to be used with a palmOne handheld.

EDK

palmOne has made available the SDIO Developer Card #1, a sample SDIO design demonstrating an SDIO interface to a microcontroller. It is an Expansion Developer Kit (EDK) that allows hardware developers to experiment with SDIO hardware and

software for prototyping and evaluation purposes. The card includes a flash programmable PIC microcontroller and a CPLD for maximum flexibility in prototyping.

palmOne's EDK is available for purchase at the palmOne Expansion Parts Store (**http://pluggedin.palmone.com/regac/pluggedin/auth/PalmPartsStore**).

Specifications

When developing an SDIO peripheral, it is extremely important that you following the specifications identified in **SD, SDIO, and MMC specifications.** Be sure to pay close attention to the power restrictions, as the palmOne handheld isn't able to deliver more power to an SDIO peripheral than the specification maximum.

SDIO Slot Driver

A palmOne handheld running Palm OS 4.0 or 5.0 supports SD/MMC expansion cards. If the SDIO slot driver is installed, it will also support SDIO expansion cards. In both cases, only one file system can be mounted for a given expansion card. Future versions of the Palm OS will likely lift this restriction, allowing up to seven file systems to be mounted for an SDIO expansion card.

In order to support SDIO peripherals, handhelds running Palm OS 4.0 must either be flash-upgraded to a version of the OS that supports SDIO, or must have the SDIO slot driver separately installed in RAM. The SDIO slot driver can be downloaded from the palmOne website and installed as a PRC file in RAM on Palm OS 4.0 devices. After a soft reset, the slot driver in RAM is recognized and takes precedence over the SD/MMC slot driver in ROM.

NOTE The latest Palm OS 5.0 based devices have the SDIO slot driver built into the ROM.

You can verify whether a given slot driver is "SDIO-aware" by calling `SDIOAPIVersion`. This function returns `expErrUnimplemented` if the specified driver doesn't support SDIO, or errNone if it does. If the driver does support SDIO this function also returns the slot driver version number through the `versionP` parameter.

To remove the SDIO slot driver from RAM, you must perform a hard reset of the handheld. You cannot delete the SDIO slot driver using the Application Launcher's "Delete" function. Note that to avoid having the SDIO slot driver reinstalled on the handheld during the next HotSync operation, you must remove the slot driver PRC from the Backup directory of your desktop computer.

SDIO Card Initialization and Identification on Palm OS

The process of identifying and initializing an SDIO card is specified in the *SDIO Card Specification*. One of the first steps in developing an SDIO card is to have the card identify itself as an SDIO card to the host.

Identification

Identification of a card is done only once, at the time the card is inserted into the handheld's SD slot. Information obtained from the card during the identification phase is retained in the handheld's memory until the card is removed. Among other things, this information includes:

- the type of card in the slot
- what the card contains
- the card's limits
- data read from tuples

By default SDIO cards power-off automatically after a certain amount of inactivity. This behavior can be modified with the `SDIOSetAutoPowerOff` function.

Initialization

A card is initialized every time it is turned on. The SDIO slot driver follows the appropriate initialization flowchart—SD mode or SPI mode—from the "SDIO Card Initialization" section of the *SDIO Card Specification* to initialize the card.

During the initialization phase, the handheld operates within the range of SD or SPI clock frequencies specified in the SD Memory Card Specifications (from zero to 400kHz). The actual clock frequency used depends upon the model of the palmOne handheld.

The TPLFID_FUNCTION tuple, located immediately after the CISTPL_FUNCID tuple in the CIS for function 0, contains the TPLFE_MAX_TRAN_SPEED byte. This byte indicates "the maximum transfer rate per one data line during data transfer"; essentially, the maximum clock frequency that the card can support. As soon as this tuple is read, the SDIO slot driver increases the clock speed to the highest possible frequency that doesn't exceed the maximum specified in TPLFE_MAX_TRAN_SPEED.

CSA

In order for an SDIO card's CSA (Code Storage Area) to be readable by the Palm OS, the CSA should be in FAT12/16 format (FAT 12/16/32 for LifeDrive™ mobile manager devices), and any drivers, data, or applications that the peripheral would like to be automatically detected by the palmOne handheld should reside in the `/Palm` and `/Palm/Launcher` directories. Once the CSA area is mounted, applications may access any data within the CSA, irrespective of the directory in which that data resides.