



C/C++ Sync Suite Companion for Macintosh

**Palm OS® Conduit Development Kit for
Macintosh, Version 4.03**

CONTRIBUTORS

Updated by Eric Shepherd and Brent Gossett
Engineering contributions by Cole Goeppinger and Chris Tate

Copyright © 1996 - 2002, PalmSource, Inc. and its affiliates. All rights reserved. This documentation may be printed and copied solely for use in developing products for Palm OS® software. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation or adaptation) without express written consent from PalmSource, Inc.

PalmSource, Inc. reserves the right to revise this documentation and to make changes in content from time to time without obligation on the part of PalmSource, Inc. to provide notification of such revision or changes.

PALMSOURCE, INC. AND ITS SUPPLIERS MAKE NO REPRESENTATIONS OR WARRANTIES THAT THE DOCUMENTATION IS FREE OF ERRORS OR THAT THE DOCUMENTATION IS SUITABLE FOR YOUR USE. THE DOCUMENTATION IS PROVIDED ON AN "AS IS" BASIS. PALMSOURCE, INC. AND ITS SUPPLIERS MAKE NO WARRANTIES, TERMS OR CONDITIONS, EXPRESS OR IMPLIED, EITHER IN FACT OR BY OPERATION OF LAW, STATUTORY OR OTHERWISE, INCLUDING WARRANTIES, TERMS, OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND SATISFACTORY QUALITY. TO THE FULL EXTENT ALLOWED BY LAW, PALMSOURCE, INC. ALSO EXCLUDES FOR ITSELF AND ITS SUPPLIERS ANY LIABILITY, WHETHER BASED IN CONTRACT OR TORT (INCLUDING NEGLIGENCE), FOR DIRECT, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, OR PUNITIVE DAMAGES OF ANY KIND, OR FOR LOSS OF REVENUE OR PROFITS, LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, OR OTHER FINANCIAL LOSS ARISING OUT OF OR IN CONNECTION WITH THIS DOCUMENTATION, EVEN IF PALMSOURCE, INC. OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

PalmSource, the PalmSource logo, AnyDay, EventClub, Graffiti, HandFAX, HandMAIL, HandSTAMP, HandWEB, HotSync, the HotSync logo, iMessenger, MultiMail, MyPalm, Palm, the Palm logo, the Palm trade dress, Palm Computing, Palm OS, Palm Powered, PalmConnect, PalmGear, PalmGlove, PalmModem, PalmPak, PalmPix, PalmPoint, PalmPower, PalmPrint, Palm.Net, Simply Palm, ThinAir, and WeSync are trademarks of PalmSource, Inc. or its affiliates. All other product and brand names may be trademarks or registered trademarks of their respective owners.

IF THIS DOCUMENTATION IS PROVIDED ON A COMPACT DISC, THE OTHER SOFTWARE AND DOCUMENTATION ON THE COMPACT DISC ARE SUBJECT TO THE LICENSE AGREEMENT ACCOMPANYING THE COMPACT DISC.

C/C++ Sync Suite Companion for Macintosh
Document Number 3002-003
June 19, 2002
For the latest version of this document, visit
<http://www.palmos.com/dev/support/docs/>.

PalmSource, Inc.
1240 Crossman Avenue
Sunnyvale, CA 94089
USA
www.palmsource.com

Table of Contents

About This Document	ix
Conduit Development Kit Documentation	x
What this Document Contains	x
HotSync Manager Application	xii
CDK Version Compatibility	xii
CDK/Handheld Compatibility	xiii
Conduit Development Environments	xiii
Conventions Used in this Document	xiii
Additional Resources	xiv
 1 Conduit Quick Start	 1
Conduit Creation Fast Track Steps	1
A. Install Conduit Development Tools.	1
B. Create a New Conduit Project	2
C. Take a Trial Run	3
D. Customize the Conduit Information Resource	4
E. Customize the Conduit Entry Points	5
Conduit Design Goals	7
 2 Conduit Development Basics	 9
Palm OS Platform Connections	9
Software Components.	11
Desktop Applications.	13
Handheld Applications	13
The HotSync Manager Application	13
Conduits	14
Conduit Development.	14
Conduit Entry Points	15
The Conduit Information Resource	15
The Sync Manager API	16
The Generic Conduit Framework.	16
The User Manager API	16
Carbonization and Mac OS X	16

3 How the HotSync Manager Application Works with Conduits	21
Terminology	22
HotSync Operations.	22
Determining the Synchronization Configuration	24
Running Conduits	25
Finishing the Synchronization	28
Fast and Slow Synchronizations	28
Determining If Fast Sync is Possible	29
Database Modification Information	29
HotSync Connections	30
 4 Developing Conduits	 31
About Conduits	31
Programming Conduits	31
Conduit Tasks	32
Conduit Synchronization Types	32
Conduit Design Goals	34
Conduit Development Basics.	34
Conduit Design Decisions	36
Conduit Design Questions.	36
 5 Generic Conduit Framework	 39
Introduction to the Generic Conduit Framework	39
Why Use GenCn?	39
What Will the Generic Conduit Framework Do for Me?	40
Cross-Platform Development With GenCn.	40
Generic Conduit Framework Overview	41
GenCn Class Hierarchy	42
Developing Data Formats	43
Synchronization Control Flow	44
Mirror-Image Synchronization	44
Basic Synchronization.	45
Overview of Mirror-image Synchronization	45
Details of Mirror-image Synchronization	46
Synchronizing to More than One Desktop	48

Overview of Synchronization Procedures	50
Creating Your Generic Conduit	53
The CPcMgr Class	54
6 Using the Sync Manager API	57
Communications Between Handheld and Desktop	57
Using the Sync Manager	58
Registering Your Conduit With Sync Manager	58
Opening Your Database	58
Working with Databases	59
Cleaning Up Records	61
Closing Your Database	62
7 Interfacing with the HotSync Manager Application	63
The Conduit Entry Points	63
ConfigureConduit	65
GetActionString	66
CopyActionStringAsCFStringRef	67
GetConduitName	69
CopyConduitNameAsCFStringRef	70
GetConduitVersion	72
OpenConduit	72
OpenConduitCarbon	74
Logging Entries in the HotSync Log	75
8 Using Expansion Technology	77
Expansion Support	77
Primary vs. Secondary Storage	78
Expansion Slot	79
Universal Connector	79
Architectural Overview	80
Slot Drivers	82
File Systems	83
VFS Manager	83
Expansion Manager	84

Standard Directories	85
Card Insertion and Removal	86
Checking for Expansion Cards	87
Verifying Handheld Compatibility	87
Checking for Mounted Volumes	90
Enumerating Slots	91
Determining a Card's Capabilities	92
Volume Operations	93
Hidden Volumes	95
Matching Volumes to Slots	95
Naming Volumes.	96
File Operations	96
Common File Operations	97
Naming Files	99
Working with Palm OS Databases	99
Directory Operations	100
Directory Paths	101
Common Directory Operations.	101
Enumerating the Files in a Directory	102
Determining the Default Directory for a Particular File Type.	103
Default Directories Registered at Initialization	104
Custom Calls.	107
Custom I/O	107
Summary of Expansion and VFS Managers	109

9 Installing Conduits 111

The Conduit Information Resource	111
Conduit Domains.	114
Using the User Manager API.	114
Installing Conduits	115
Installing Handheld Applications	115
Testing Your Conduit	115
Conduit Cautions	116

10 Debugging Conduits	117
Using the HotSync Manager Application Log	117
Location of the Log File	118
Debugging Tips and Tricks.	118
Common Troubleshooting Help	119
Disabling Timeouts.	122
Disabling Other Conduits	122
 Glossary	 125
 Index	 131

About This Document

The C/C++ Sync Suite for Macintosh is the new name of the Conduit Development Kit (CDK) for Macintosh from PalmSource, Inc. It provides APIs, C++ class frameworks, samples, and documents to help developers create C API-based conduits that run on Macintosh computers. Key to the success of the Palm OS[®] platform, conduits are software objects that exchange and synchronize data between an application running on a desktop computer and a Palm Powered[™] handheld.

The *C/C++ Sync Suite Companion for Macintosh* provides an overview of how C API-based conduits operate and how to develop them with the C/C++ Sync Suite for Macintosh.

The sections in this introduction are:

- [Conduit Development Kit Documentation](#)
- [What this Document Contains](#)
- [HotSync Manager Application](#)
- [CDK Version Compatibility](#)
- [CDK/Handheld Compatibility](#)
- [Conduit Development Environments](#)
- [Conventions Used in this Document](#)
- [Additional Resources](#)

About This Document

Conduit Development Kit Documentation

Conduit Development Kit Documentation

The latest versions of the documents described in this section can be found at

<http://www.palmos.com/dev/tech/docs/>

The following two documents are part of the C/C++ Sync Suite:

Document	Description
<i>C/C++ Sync Suite Companion for Macintosh</i>	An introduction to conduit development that provides an overview of how conduits are used and how to implement them.
<i>C/C++ Sync Suite Reference for Macintosh</i>	This API reference document that contains descriptions of all conduit function calls and important data structures.

For more information about programming for the Palm OS platform, see the following Palm OS documents, which are part of the Palm OS Software Development Kit:

Document	Description
<i>Palm OS Programmer's API Reference</i>	An API reference document that contains descriptions of all Palm OS function calls and important data structures.
<i>Palm OS Programmer's Companion</i>	A guide to application programming for the Palm OS. This volume contains conceptual and "how-to" information that compliments the Reference.
<i>Palm OS Programming Development Tools Guide</i>	A guide to writing and debugging Palm OS applications with the various tools available.

What this Document Contains

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

- [Chapter 1, "Conduit Quick Start."](#) Provides you with a quick guide to developing a conduit. Use this quick start guide if you already understand some of the basic concepts of HotSync® technology and developing software for the Palm OS platform.

- [Chapter 2, “Conduit Development Basics.”](#) Provides programmers who are new to the Palm OS development world with an overview of the hardware and software components that are involved and the interactions among them.
- [Chapter 3, “How the HotSync Manager Application Works with Conduits.”](#) Provides a detailed description of the HotSync process and how it is used to synchronize data between a desktop computer and a handheld.
- [Chapter 4, “Developing Conduits.”](#) Provides an overview of conduits and the major components that you must implement to develop a conduit.
- [Chapter 5, “Generic Conduit Framework.”](#) Describes the Generic Conduit Framework, a pre-assembled conduit, written in C++, that you can augment with your own subclasses to easily develop the conduit you need.
- [Chapter 6, “Using the Sync Manager API.”](#) Provides an overview of the Sync Manager API, which is the application programming interface that conduits use to send and retrieve data between the handheld and the desktop computer.
- [Chapter 7, “Interfacing with the HotSync Manager Application.”](#) Describes the entry points that the HotSync Manager calls in a conduit.
- [Chapter 8, “Using Expansion Technology.”](#) Describes how to work with handheld expansion cards and add-on devices from the desktop using the Expansion and Virtual File System (VFS) Managers.
- [Chapter 9, “Installing Conduits.”](#) Describes the conduit information resource, which you must provide for your conduit to work on Macintosh desktop computers.
- [Chapter 10, “Debugging Conduits.”](#) Provides an overview of techniques that you can use to help debug your conduits.

HotSync Manager Application

On Macintosh desktop computers, the HotSync Manager Application is implemented in three components:

- The HotSync Manager is the executable that provides configuration and logging facilities for synchronization operations.
- The Transport Monitor is a background application that watches the serial port for the wakeup packet from the handheld and then passes control to the Conduit Manager.
- The Conduit Manager manages the synchronization process and the execution of the individual conduits.

This book uses the term “the HotSync Manager application” to refer to these three components as a single entity.

CDK Version Compatibility

Each Palm Powered handheld ships with a specific version of the HotSync Manager, which is the application that runs on the Macintosh desktop computer and controls synchronization operations.

Each version of the HotSync Manager includes a version of the interface to an important data access library used by conduits, which is called the Sync Manager API.

You can use this conduit development kit to develop conduits for the Macintosh HotSync Manager version 3.0, and for Sync Manager API versions through version 2.3.

The table below shows the mapping from HotSync Manager versions to Sync Manager API versions.

Palm Desktop Version	HotSync Manager Version	Sync Manager API Version
2.X	2.0	2.1
2.6.3	2.0	2.2
4.0	3.0	2.3

For more information about version compatibility, see the *C/C++ Sync Suite Reference for Macintosh*.

CDK/Handheld Compatibility


You can use this CDK to develop conduits for all Palm Powered handhelds, including those developed by Palm OS licensees and OEM partners.

Conduit Development Environments

The principal development environment for developing conduits is Metrowerks CodeWarrior Release 7.2 for Macintosh. If you are using the conduit development kit, you must use this environment.

Conventions Used in this Document

This guide uses the following typographical conventions:

This style...	Is used for...
fixed width font	Code elements such as function, structure, field, bitfield.
bold	Emphasis.
<u>blue and underlined</u>	Hot links.
	New functions added since the last release of the CDK.
-->	Parameter is passed in to a function.
<--	Parameter is passed out of a function.
<-->	Parameter passed in and out of a function.

Additional Resources

- Documentation

PalmSource publishes its latest versions of this and other documents for Palm OS developers at

<http://www.palmos.com/dev/support/docs/>

- Training

PalmSource and its partners host training classes for Palm OS developers. For topics and schedules, check

<http://www.palmos.com/dev/training>

- Knowledge Base

The Knowledge Base is a fast, web-based database of technical information. Search for frequently asked questions (FAQs), sample code, white papers, and the development documentation at

<http://www.palmos.com/dev/support/kb/>

Conduit Quick Start

This chapter provides a quick start guide to developing conduits with the Conduit Development Kit, without attempting to explain the conceptual underpinnings. You can use this guide if you are already familiar with programming for the Palm OS[®] platform and how to use HotSync[®] technology.

A conduit is a Code Fragment Manager plug-in module that provides a translation bridge between a Palm Powered[™] handheld application and a particular desktop application. You create conduits using the Metrowerks CodeWarrior development environment. You must also have a Palm Powered handheld available for testing your conduit.

For more information about developing a conduit, see the other chapters in this book. For information about the Sync Manager API and the conduit entry points, see the *Conduit Developer Reference*.

Conduit Creation Fast Track Steps

Follow the five steps listed here to quickly create a functional conduit.

A. Install Conduit Development Tools

You need to perform the following installations on your Macintosh desktop computer:

1. Install Metrowerks CodeWarrior. You can use Metrowerks CodeWarrior Release 7.2 for Macintosh to develop conduits with this kit.
2. Install the Conduit Development Kit (the CDK).

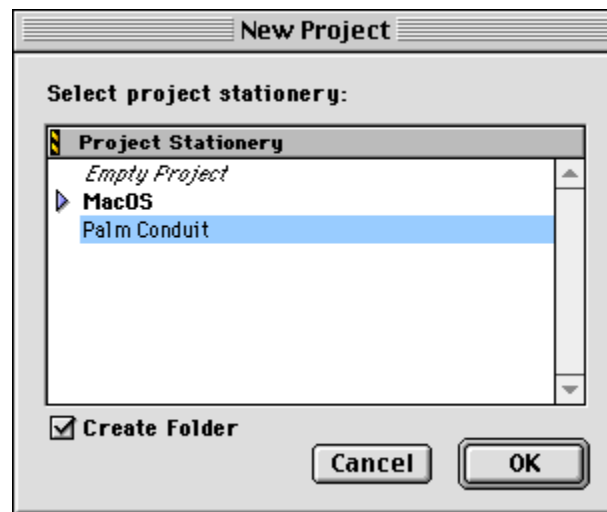
NOTE: You can use the CDK regardless of whether the Palm Desktop software is installed on your Macintosh.

B. Create a New Conduit Project

When you install the Conduit Development Kit, the conduit stationery is added to your Metrowerks installation. To use the stationery, follow these steps:

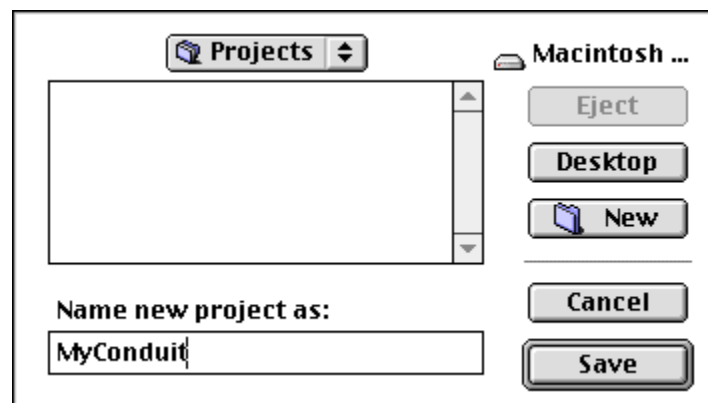
1. Select **New** from the File menu, click on the **Projects** tab, and select the *Palm Conduit* stationery, as shown in [Figure 1.1](#).

Figure 1.1 Using the conduit stationery



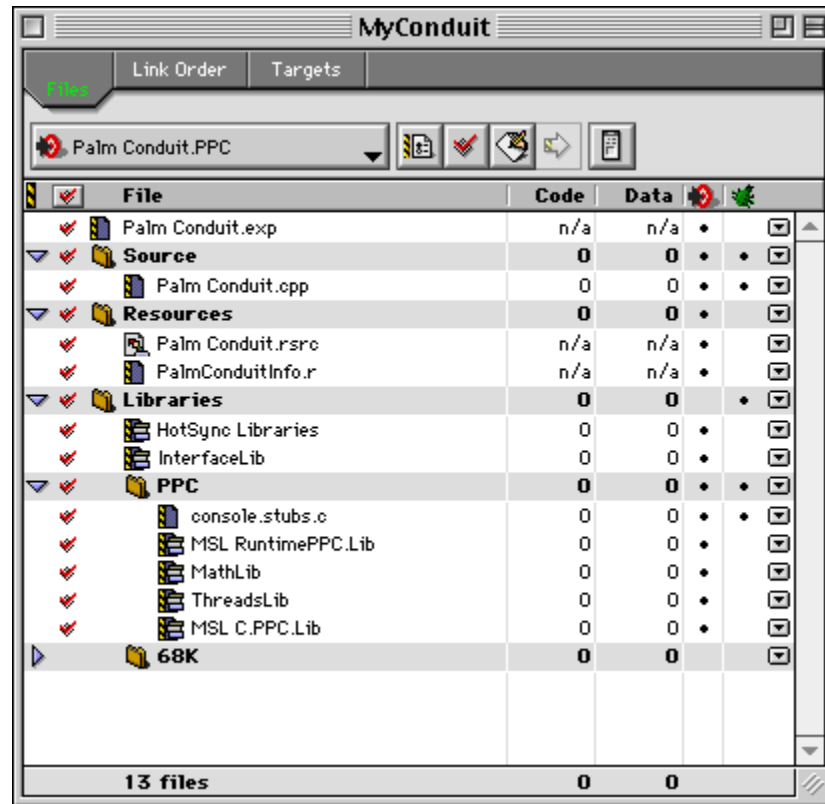
2. Name your project and specify the folder in which you want it saved, as shown in [Figure 1.2](#).

Figure 1.2 Naming your project



3. Press the **Save** button to create your project. Metrowerks displays the project window with your newly created conduit files, as shown in [Figure 1.3](#).

Figure 1.3 The conduit project window



C. Take a Trial Run

Test drive your conduit shell to verify that you can build and install it, and that the HotSync Manager application can run it:

1. Select **Make** from the CodeWarrior Project menu to build a release version of the conduit generated from the stationery without making any changes to the project or source files.
2. You may need to drag your conduit executable into the Conduits folder in the Palm folder on your system, as described in the following note.

Conduit Quick Start

Conduit Creation Fast Track Steps

IMPORTANT: When the HotSync Manager application performs a synchronization, it runs each conduit in the Conduits folders under each of the supported domains. See “[Conduit Domains](#)” on page 114 for details.

When you build a conduit using one of the projects supplied with the CDK, your conduit executable is stored in the `Conduits` folder contained within the CDK directory tree, not in the `Conduits` folder of the Palm Desktop software, of the CDK's `HotSync` folder. The conduits that you build will only run during a HotSync operation if they are located in the correct folder.

The CDK installer places an alias to the CDK target build folder into the system conduit folder, so that build conduits will be found by the HotSync Manager. This is a development configuration only.

If you have not installed the Palm Desktop software, this isn't an issue: your conduits will automatically run when you press the HotSync button on your cradle, provided that you have started the Transport Monitor.

3. Run your conduit by performing a standard synchronization operation. Check the log file to verify that your conduit ran properly.

D. Customize the Conduit Information Resource

The conduit stationery provides a default version of the conduit information ('CInf') resource. You need to modify some of the fields in this resource for your conduit:

1. Change the name and version number of your conduit, if required.

2. Change the creator ID for your conduit to the creator ID that you have obtained from PalmSource, Inc. The creator ID must be unique in order for synchronization to run properly.
3. Change the user interface field to match your configuration code:
 - if you provide a modal dialog box in your `ConfigureConduit` function, set this field to `wantsUserInterface`.
 - if you do not provide a modal dialog box, set this field to `doesntWantUserInterface`.
4. Change the names of the handheld and desktop computer databases to the ones used by your conduit.

For more information about the conduit information resource, see [Chapter 9, “Installing Conduits,”](#) on page 111.

E. Customize the Conduit Entry Points

The conduit stationery provides simple implementations of the conduit entry point functions. You need to add code to several of these functions:

1. Add code to the `GetActionString` function to return the string that the HotSync Manager application displays in the “Next HotSync action” field of the Actions dialog box.
2. If your conduit provides a user interface, you need to add code that displays and handles a modal dialog box in your conduit’s `ConfigureConduit` function. The template conduit program includes a model implementation of the `ConfigureConduit` function.

Note that you must change the user interface flag in your ‘CInf’ resource to `doesntWantUserInterface` if you do not provide a dialog box and to `wantsUserInterface` if you do provide a dialog box.

3. You need to add code to the `OpenConduit` entry point, which is where the interesting work of your conduit is done. The HotSync Manager application calls this entry point to have your conduit exchange data between the desktop computer and handheld.

Conduit Quick Start

Conduit Creation Fast Track Steps

Use the Sync Manager API functions in your `OpenConduit` implementation. Most conduits perform steps such as the following:

- a. Open the database on the handheld and open the data source on the desktop computer.
- b. Use the Sync Manager API to read data from the handheld, and use the desktop application's native API to read data from the desktop data source.
- c. Exchange or synchronize the data as required.
- d. Write the resulting data back to the handheld database and desktop data source.
- e. Clear the handheld's record status flags.
- f. Close the handheld database and the desktop data source.

The Generic Conduit test conduit source code included with the CDK includes multiple targets in its project file. You can choose to build either a bundled or non-bundled version of the conduit by simply choosing the appropriate target ("`GenericConduit(bundle)`" or "`GenericConduit`") before building.

The Generic Conduit provides an examples of `OpenConduit` and `OpenConduitCarbon` implementations that instantiates and passes control to a `CSynchronizer` object. The template conduit program includes an example of the `OpenConduit` function that calls Sync Manager API functions.

For debugging and troubleshooting assistance, see [Chapter 10, "Debugging Conduits,"](#) on page 117.

For more information about the conduit entry points, see [Chapter 1, "Conduit API,"](#) on page 1 in the *C/C++ Sync Suite Reference for Macintosh*, or [Chapter 7, "Interfacing with the HotSync Manager Application,"](#) on page 63.

Conduit Design Goals

HotSync technology has been a very important part of the success of the Palm OS platform. One of the reasons for this is that the conduits called by the HotSync Manager application run quickly and adhere to a strong set of design goals. You need to develop your conduit with these same goals in mind, as described in [Table 1.1](#).

Table 1.1 Conduit design goals

Design goal	Description
Fast execution	The overarching goal for the HotSync process is that a complete synchronization happen very quickly. Conduits need to be designed for optimal processing speed and minimal data transfer.
Zero data loss	Conduits must take measures required to prevent data loss under any circumstances, including loss of connection during a synchronization process.
Good conflict handling	Conduits that perform mirror image synchronizations must be able to gracefully handle conflicting modifications. This occurs when the user modifies a record on both the desktop computer and the handheld. In addition to managing the conflict, the conduit should add an entry to the log to notify the user of this situation.
No user interaction	The user expects to be able to press the HotSync button on the cradle and have synchronization proceed without any required interaction. Conduits need to be constructed to avoid the need for user interaction.

Conduit Development Basics

This chapter provides an overview of how different software and hardware components fit into the world of software development for the Palm OS® platform, and how conduits fit with those components.

Palm OS Platform Connections

Users connect Palm Powered™ handhelds and use HotSync® technology to perform the following operations:

- to synchronize data stored on the handheld with data stored on the desktop computer
- to back up data stored on the handheld to the desktop computer
- to install new handheld applications that have been stored on the desktop computer

The handheld features a processor that is significantly slower than the processors on most desktop computers. Because of this, PalmSource encourages developers to write applications for the Palm OS platform that off-load the processor-intensive tasks to the desktop computer. Specifically, handhelds are intended for applications such as the following:

- portable data entry
- portable data viewing
- remote transactions

Conduit Development Basics

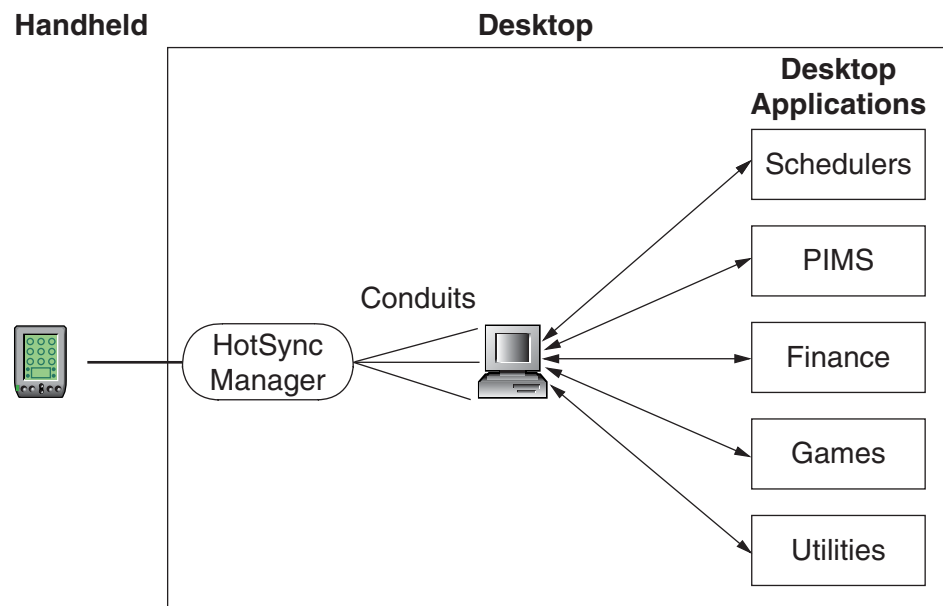
Palm OS Platform Connections

And applications that perform the following operations are considered more suitable for the desktop computer's processing power:

- high volume data entry
- backing up data
- printing
- configuration
- data storage

[Figure 2.1](#) shows the relationship between the desktop computer and handheld.

Figure 2.1 Palm OS platform connections



The HotSync transport component manages communications between the desktop computer and the handheld. This is true regardless of how the two are communicating. There are three communication connection types available:

- direct cable connection
- modem connection
- network connection

The software on the desktop computer that communicates with the handheld is the HotSync Manager application. This program is described in [Chapter 3, “How the HotSync Manager Application Works with Conduits,”](#) on page 21. The HotSync Manager application uses the communications API to handle the actual sending and receiving of bytes to and from the handheld, which makes the HotSync Manager application independent of the connection type. And since conduits use the HotSync transport to send and receive data, conduit code is completely independent of the communication connection type.

Software Components

There are four software components involved in using a handheld:

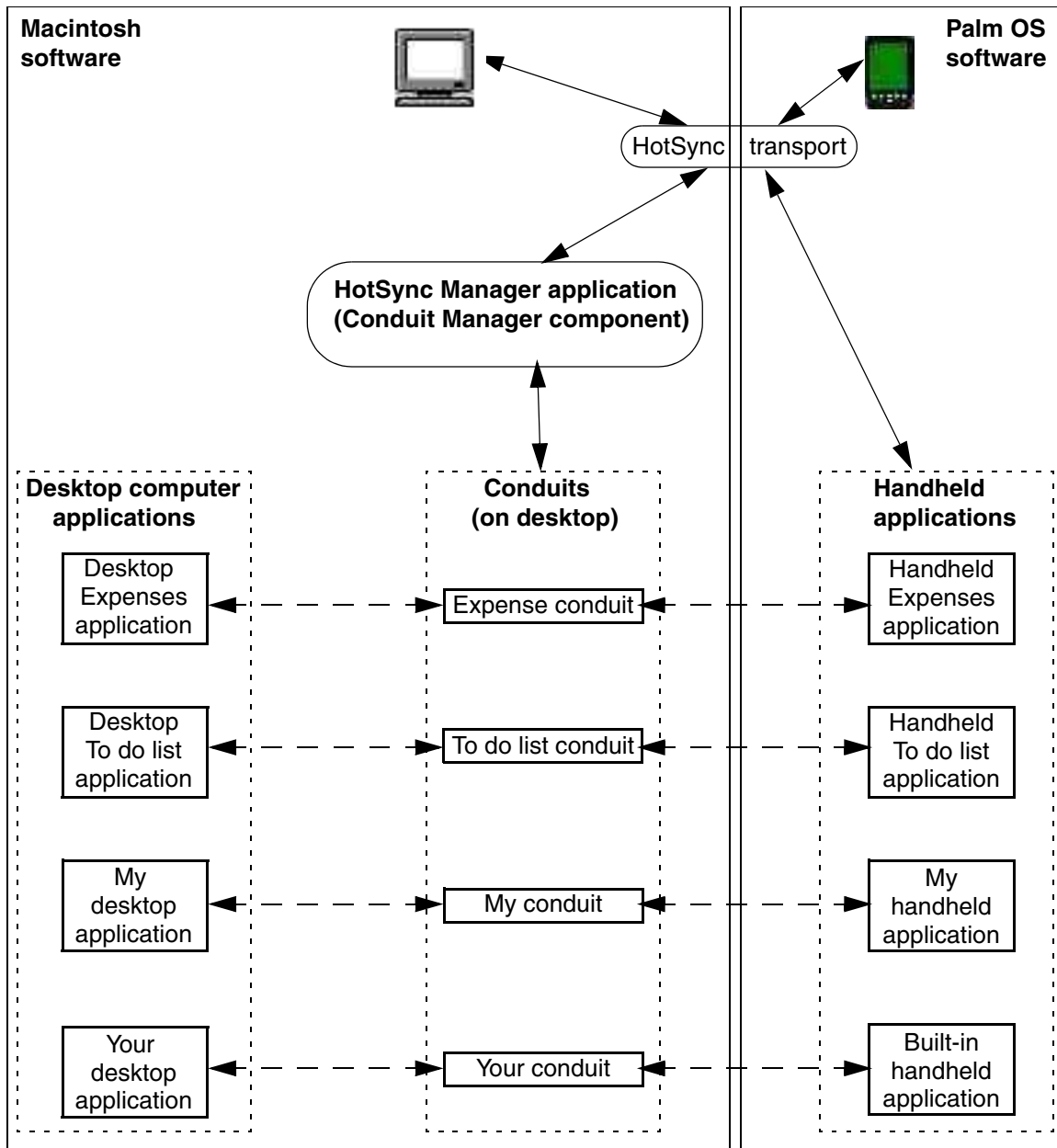
- *Desktop applications*, which are developed by you or another developer, run on the desktop computer and operate on data that is sent to and/or retrieved from the handheld.
- Palm OS applications developed by Palm OS developers that run on the handheld. These are also referred to as *handheld applications*.
- The *HotSync Manager application*, which is supplied by PalmSource, Inc., runs on the desktop computer and communicates with the handheld. The HotSync Manager application's Transport Monitor component awakens when the user initiates synchronization operations, and passes control to the Conduit Manager component, which calls each of the conduits that have been loaded onto the user's desktop computer.
- *Conduits* are the shared libraries that the HotSync Manager application activates to interact with specific data on the handheld. You use the Conduit Development Kit to create conduits.

Conduit Development Basics

Software Components

[Figure 2.2](#) shows the process flow through these components.

Figure 2.2 Palm OS platform process flow



Desktop Applications

Desktop applications of various kinds can exchange data with handhelds. These are standard desktop computer applications that either share data with a handheld application or have a data entry component that runs on the handheld. For example, the Palm Desktop software shares date book, address, memo, and to-do data with the built-in handheld applications.

Handheld Applications

Handheld applications are either the built-in applications or third party programs that have been installed on the handheld.

Developers create applications for Palm Powered handhelds using various development environments, as described in PalmSource developer documentation, including the *Palm OS Programmer's API Reference* and *Palm OS Programmer's Companion* books.

The HotSync Manager Application

The HotSync Manager application oversees the process of synchronizing a handheld with a desktop computer. The HotSync Manager application runs in the background on the user's computer, monitoring a communications port for the signal to begin synchronizing. When the HotSync Manager application receives that signal, it initiates the synchronization process and then calls the conduits that have been installed on the user's desktop computer. Each conduit performs its own synchronization operations.

IMPORTANT: The ability to quickly synchronize data between the desktop computer and the handheld is fundamental to the Palm OS platform philosophy.

For more information about the HotSync Manager application, see [Chapter 3, "How the HotSync Manager Application Works with Conduits,"](#) on page 21.

Conduits

A conduit is a plug-in module for the HotSync Manager application that transfers a specific kind of data between the handheld and the desktop computer.

Some applications create or use data that must be shared with data on a desktop computer. For example, the built-in date book application exchanges data with the Palm Desktop software that runs on the user's desktop computer. The date book conduit synchronizes the date book databases on the desktop computer and handheld.

Other applications create data that is backed up onto the desktop computer. The backup conduits copies the data from the handheld to the user's desktop computer.

IMPORTANT: Conduits must operate without user interaction so that users can press the HotSync button on the handheld cradle and have data synchronized without any required intervention. This is especially important for users who are synchronizing remotely and cannot interact with the desktop computer.

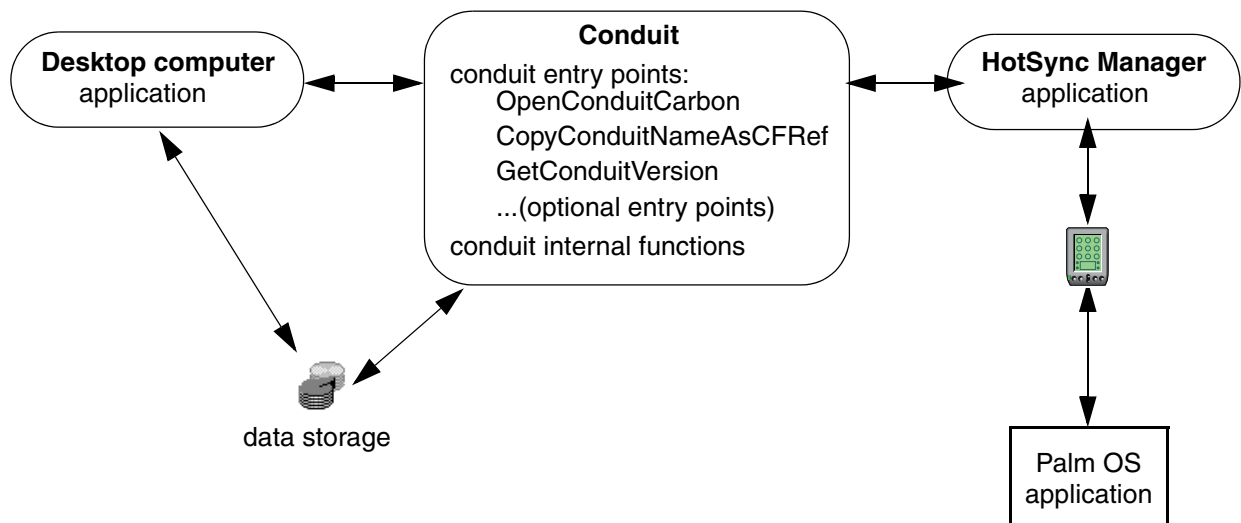
For more information about conduits, see [Chapter 4](#), “[Developing Conduits](#),” on page 31.

Conduit Development

You can develop conduits with the Palm CDK for Macintosh in C/C++ with Metrowerks CodeWarrior running on Mac OS 9 or Mac OS X. (See “[Conduit Development Environments](#)” on page xiii for supported CodeWarrior versions.)

[Figure 2.3](#) shows the software components involved with developing a conduit.

Figure 2.3 Synchronization software components



Conduit Entry Points

The HotSync Manager application calls several entry points in a conduit, including the `OpenConduit` function, which allows the conduit to perform its tasks. A conduit must provide some of these entry points and can optionally provide others.

For more information about the required and optional entry point functions, see [Chapter 7, “Interfacing with the HotSync Manager Application,”](#) on page 63.

The Conduit Information Resource

Macintosh conduits use a conduit information (`'CInf'`) resource to specify certain conduit characteristics, including the name and version number of the conduit. You need to include a conduit information resource for your conduit. This resource is described in [Chapter 9, “Installing Conduits,”](#) on page 111.

The Sync Manager API

The Sync Manager API is the programmatic interface for communicating with the handheld. Conduits call functions in this API to send data to and receive information from the handheld.

For more information about the Sync Manager API, see [Chapter 6](#), “[Using the Sync Manager API](#),” on page 57.

The Generic Conduit Framework

Rather than relying directly on the Sync Manager API, you can develop your conduit using the Generic Conduit Framework (GenCn). GenCn is a generic conduit, based on C++ classes that you can override to implement your own functionality.

For more information about Generic Conduit, see [Chapter 5](#), “[Generic Conduit Framework](#),” on page 39.

The User Manager API

If you’re developing a desktop application that needs to look up or change information about HotSync users, such as their name, synchronization settings, and so forth, you can use the User Manager API to accomplish these tasks.

For more information about the User Manager API, see [Chapter 6](#), “[User Manager API](#),” on page 233.

Carbonization and Mac OS X

CDK 4.0.3 supports development of conduits that will work with HotSync Manager on both Mac OS 9 and Mac OS X. To do this, your conduit must be written to Apple’s Carbon API. Carbon is a subset of the classic Mac OS API that is available for both Mac OS 9 and Mac OS X applications. For the most part, the functions eliminated in Carbon are obsolete and redundant.

CarbonLib

Developing conduits requires linking against Apple’s CarbonLib 1.5 or later. You can download the latest version of CarbonLib at:

<http://developer.apple.com/sdk/index.html>

Updating an Existing Conduit

Updating an existing conduit to work with HotSync Manager 3.0 is a fairly straightforward process.

Carbonizing

The first step to updating (“Carbonizing”) an existing conduit to work under both Mac OS 9 and Mac OS X is to download and use Apple’s Carbon Dater utility, which will examine your existing conduit and create a report indicating what changes are needed in order to make it Carbon-compliant.

Carbon Dater and the corresponding documentation can be found at:

<http://developer.apple.com/carbon/dater.html>

Once you’ve identified which Mac OS functions you’re using that are no longer supported in Carbon, you can update your code.

NOTE: Your conduit must have either a 'carb' or 'plst' resource of ID 0 in order to be recognized as a Carbonized conduit. Read Apple Technical Note 2013 at <http://developer.apple.com/technotes/tn/tn2013.html> for details.

Updating the Conduit Info Resource

You also need to update your conduit info ('Cinf') resource to indicate a resource version of at least 0x0300 in order to be loaded by HotSync Manager on Mac OS X.

Updating the User Interface

If your conduit has a dialog box for [ConfigureConduit](#), you should take whatever steps are needed to bring it into line with Apple’s Aqua Human Interface Guidelines.

For details on these guidelines, visit:

<http://developer.apple.com/techpubs/macosx/Essentials/AquaHIGuidelines/index.html>

Conduit Development Basics

Conduit Development

Bundling

If you choose to distribute your conduit as a bundle, you also need to replace your [OpenConduit](#), [GetConduitName](#), and other functions with the new Carbon versions, such as [OpenConduitCarbon](#) and [CopyConduitNameAsCFStringRef](#).

For details on bundling, see *Inside Mac OS X: System Overview*:

<http://developer.apple.com/techpubs/macosx/Essentials/SystemOverview/index.html>

Updating the Installer

Your conduit's installer needs to be updated not to use the Install Aide API, which has been deprecated. Instead, you need to use the User Manager API. See [Chapter 6](#), "[User Manager API](#)," on page 233 for details on this API.

Bundles

A bundle is a directory following a particular structure that groups executable files and resources together into a unit. The resources can include any form of data, including entire text files, graphics files, sounds, and so forth.

The most common use for bundles in a conduit is to provide support for multiple languages and locales. Your conduit can simply include resources for multiple locales in its bundle, and the conduit will automatically be localized to the user's locale when the conduit is loaded.

Details on the structure of a bundle are beyond the scope of this documentation, but the key to localization is to create `Language.lproj` directories within your bundle's Resources directory; each of these directories should contain the resources for the specified language. For example, if your conduit is localized for US English and Japanese, you would include directories called `en_US.lproj` and `Japanese.lproj`.

NOTE: The localization directories in your bundle can have generic names if the region is unimportant (like English.lproj if you don't have separate US and UK English versions). Otherwise, your localization directory names are comprised of the ISO 3166 standard country code, an underscore, and the ISO 639 standard for language code. Visit <http://www.iso.ch> for a complete list of these codes.

The resources contained in your bundle's language resource directories would minimally need to include a localized version of your conduit's user interface .nib file and any text files contained in your bundle (such as help files displayed from within your conduit). Graphics files that contain textual data, such as splash screens, would possibly also be localized and contained within your language-specific resource directories.

For details on bundling, see *Inside Mac OS X: System Overview*:

<http://developer.apple.com/techpubs/macosx/Essentials/SystemOverview/index.html>

Carbon Documentation

Apple provides extensive documentation about Carbon on its web site:

<http://developer.apple.com/techpubs/macosx/Carbon/carbon.html>

Apple's Carbon site provides not only documentation, but tools, technical notes, sample code, and links to purchase printed manuals.

How the HotSync Manager Application Works with Conduits

The HotSync® Manager application uses HotSync technology to enable synchronization of data between handhelds and desktop computers. The HotSync Manager application also performs other actions, including:

- management of individual or multiple handhelds with a centralized desktop computer
- backing up of handheld data onto a desktop computer
- initialization of uninitialized handhelds
- installation of new applications and databases on the handheld
- restoration of handheld data

The *HotSync Manager* application oversees synchronization operations, which are sometimes referred to as the *HotSync process*. The HotSync process automatically synchronizes data between a handheld and a desktop computer by invoking each conduit installed on the desktop computer, and also installs applications and databases onto the handheld.

The HotSync Manager application actually consists of three components:

- The Transport Monitor component is a background application that watches the various supported ports (serial, USB, and PalmConnect, for example) for the wakeup packet from the handheld and then passes control to the Conduit Manager.
- The HotSync Manager component is the executable that provides configuration and logging facilities for

synchronization operations. Once the Transport Monitor detects a HotSync command, the Conduit Manager takes over, determining which synchronization operations need to be run.

- The Conduit Manager component manages the synchronization process and the execution of the individual conduits.

This chapter uses the term “the HotSync Manager application” to refer to these three components as a single entity.

Terminology

The [Glossary](#) provides definitions for a number of terms that are used in this book. To understand this chapter, you must understand the following glossary terms:

Term	Definition
creator ID	The unique ID associated with each database and application on the handheld. Each conduit is associated with a specific creator ID.
PC ID	A pseudo-random number that the HotSync Manager application generates and uses to identify a desktop computer.
user ID	A unique user ID associated with each handheld. This is a pseudo-random number generated by the HotSync Manager application during the handheld’s initial synchronization after a complete reset, or during its first-ever synchronization process.

HotSync Operations

After the user initiates a HotSync Manager operation, which is usually done by pressing the HotSync button on the handheld cradle, the HotSync process initializes itself and performs the sequence of operations described in [Table 3.1](#).

Table 3.1 HotSync operations sequence

User action or display	HotSync Manager actions	Handheld actions
1. User presses HotSync button	<p>Reads the user ID for that user on the desktop computer and validates the user.</p> <p>Locates the data for the user on the desktop computer.</p> <p>Reads system and state information from the handheld.</p>	HotSync Manager process starts on the handheld and sends a signal to the desktop computer.
2. "Connecting with desktop"	Retrieves the list of databases and applications to be processed from the handheld.	
3. "Synchronizing..."	<p>Calls the Install conduit to install any applications or databases that have been stored on the desktop computer for that purpose.</p> <p>Runs each conduit in the <code>Conduits</code> folder by calling specific entry points, including: <code>GetConduitName</code>, <code>OpenConduit</code> or <code>OpenConduitCarbon</code>.</p> <p>Calls the Backup conduit to copy any databases whose backup bit is set.</p>	
4. Completion message(s)	Updates synchronization information, including the sync time and user ID, on the handheld.	Notifies handheld applications whose databases were modified.

Determining the Synchronization Configuration

This section describes the first step of the synchronization process, which begins after the user has activated a HotSync operation. The HotSync Manager application needs to determine certain synchronization properties, including the following:

- what kind of synchronization (fast or slow) to perform
- which databases on the handheld require backing up

The HotSync Manager uses the user ID and list of database creator IDs from the handheld to determine these properties.

At the start of synchronization operations, the HotSync Manager application compares ID information on the desktop computer with ID information on the handheld to determine if the two were most recently synchronized with each other. If that is true, then a fast synchronization can be performed, as described in “[Fast and Slow Synchronizations](#)” on page 28.

The PC ID

At the start of synchronization operations, the HotSync Manager application retrieves the PC ID of the desktop computer. The PC ID is a pseudo-random number that the HotSync Manager application generates and uses to identify the desktop computer.

Each version of the HotSync Manager application generates its own unique PC ID on the desktop computer.

The User ID

Each handheld has a unique user ID associated with it after it has been synchronized. The user ID is a pseudo-random number that is generated by the HotSync Manager during its initial synchronization after a complete reset, or during its first-ever synchronization process. Note that the same user ID is not generated for the same user name on two different handhelds.

The User ID is stored on the handheld by the HotSync Manager application, which maintains a database on the desktop computer of the user ID of each handheld with which the computer has been synchronized. This user database also tracks the most recent synchronization date for the desktop computer.

Database Creator IDs

Each database and each application on the handheld has a unique creator ID associated with it. Information about each application and/or database is stored on the desktop computer and associated with this creator ID.

The HotSync Manager application uses the creator ID to associate databases with the most appropriate conduits for synchronizing them.

If you are developing a Palm OS® application, you need to make sure that you use a unique creator ID. To select and register your ID, visit the following location on the Internet:

<http://www.palmos.com/dev/support/creatorid/>

Determining Which Conduits to Run

The Macintosh version of HotSync Manager runs every conduit in the Conduits folder with unique creator IDs, unlike the Windows version, which only runs a conduit when there is a database matching its creator ID.

Running Conduits

To run each conduit, the HotSync Manager application's Conduit Manager component calls entry points in the conduit. Each conduit can implement a number of these functions. Each conduit *must* implement these four entry points:

- the [GetConduitName](#) (or [CopyConduitNameAsCFStringRef](#), if your conduit is bundled) function retrieves the name of the conduit to display to the user
- the [GetConduitVersion](#) function retrieves the version number of the conduit
- the [OpenConduit](#) (or [OpenConduitCarbon](#), if your conduit is bundled) function performs the synchronization activities of the conduit

After [OpenConduit](#) or [OpenConduitCarbon](#) returns control, the conduit is done with its processing.

How the HotSync Manager Application Works with Conduits

HotSync Operations

For more information about the conduit entry points that the HotSync Manager application calls, see [Chapter 7, “Interfacing with the HotSync Manager Application,”](#) on page 63.

Running the Install Conduit

To install a new application or database (an *installable*) on a handheld, the user can select the **Install** command from the HotSync Manager application’s HotSync menu. When a user selects a file to be installed, this command copies the installable files to a specific location in the user’s directory; for example, .prc and .pdb files are copied to the conduits to install directory for the HotSync user; see the [UmGetFilesToInstallFolderSpec](#) function to locate this directory.

When a device is synchronized, every install conduit is run.

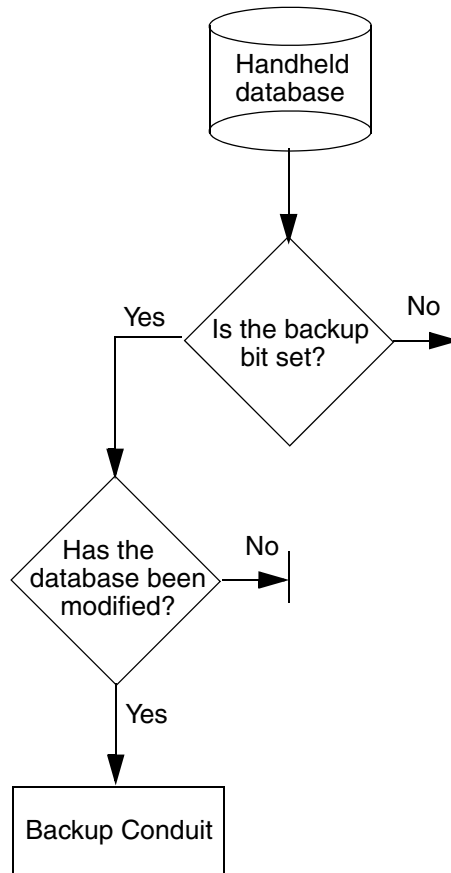
Running the Backup Conduit

The HotSync Manager runs the Backup conduit after the other conduits have completed their operations. This conduit handles any databases marked for backing up that have not been handled by another conduit. [Figure 3.1](#) shows the logic used to determine if a handheld database needs to be backed up by the Backup conduit.

The Backup conduit copies a database from the handheld to storage on the desktop computer. If you are developing a Palm OS application for which there is no desktop computer component, the Backup conduit may well handle your needs, meaning that you do not need to develop your own conduit.

There are several classes of applications that use data that is not manipulated on the desktop computer; these applications are good candidates for using the Backup conduit. For example, a handheld game program that maintains a high score database can rely on the Backup conduit to copy that data. Similarly, a utility program that adjusts settings on the handheld, or an electronic book that the user can view on the handheld. All of these application types can use the Backup conduit rather than a custom developed conduit.

Figure 3.1 Determining if a database is handled by the Backup conduit



The Backup conduit also backs up any handheld applications or .prc files that have their backup bit set, which is useful when restoring a handheld on which a hard reset has been performed.

IMPORTANT: PalmSource strongly recommends that you set the backup bit for all applications and data, with the exception of data that you do not want restored. Note that if you set the backup bit of a .prc file, the Backup conduit will only back up the file when necessary; a performance penalty is not unnecessarily incurred.

Finishing the Synchronization

After running all of the conduits, the HotSync Manager application finishes synchronization activities and the operating system notifies each handheld application whose database(s) have been modified during the synchronization. The applications can respond to this notification as required.

Finally, the HotSync Manager application displays a completion message to the user. If any of the conduits added messages to the log, the HotSync Manager displays a notification message to the user, who can then examine those log entries.

Fast and Slow Synchronizations

The HotSync process is designed for maximum efficiency, which means minimizing the data that needs to be transferred whenever possible. The HotSync Manager application accomplishes this by using two forms of record-level synchronization:

- **Fast sync**, or fast synchronization of records, which only sends data between the desktop and handheld if a change has occurred since the most recent synchronization. Fast sync does not handle records that have not been modified.
- **Slow sync**, or slow synchronization of records, which is used in situations in which fast synchronization is not possible. This happens when the record status flags on the handheld cannot be reliably used because the desktop computer and handheld were not most recently synchronized with each other. Slow sync compares each record in the handheld database with the corresponding record on the desktop computer to determine how to synchronize the records.

The HotSync Manager application automatically determines which synchronization form is appropriate, and communicates this to each conduit in the `CSyncProperties` structure that it passes to the conduit in [OpenConduit](#) or [OpenConduitCarbon](#).

For more information about synchronization of individual records, see [Chapter 4](#), “[Developing Conduits](#),” on page 31.

Determining If Fast Sync is Possible

The HotSync Manager application retrieves the user ID and the PC ID of the most recent synchronization from the handheld. It compares these values with the values stored on the desktop computer. If the values match, then conduits can perform a fast sync, meaning that they can depend on the modification flag in each record to determine if the record requires synchronizing.

When the HotSync Manager application invokes a conduit, it passes a synchronization properties structure to the conduit's [OpenConduit](#) or [OpenConduitCarbon](#) function. This structure specifies whether the conduit should perform a fast or slow synchronization.

Note that there are some situations in which all of the records in an application's database must be synchronized, even if they have not been changed since the last time synchronization:

- If a database has been deleted on either the handheld or desktop computer, all of the records in the surviving database must be synchronized.
- When a new application is installed on the handheld, all of the corresponding data records must be synchronized.

Database Modification Information

Databases on the handheld include modification information that HotSync Manager uses to improve the efficiency of synchronization operations:

- a database modification date, which indicates when the database was last modified
- a record modification flag for each record, which indicates whether the record has been changed since the most recent synchronization

If the HotSync Manager application determines that it can perform a fast sync, conduits only need to work with records that have been changed.

HotSync Connections

The HotSync Manager application allows three different kinds of connection between a handheld and a desktop computer. The HotSync Manager application handles each type of connection differently, as described in [Table 3.2](#).

Table 3.2 HotSync connection types

HotSync connection type	HotSync activation description
Direct cable using the handheld cradle	The handheld cradle cable is connected to the desktop computer. When the user presses the HotSync button on the cradle, two pins are shorted on the handheld. This causes the Palm OS to activate the HotSync client application on the handheld, which “awakens” the HotSync Manager application on the desktop computer.
Modem	The handheld is connected to a modem that is dialed into a desktop computer. The user sets up HotSync Manager on the desktop computer to use a modem, and then selects Modem Sync in the HotSync client application on the handheld. The handheld dials into the desktop computer and connects with the HotSync Manager program.

Developing Conduits

This chapter provides an overview of conduit development. For more information about the functions and data structures that you use to develop conduits, see the [C/C++ Sync Suite Reference for Macintosh](#).

About Conduits

A conduit is a plug-in module for the HotSync® Manager application that transfers data between the handheld and the desktop computer. The HotSync Manager application runs the conduits that have been installed on the user's desktop computer. Each conduit is designed to synchronize or back up data from specific handheld applications' database or databases.

Some handheld applications create or use data that must be shared with data on a desktop computer. For example, the built-in Date Book application exchanges data with the Palm Desktop application that runs on the user's desktop computer. The Date Book conduit synchronizes the Date Book databases on the computer and handheld.

Other handheld applications create data that is backed up onto the desktop computer. The backup conduit copies the data from the handheld to the user's desktop computer.

Programming Conduits

Each conduit running on a Macintosh system is a standard Carbon Core Foundation shared library. These shared libraries on the Macintosh are similar to DLLs on Windows systems. Your conduit code can perform any actions that any shared library can perform. However, to maintain the integrity of the HotSync process, conduits need to meet certain design goals, as described in "[Conduit Design Goals](#)" on page 34.

Conduit Tasks

Your conduit synchronizes data for a specific application on the handheld with the desktop computer. To do so, your conduit must perform the following tasks:

- open and close databases on the handheld
- add, delete, and modify records on the handheld and desktop computer, converting formats as required
- compare records to support fast sync, as described in “[Fast and Slow Synchronizations](#)” on page 28.

Since each application stores data in its own format, each conduit has to use custom data handling and conversion algorithms. To work with data on the handheld, you need to use the Sync Manager API, as described in [Chapter 6](#), “[Using the Sync Manager API](#),” on page 57.

Conduit Synchronization Types

The form of synchronization that you choose for your conduit depends on the nature of the applications involved and how you want to handle their data. The synchronization types are summarized in [Table 4.1](#).

Table 4.1 Conduit synchronization types

Synchronization type	Description	Notes
Mirror-image	<p>Ideal for applications that run on both platforms and allow modifications on both; makes data identical on both platforms.</p> <p>Example applications for this kind of synchronization include Address Book and Date Book applications.</p>	<p>Your conduit must provide conflict resolution solutions when the same record has been modified on both platforms. This must occur without user intervention.</p>

Table 4.1 Conduit synchronization types (*continued*)

Synchronization type	Description	Notes
One directional	For applications that run on both platforms but only allow data modification on one; copies the data to the other platform An example application for this kind of synchronization is a stock quotation program that updates data on the desktop computer and copies the latest data to the handheld.	Requires less time, and should be used when possible.
Transaction-based	For applications that need to perform additional processing on the desktop computer between record synchronizations. An example of using this kind of synchronization is an e-mail program that uses transaction processing to update the In Box on the desktop computer.	Only use this when required, because it slows down the HotSync process.
Backup	For applications that don't have desktop computer components; simply copies the handheld data to the desktop computer.	Can use the Backup conduit provided by PalmSource, Inc.

NOTE: A major design goal for conduits is to minimize synchronization time. This means that you should choose the simplest method of synchronization that can be used for your conduit. The synchronization types in order, from simplest to most complex are: backup, one-directional, mirror-image, and transaction-based.

Conduit Design Goals

HotSync technology has been a very important part of the success of the Palm OS platform. One of the reasons for this is that the conduits called by the HotSync Manager application run quickly and adhere to a strong set of design goals. You need to develop your conduit with these same goals in mind, as described in [Table 4.2](#).

Table 4.2 Conduit design goals

Design goal	Description
Fast execution	The overarching goal for the HotSync process is that a complete synchronization happen very quickly. Conduits need to be designed for optimal processing speed and minimal data transfer.
Zero data loss	Conduits must take measures required to prevent data loss under any circumstances, including loss of connection during a synchronization process.
Good conflict handling	Conduits that perform mirror image synchronizations must be able to gracefully handle conflicting modifications. This occurs when the user modifies a record on both the desktop computer and the handheld. In addition to managing the conflict, the conduit should add an entry to the log to notify the user of this situation.
No user interaction	The user expects to be able to press the HotSync button on the handheld cradle and have synchronization proceed without any required interaction. Conduits must never require user interaction.

Conduit Development Basics

The easiest way to create a new conduit is to create a new project based on the conduit stationery that is installed for CodeWarrior when you install the CDK.

To customize the stationery conduit into your own version, you must perform the following minimal steps:

- modify the `GetConduitName` function (for non-bundled conduits) or `CopyConduitNameAsCFStringRef` function (for bundled conduits) to return the name of your conduit
- modify `GetConduitVersion` to return the version number of your conduit
- add code to the `OpenConduit` function (for non-bundled conduits) or `OpenConduitCarbon` (for bundled conduits) to perform your actual synchronization operations
- modify the `ConfigureConduit` function to allow the user to perform configuration of your conduit
- modify the `GetActionString` function (for non-bundled conduits) or `CopyActionStringAsCFStringRef` (for bundled conduits) to return the string that the HotSync Manager application displays in the “Next Action” dialog box

You must also create a Conduit Information (`'CInf'`) resource that specifies information about your conduit. For more information about the conduit information resource, see [Chapter 9, “Installing Conduits,”](#) on page 111.

NOTE: Your conduit must have either a `'carb'` or `'plst'` resource of ID 0 in order to be recognized as a Carbonized conduit. Read Apple Technical Note 2013 at <http://developer.apple.com/technotes/tn/tn2013.html> for details.

You can also refer to the source code for the stationery conduit that is provided with the CDK. The stationery conduit source code implements the basic conduit functions.

Many conduit developers find that the best way to develop their conduits is to examine and borrow code from the example conduits that are provided with the Conduit Development Kit.

Conduit Design Decisions

This section provides information that you can use to make decisions regarding the top-level design of your conduit.

Conduit Design Questions

The most important considerations when designing your conduit are:

- how the data is stored on both platforms
- whether synchronization will be one directional, mirror image, or transaction based

Your answers to the following questions will help determine the type of conduit you need to implement:

- Do the existing desktop computer databases offer a public application programming interface for record I/O? If so, is this API packaged as a stand-alone library?
- Does each record have a unique Record ID field? If so, can the conduit perform keyed look-ups by Record ID? If so, can the conduit re-assign the Record ID to a record? Is the record ID format on the desktop computer compatible with the record ID format on the handheld? PalmSource strongly recommends that record IDs only get assigned on the handheld.
- Does each record maintain a status flag field? If so, are the available status values Add, Modify, and Delete? How will you clear the status flags at the end of synchronization operations?
- Is there an easy way to detect modified records? A time stamp indicating the last modified date (with hours, minutes, and seconds) will work for this purpose.
- Is the concept of categories supported in the existing databases? If so, what is the limit on number of categories allowed? Will category information be synchronized? How will the desktop categories be mapped to the categories on the handheld?

- Is the possibility of downloading only a subset of field data to the handheld acceptable? If so, consider how to retain that data on the desktop computer during the synchronization process.
- Will the conduit need to synchronize more than one database on the handheld to one or more database(s) on the desktop computer (multi-database synchronization)?
- Is one-way database copying appropriate?
- Will the conduit behave differently during a fast sync than during a slow sync?
- Is the data to be synchronized at the record level, so that each record that has changed is updated on both sides?
- Will the conduit need to synchronize one database on the handheld to one database on the desktop computer (single database synchronization)?
- How well do the desktop computer record fields and handheld records map to each other?
- Does the third party application support the concept of multiple databases or multiple users?
- Will archiving be supported?
- Can you pre-process data for efficiency, to keep the synchronization time to a minimum, and to avoid timeouts? For example, if your conduit accesses information over the Internet, it is best to access that information prior to beginning the synchronization.

Generic Conduit Framework

This chapter provides information about the Generic Conduit class framework for developing conduits.

For details about the classes and data types comprising the Generic Conduit Framework use the Apple Help Viewer program to read the “Generic Conduit Help” topic; this is installed for you when you install the CDK.

Introduction to the Generic Conduit Framework

The Generic Conduit Framework — called simply *GenCn* (“gen-con”) in this chapter — provides a streamlined approach to conduit development, which can be used to develop a wide variety of conduits quickly.

Why Use GenCn?

GenCn makes it easy to synchronize anything from a text file to a large enterprise database on the desktop with an application on the handheld. GenCn does not rely on any platform-specific API or data formats for database storage the way Palm[™] Desktop software’s PIM applications do, and so is ideal for Macintosh/Windows cross-platform development.

Its design allows you to focus directly on your application. Essentially, you provide an interface to your database using the classes supplied by the GenCn Framework, and the framework does the rest, managing the synchronization and transfer of records between the handheld and desktop applications.

Generic Conduit Framework

Introduction to the Generic Conduit Framework

The principal features of GenCn are:

- Ideal for cross-platform development (not reliant on platform-specific APIs)
- Provides a high-level conduit architecture
- Full code is supplied and supported

What Will the Generic Conduit Framework Do for Me?

Conduits that simply upload or download databases are relatively simple. They must register and unregister the conduit with the Sync Manager, open and close their databases, read and write records, and manage category information if necessary.

Generic Conduits can do simple synchronization, but can also perform per-record, mirror-image synchronization. They must perform all the steps of the simpler conduits, but must in addition determine which records have been added, deleted, or modified from both the handheld and desktop databases. They must decide what to do about the differences.

Generic Conduits must deal with the fact that record IDs are assigned on the handheld and not on the desktop. Thus, if a new record on the handheld must be copied to a desktop database, the handheld's record ID must be preserved in the desktop database.

Likewise, a record created on the desktop must be copied to the handheld, and the record ID of the new record must be fetched back from the handheld and recorded in the desktop record. Finally, a conduit performing mirror-image synchronization must provide logic for both fast and slow synchronization modes. The HotSync[®] Manager application will inform the conduit in which mode it should synchronize.

The Generic Conduit Framework automates most of this process; all you need to do is replace the default behaviors where appropriate to customize the Generic Conduit to meet your specific requirements.

Cross-Platform Development With GenCn

When developing cross-platform conduits using the Generic Conduit Framework, it's helpful to note that both the Mac and

Windows versions of GenCn use the same source files. GenCn includes separate project files for Visual Studio on Windows and CodeWarrior on the Macintosh.

However, at this time the CDK's headers and libraries are laid out differently on the two platforms, so if you wish to get both the Mac and Windows project files to build from the same sources at the same time, you'll have to have both layouts installed by merging a Mac CDK installation and a Windows CDK installation.

Also keep in mind that the Macintosh version of GenCn doesn't include all the sample classes for manipulating Palm Desktop databases that are included with the Windows CDK, since the formats of the Mac desktop databases are not documented.

The Mac CDK requires that the `Headers` and `Libs` folders be in the same directory as the `Conduits` folder. The Windows CDK requires that these folders be called `Include` and `Lib`, and be located in the same directory as the `Samples` folder.

The Macintosh version of the CDK includes the Visual Studio project file uses for building GenCn on Windows.

Generic Conduit Framework Overview

Conduit development using the C++ Generic Conduit Framework (GenCn) consists principally of:

1. Describing the handheld and desktop data formats you wish to use by subclassing and customizing certain C++ classes in the GenCn Framework
2. Providing standard entry points, which allow the HotSync Manager application to call into your conduit

Once these elements have been created, the conduit must be "registered" with the HotSync Manager application so that your conduit may be called when the user wishes to synchronize data; on the Macintosh, this is simply a matter of installing the conduit through normal means.

When invoked by the HotSync Manager application, your conduit will use the GenCn Framework to synchronize the records between the handheld database and your desktop database using the data conversion methods you supplied in step 1. The GenCn Framework

Generic Conduit Framework

Generic Conduit Framework Overview

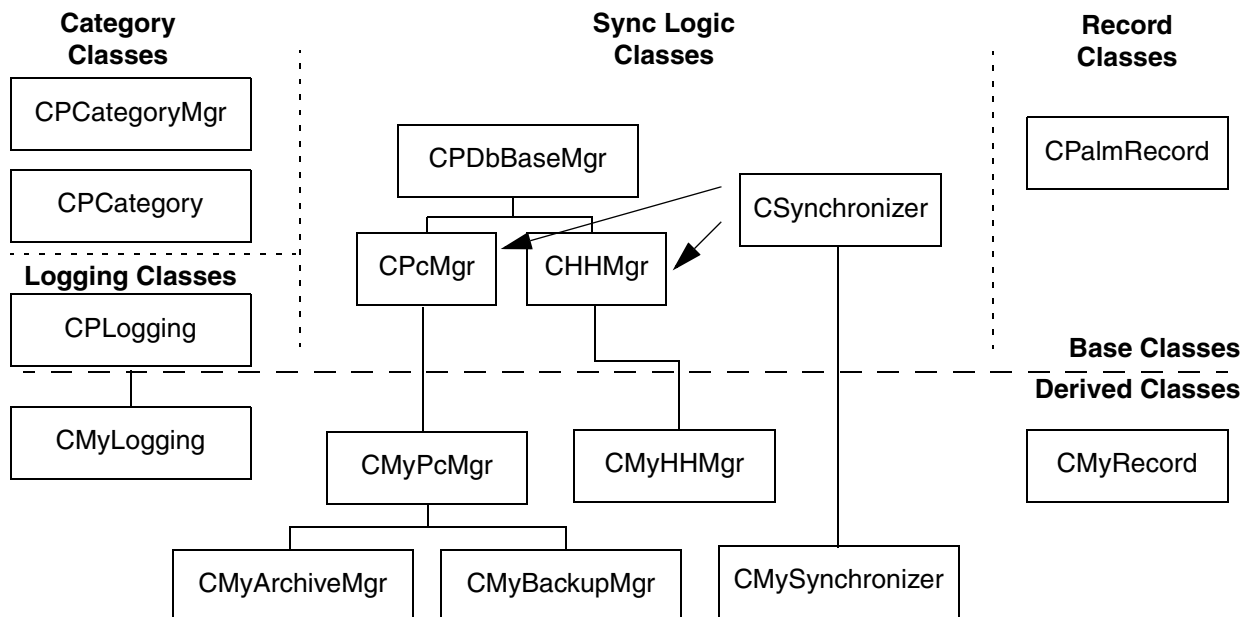
then performs the synchronization using your data conversion methods to compare and update databases.

The GenCn Framework is designed to facilitate these steps. The next section describes the class hierarchy of the GenCn Framework.

GenCn Class Hierarchy

[Figure 5.1](#) illustrates the basic structure of the C++ Generic Conduit Framework.

Figure 5.1 Generic Conduit Framework



The base classes of the GenCn Framework can be grouped as follows:

- *Category* classes access and manage standard Palm OS® category data.
- *Record* classes convert records between the format on the handheld and the format on the desktop computer.
- *Database Manager* classes store and retrieve records from the handheld or the desktop computer.

- *Synchronizer* classes synchronize records and application information (the AppInfo block) from a database on the handheld. The core synchronization logic is implemented in these classes.
- The *Logging* class is used to print diagnostic information into a log file maintained by HotSync Manager.

Developing Data Formats

The heart of GenCn development is subclassing and customizing desktop Database Manager classes to support your database format, and modifying the Synchronizer classes to refer to them. Here is a breakdown of the required steps:

1. Either implement your own custom record class or subclass from the `CPCustomRecord` class. If using the `CPCustomRecord` class, add the data members required for your database records.
2. Subclass methods in your derived `CPcMgr` class for storage and retrieval of your records on the desktop.
3. In your derived `CPcMgr` class, create methods to convert from your custom record class to/from the `CPalmRecord` format. These methods are standardly called "ConvertPCtoGeneric" and "ConvertGenericToPc." Because your records are of a different base class, you will not be subclassing these methods.
4. Implement storage and retrieval of a backup database, an archive database, or both by further subclassing the `CPcMgr` base class.
5. You may need to implement the conversion of records to/from the generic `CPalmRecord` format inside your custom record class.
6. Implement any synchronization of AppInfo data, beyond the Category data.

Synchronization Control Flow

The following steps outline the basic flow of control for a typical synchronization operation when HotSync Manager passes control to a conduit based on the Generic Conduit Framework:

1. HotSync Manager calls your `OpenConduit` or `OpenConduitCarbon` function.
2. Your synchronization entry point instantiates a `CSynchronizer` object, which performs the following tasks:
 - a. Registers the conduit.
 - b. Examines the synchronization properties passed to it to determine the type of synchronization, typically fast sync or slow sync.
 - c. Creates the Database Manager objects (handheld, desktop, archive, and backup).
 - d. Synchronizes category information using the provided synchronization logic.
 - e. Synchronizes application information (`AppInfo` block).
 - f. Synchronizes records using the fast sync, slow sync, HH overwrites desktop, Desktop overwrites HH, or No Action.
 - g. Deletes the Manager objects.
 - h. Logs results to the HotSync log.

These steps are spelled out in greater detail later in this chapter.

Mirror-Image Synchronization

Generic Conduits have the ability to seamlessly perform mirror-image synchronization. This section describes the database management issues involved in this synchronization strategy and outlines how GenCn-based conduits perform mirror-image synchronization.

Mirror-image synchronization produces identical databases on the desktop and handheld. Records can be created, modified, or deleted in either database, and all such changes are reconciled during synchronization so that the databases are mirror images of each other.

Basic Synchronization

Each conduit compares status information contained in its database records on the handheld and the desktop, and uses that information to delete, archive, copy, or replace records from one side to the other as needed so that the databases are mirror images.

All Palm OS[®] records contain three flag bits in both the handheld and desktop databases to facilitate synchronization:

- **Modify** — indicates the record was created or modified since the last synchronization
- **Delete** — indicates the record was deleted since the last synchronization
- **Archive** — indicates that the conduit should add the record to the database's archive on the desktop.

These flag bits plus other pieces of state information — such as whether a record exists on the other side — are used to perform mirror-image synchronization, as outlined in the following subsections.

Overview of Mirror-image Synchronization

For simplicity, assume that there is only one PC in the world, only one handheld, and only one database. Intuitively, mirror-image synchronization would proceed as follows:

- If a new record exists on one side but not the other, copy it to the other side. (New records are indicated by setting the Modified bit. There is no procedural distinction between a newly created record and a modified record, so this scheme is OK.)
- If a record is deleted on one side and is unchanged on the other side, delete it from both sides. Otherwise, if the record is deleted on one side but changed on the other, do not delete the changed record; instead, copy the changed record to the other side, overwriting the deleted record.
- If a record is modified only on one side, update the corresponding record on the other side.
- If a record is modified on both sides and the modification changes are identical, take no action.

Generic Conduit Framework

Mirror-Image Synchronization

- If a record is modified on both sides and the modification changes are different, then a few steps are needed. First, change the record ID of the desktop record ID to 0 and then write the record to the device. This adds the record as a new record. Once the device assigns the record a new ID, add this new record to the desktop database. Finally, write the original handheld record to the desktop, overwriting the original desktop record. In the end, there will be two records where there was only one before.

This is known as a **double modify** case. This procedure for handling it reflects one of the core concepts of conduit design: zero data loss. Because you cannot know what the user intended in this case, discard nothing so the user can decide later which record to delete.

For instance, if you scheduled a meeting for 2:00 P.M. today on your handheld and your secretary updated your desktop schedule with a conflicting meeting, you will discover, after HotSync Manager runs, that you have two records side by side in your datebook for 2:00 P.M. today.

The purpose of archiving is to allow for the deletion of a record from the normal databases while preserving the data. A request to archive a record should be ignored if the corresponding record on the other side has been changed, unless the change is identical. If the request should be ignored because the record on the other side has changed and the changes don't match, copy the changed record to this side.

Details of Mirror-image Synchronization

The foregoing intuitive presentation of mirror-image synchronization is spelled out in [Table 5.1](#), which shows the specific criteria and actions needed to perform a mirror-image synchronization for the combinations of the Modify, Delete, and Archive bits.

Table 5.1 Synchronizing Records

Handheld Record Status	Desktop Record Status	Action
Add M=1, D=0, A=0	No record nil	Add the handheld record to the desktop database.
Archive M=0, D=0, A=1	Delete M=0, D=1, A=0	Archive the handheld record and delete the record from both the handheld and desktop databases.
Archive M=0, D=0, A=1	No change M=0, D=0, A=nc	Archive the handheld record and delete the record from both the handheld and desktop databases.
Archive M=0, D=0, A=1	No record nil	Archive the handheld record and delete the record from the handheld.
Archive, change M=1, D=0, A=1	Change M=1, D=0, A=0	If the changes are identical, archive both the handheld record and the desktop record and delete the record from both the handheld and desktop databases. If the changes are not identical, do not archive the handheld record; instead handle the records as a double modify case.
Archive, no change M=0, D=0, A=1	Change M=1, D=0, A=0	Do not archive the handheld record; instead replace it with the desktop record.
Change M=1, D=0, A=0	Archive, change M=1, D=0, A=1	If the changes are identical, archive the handheld record and then delete the records from both the handheld and desktop databases. If the changes are not identical, do not archive the desktop record; instead handle the records as a double modify case
Change M=1, D=0, A=0	Archive, no change M=0, D=0, A=1	Do not archive the desktop record; instead, replace the record in the desktop database with the handheld record.

Generic Conduit Framework

Mirror-Image Synchronization

Table 5.1 Synchronizing Records (*continued*)

Handheld Record Status	Desktop Record Status	Action
Change M=1, D=0, A=0	Change M=1, D=0, A=0	If changes are identical, no action. If changes are not identical, handle the records as a double modify case.
Change M=1, D=0, A=0	Delete M=0, D=1, A=0	Do not delete the desktop record; instead, replace the desktop record with the handheld record.
Change M=1, D=0, A=0	No change M=0, D=0, A=0	Replace the desktop record with the handheld record.
Delete M=0, D=1, A=0	Change M=1, D=0, A=0	Do not delete the handheld record; instead, replace the handheld record with the desktop record.
Delete M=0, D=1, A=0	No change M=0, D=0, A=0	Delete the record from the desktop and handheld databases.
No change M=0, D=0, A=0	Archive M=0, D=0, A=1	Archive the desktop record, then delete the record from both databases.
No change M=0, D=0, A=0	Change M=1, D=0, A=0	Replace the handheld record with the desktop record
No change M=0, D=0, A=0	Delete M=0, D=1, A=0	Delete the record from the desktop and handheld databases.

Synchronizing to More than One Desktop

If we now expand the simplified scenario above for mirror-image synchronization to include the possibility of multiple handhelds and multiple PCs, additional state information and additional actions are required for correct mirror-image synchronization.

Synchronizing to Multiple Users

Palm OS allows multiple handhelds to synchronize to the same desktop. A unique user name must be assigned to each handheld. HotSync Manager on the desktop identifies the user of the handheld

to determine which set of desktop databases to use with this handheld.

A separate directory for each user name is maintained on the desktop, and each handheld application is given a subdirectory there, in which the associated conduit maintains that user's database. There are also subdirectories for archive and backup files.

Synchronizing to Multiple Desktops

Palm OS also allows handhelds to be synchronized to multiple desktops. Thus, you might have the Palm[™] Desktop software installed at home and at work, and want to carry information back and forth on the handheld.

In order to determine where a handheld was last synchronized, each desktop machine running the HotSync Manager application is given a unique identification number. As part of its standard operation, HotSync Manager always stores the ID of the desktop performing the HotSync session in a field reserved for this purpose in the handheld. When the handheld is next synchronized, HotSync Manager compares the desktop ID stored in the handheld with the current desktop ID. If they are the same, HotSync Manager knows that this handheld was last synchronized to this desktop. Otherwise, it was synchronized to another desktop machine.

If the handheld was synchronized to this machine, the Modify, Delete, and Archive bits accurately reflect the differences between the handheld and the desktop. The conduit only needs to iterate through the records which have those bits set on both sides in order to perform mirror-image synchronization. This is the so-called *fast sync* operation because only a subset of records must be visited.

If the handheld was previously synchronized to another machine, then its Modify, Delete, and Archive bits reflect what has changed since the last synchronization with the *other desktop*, and they do not match the last synchronization to this machine. Basically, the conduit must compare each and every record on the desktop and the handheld. This is the so-called *slow sync* operation because every record must be visited. Because this potentially includes bringing every record across a serial interface, the transaction time for a slow sync can be considerably longer than for a fast sync, depending upon such factors as the speed of the connection, the size of the

database, and the relative size of the records (a few big records generally require less handshaking than many little records, and can have a considerable speed advantage during a slow sync).

Overview of Synchronization Procedures

Fast Sync

As described in the previous section, fast sync can be performed when HotSync Manager observes that the desktop ID stored in the handheld indicates that this same desktop was last synchronized to this same handheld. In that case, record flag bits in each database correctly indicate the disposition of each record. The conduit only needs to iterate through the records on the handheld which have those bits set and compare them to corresponding records on the desktop.

The GenCn Framework provides the logic that implements the rules given in [Table 5.1](#). If your conduit interfaces to one of the standard Palm OS handheld applications, all you need to provide are methods to interface to your desktop database. If you have created a custom handheld application, you must additionally provide methods to interface to your handheld application's records. In either case, if HotSync Manager determines that a fast sync can be performed, then HotSync Manager together with the GenCn Framework will arrange to call your interface methods in order to perform the appropriate mirror-image synchronization.

Slow Sync

If a fast sync can not be performed, a slow sync must be performed. The aim of a slow sync is the same as a fast sync: assemble state information that will allow us to perform a set of operations resulting in mirror-image synchronization.

But here's the problem: if the handheld was previously synchronized to another machine, then the flag bits on the handheld database do not reflect what has changed since the last synchronization with this machine; instead, they reflect what has changed since synchronization with another machine.

For example, suppose you synchronize your handheld with your desktop at work. As a result, your handheld and work desktop

databases are mirror images. The work desktop stores a local backup copy of the synchronized database. On the way home, you create, delete, and modify some records in your handheld's Datebook application to set up your appointments for tomorrow. You get home and perform a sync to your home machine to pick up some important phone contacts off your home desktop. As part of that sync, all deleted records vanish from your handheld; records to be archived are shipped off to the local archive and are likewise deleted from the handheld; modified records and new records are reconciled with the home desktop, and their modified bits are cleared. Lastly, the home desktop's ID is copied to the handheld, indicating it was last synchronized to this machine.

When you get back to the office the next day, your boss mentions that he had to change the time of the sales meeting. You know that this change will conflict with a meeting you entered in your handheld last night. You get to your desk, pop the handheld in its cradle and hit the HotSync button. What happens? Your work desktop machine observes that your handheld has been visiting other desktops and ponders its options....

Clearly, if records are identical, there's no problem. But what if the records are different? They could have been modified or created on the handheld, modified or created on the desktop, or changed on both. Thus, simply comparing the current state of the desktop to the current state of the handheld is not effective. What information does your work desktop machine have to work with?

Well, recall that at the end of every sync, the conduit saves the synchronized state in a backup file. The backup file thus contains an historical record of both the state of the handheld and the state of the desktop databases at that instant they were last mirror-image synchronized:

- The difference between the backup file and the current state of the desktop database identifies what changed on the desktop since the last sync.
- The difference between the backup file and the current state of the handheld database identifies what changed on the handheld since the last sync with this desktop.

Generic Conduit Framework

Mirror-Image Synchronization

We can salvage the situation by a two-step operation as follows:

1. Compare every record in the backup file to the current state of the desktop databases.
2. Compare every record in the backup file to the current state of the handheld databases.

This recreates valid Modify/Delete flags for each side, which can then be used to perform a standard mirror-image synchronization.

For our particular example, here are the actions the conduit takes:

- The records that had their Delete bit set on the way home were removed by the sync to the home desktop. Upon iteratively comparing every record between the backup and the handheld, these deleted records are eventually identified. Because they were deleted from the handheld but still exist unmodified on the work desktop, they are copied to the handheld.
- The phone contact records that were added to the handheld by the home desktop do not exist on the work desktop. Upon iteratively comparing every record between the backup and the handheld, these new records are eventually identified. Because they exist on the handheld and not on the work desktop, they are copied to the work desktop.
- The records you changed on the way home do not have their Modify bits still set, because the sync with the home desktop cleared the Modify bits. Upon iteratively comparing every record between the backup and the handheld, these changed records are eventually discovered. Because they are changed on the handheld and not on the desktop, they are copied to the desktop.
- Lastly, the record your boss modified on your desktop had its Modify bit set in the database, but it's not set in the backup file. Upon iteratively comparing every record between the backup and the desktop databases, this modified record is eventually discovered. Because it is changed on the desktop and not on the handheld, it is copied to the handheld. When you view your schedule of meetings today, you discover that you have two records that appear in the same time slot.

Clearly, there are many variations on the above scenario that could be fruitfully considered. This is left as an exercise to the interested reader.

Creating Your Generic Conduit

The `GenCn` folder in the CDK's `Conduits` folder contains a sample Generic Conduit that you can adapt to meet your needs. Make a copy of the entire folder to create your new conduit project, then open the `Generic Conduit.mcp` project file.

You should open the project settings window and make adjustments specific to your conduit (changing the executable file's name, and so forth), then proceed to modify the base code to create your conduit.

This project has several targets; the two key targets are the “GenericConduit” and “GenericConduit(bundle)” targets; by selecting one of these two targets, you can select whether to build a non-bundled or bundled version of your conduit.

There are six groups in this project:

- **Source**—contains the conduit's source code. There are two files here; `GenCondNoBundle.cpp` is used by the “GenericConduit” target, while `GenCondBundle.cpp` is used by the “GenericConduit(bundle)” target.
- **Generic Conduit Classes**—contains the source files for the Generic Conduit Framework itself.
- **PFC**—contains the source files for the Palm Foundation Classes. These are utility classes to make conduit development easier.
- **Shared**—contains source files that are shared with the other example conduits.
- **Resources**—contains the various resource files used by the conduit, including separate resource files for each language the conduit supports.
- **Libraries**—contains the libraries the conduit uses, including HotSync Libraries, CarbonLib, and the MSL library.

Generic Conduit Framework

Creating Your Generic Conduit

As a general rule, you should only need to make changes to files in the Source and Resources groups. Instead of making changes to the Generic Conduit classes, you should create subclasses that implement the altered functionality you require.

The `GenericConduit.exp` and `GenericConduitBundle.exp` files specify which functions in the non-bundled and bundled versions of the Generic Conduit should be exported, and therefore public to the HotSync Manager application. You shouldn't need to make any changes to these files.

NOTE: The source for the sample conduits provided by PalmSource, Inc. includes functions `ConduitInit` and `ConduitExit`. These functions are an implementation detail; they are called when the conduit is loaded and unloaded by the operating system to perform low-level setup and teardown of the conduit library. You shouldn't need to change either of these functions.

The CPcMgr Class

The CPcMgr class provides methods to interact with a database stored on the desktop computer. By default, your data is saved in a binary format that the Generic Conduit is capable of reading and writing without modification. If you wish to store data in a custom format on the desktop computer, you must subclass CPcMgr and replace the `StoreDB` and `RestoreDB` methods to manipulate data in the desired format.

Adding Category Support

You'll need to store in your desktop file:

- category ID

You will also need to store, for each category:

- Category name
- Category ID (use 128-255 for new categories on the desktop)
- Whether the category has been modified on the desktop.

To enable category syncing, modify `OpenConduit` or `OpenConduitCarbon`:

```
new CMySynchronizer(rProps, 0);
```

to:

```
new CMySynchronizer(rProps,  
    GENERIC_FLAG_CATEGORY_SUPPORTED |  
    GENERIC_FLAG_APPINFO_SUPPORTED);
```

Add code to `StoreDB` to write out the categories:

```
    if (!m_pCatMgr)          // on a HH->PC copy,  
        // the PC category mgr isn't created  
        CPcMgr::ExtractCategories();  
if (m_pCatMgr) {  
    CCategory *cat;  
    for (cat = m_pCatMgr->FindFirst(); cat != NULL;  
        cat = m_pCatMgr->FindNext()){  
        WriteCategory(cat->GetID(), cat->GetName());  
    }  
}
```

Override `CMyPcMgr::ExtractCategories` to do nothing
(because you'll read your categories directly in `RetrieveDB`):

```
long CMyPcMgr::ExtractCategories(void)  
{  
    // when we read in the data in our  
    // RetrieveDB, we create the category  
    // manager there  
    return 0;  
}
```

Generic Conduit Framework

Creating Your Generic Conduit

Add code to RetrieveDB to read in the categories:

```
m_pCatMgr = new CPCategoryMgr;
// for each category in file {
    CPCategory cat;

    cat.SetID(catID);
    cat.SetName(catName);
    if (categoryChanged)
        cat.SetDirty();
    m_pCatMgr->Add(cat);
}
```

Using the Sync Manager API

The Sync Manager API is the programming interface to HotSync[®] technology operations. This interface provides the following capabilities:

- communication with the handheld
- access to databases on the handheld
- access to record level operations
- access to record categories

The Sync Manager API libraries are located in the HotSync Folder directory of the CDK. PalmSource, Inc. also provides debug versions of these libraries.

Communications Between Handheld and Desktop

The Sync Manager manages all communications between the handheld and the desktop computer during synchronization operations. This allows conduits to perform their operations without any consideration of how the data is moving between them.

The Sync Manager uses the HotSync transport to perform all communications. Your conduit does not need to distinguish between the various connection transports. In fact, your code doesn't need do anything with respect to the mechanics of exchanging data between the desktop computer and the handheld.

Using the Sync Manager

This section describes the flow of calls from a typical conduit to the Sync Manager. These calls falls into the following categories:

- registering the conduit
- opening a database
- reading and writing records
- working with record categories
- closing the database

Registering Your Conduit With Sync Manager

You must register your conduit with the Sync Manager by calling the `SyncRegisterConduit` function. The Sync Manager returns a handle for your conduit, which you use in subsequent calls. When your conduit is done, you unregister it by calling the `SyncUnRegisterConduit` function. Most conduits make these calls in their `OpenConduit` or `OpenConduitCarbon` functions.

Opening Your Database

You can use the `SyncCreateDB` function to create a new, opened database on the handheld, or you can use the `SyncOpenDB` function open an existing handheld database. [Listing 6.1](#) shows a section of code from the `OpenConduit` or `OpenConduitCarbon` function of the template conduit. This function calls both the `SyncCreateDB` and `SyncOpenDB` functions.

Listing 6.1 Opening a handheld database

```
err = ::SyncOpenDB(inSyncProperties.m_RemoteName[0], 0, dbHandle);
if (err == SYNCERR_FILE_NOT_FOUND && inSyncProperties.m_SyncType != eHHtoPC)
{
    err = ::SyncAddLogEntry ("Creating new Sample database on Palm Powered
handheld\n");
    CDbCreatedb dbInfo                = {};

    dbInfo.m_Creator                   = inSyncProperties.m_Creator;
    dbInfo.m_Flags                     = eRecord;
    dbInfo.m_CardNo                   = ( (BYTE)inSyncProperties.m_CardNo);
    dbInfo.m_Type                     = inSyncProperties.m_DbType;
```



```
strcat( dbInfo.m_Name, inSyncProperties.m_RemoteName[0] );  
  
err = ::SyncCreateDB(dbInfo);  
if( err == 0 )  
    dbHandle = dbInfo.m_FileHandle;  
}
```

Working with Databases

Each database is a collection of records with the following characteristics:

- Each record in a database has an ID that is unique within the database.
- Each record in a database has flags that specify whether it has been marked as private, deleted, modified, archived, or busy.
- Each record in a database belongs to a category.
- The database contains an application information block that stores global information about the database. This block is typically used to store the names of categories for records in the database.
- The database has a sort information block that stores ordering information for records in the database. This block is for use by the application; the Palm OS® does not use it.

Reading Records From a Handheld Database

To read records from a handheld database, you can call one of functions shown in [Table 6.1](#). Several of these functions iterate through a database: you call the function repeatedly and it automatically retrieves the next record for you until it reaches the last record meeting the specified criteria.

Using the Sync Manager API

Using the Sync Manager

Table 6.1 Functions for reading records from handheld databases

Function	Description
<code>SyncReadNextModifiedRec</code>	An iterator function that retrieves the next modified, archived, or deleted record from an opened record database on the handheld. Each call retrieves the next modified record until all modified records have been returned.
<code>SyncReadNextModifiedRecInCategory</code>	An iterator function that retrieves modified records in a category from an open database on the handheld. Each call retrieves the next modified record from the category, until all modified records have been retrieved.
<code>SyncReadNextRecInCategory</code>	An iterator function that retrieves any record in a category from an open database on the handheld, including deleted, archived, and modified records. Each call retrieves the next record from the category, until all records have been retrieved.
<code>SyncReadRecordById</code>	Retrieves a record, by ID, from an open database on the handheld.
<code>SyncReadRecordByIndex</code>	Retrieves a record, by index, from an open database on the handheld.
<code>SyncReadResRecordByIndex</code>	Retrieves a resource record, by index, from an open resource database on the handheld.

[Listing 6.2](#) shows a section of code from the template conduit that iterates through the records in the database that have been modified.

Listing 6.2 Reading modified records from a database

```
CRawRecordInfo rawRecordInfo = {};  
  
err = SYNCERR_NONE;  
while(err == SYNCERR_NONE)  
{  
    err = ::SyncReadNextModifiedRec(rawRecordInfo);  
    printf("SyncReadNextModifiedRec err == %d\n", err);  
    // **** work with this record  
}
```

Writing Records To a Handheld Database

You can use the `SyncWriteRec` to write a record to a handheld record database, and you can use the `SyncWriteResourceRec` to write a resource to a handheld resource database.

Cleaning Up Records

You can use the functions shown in [Table 6.2](#) at the end of your synchronization operations to remove all of the deleted records in your handheld database, and to clear the record status flags.

Table 6.2 Record clean-up functions

Function	Description
<code>SyncPurgeDeletedRecs</code>	Deletes all of the records that are marked as deleted or archived from an open record database on the handheld
<code>SyncResetSyncFlags</code>	Resets the modified flag of all records in the opened record database on the handheld. Resets the backup date for an opened record or resource database.

Closing Your Database

After you have finished with a handheld database, you must call the `SyncCloseDB` function or `SyncCloseDBEx` function to close it. The Sync Manager allows only one database to be open at any time.

IMPORTANT: If you open a database, you must close it before exiting your conduit; otherwise, other conduits will not be able to open their databases.

Interfacing with the HotSync Manager Application

This chapter describes how conduits interface with the HotSync[®] Manager application. There are two areas of interaction between a conduit and the HotSync Manager application:

- The conduit entry points are functions that your conduit provides for the HotSync Manager application to call during synchronization operations.
- Your conduit can call the HotSync log functions to add entries to the log that the user can view after synchronization operations are complete.

The remainder of this chapter describes how to interface with the HotSync Manager application in these ways.

The Conduit Entry Points

[Table 7.1](#) shows a summary of the entry point functions. The remainder of this section provides a brief description of each function. For more information about these functions, see [Chapter 1](#), “[Conduit API](#),” on page 1 in the *C/C++ Sync Suite Reference for Macintosh*.

Interfacing with the HotSync Manager Application

The Conduit Entry Points

NOTE: The source for the sample conduits provided by PalmSource, Inc. includes functions `ConduitInit` and `ConduitExit`. These functions are an implementation detail; they are called when the conduit is loaded and unloaded by the operating system to perform low-level setup and teardown of the conduit library. You shouldn't need to change either of these functions.

Table 7.1 Conduit entry points called by the HotSync Manager application

Function name	Description
<code>ConfigureConduit</code>	Presents a dialog that allows the user to configure the conduit. If your conduit does not provide a user interface, you can provide an empty version of this function.
<code>GetActionString</code>	Returns a string that the HotSync Manager application displays in the "Next HotSync action" window. <i>Note:</i> This function should be provided only for non-bundled conduits.
<code>CopyActionStringAsCFStringRef</code>	Returns a string, as a <code>CFStringRef</code> reference, that the HotSync Manager application displays in the "Next HotSync action" window. <i>Note:</i> This function should only be provided for bundled conduits.
<code>GetConduitName</code>	Returns the conduit's name for display purposes. <i>Note:</i> This function should be provided only for non-bundled conduits.
<code>CopyConduitNameAsCFStringRef</code>	Returns the conduit's name, as a <code>CFStringRef</code> reference, for display purposes. <i>Note:</i> This function should only be provided for bundled conduits.

Table 7.1 Conduit entry points called by the HotSync Manager application (*continued*)

Function name	Description
GetConduitVersion	Returns the conduit's version number. The supported range for conduit versions is 0x00000101 to 0x00000300
OpenConduit	The main conduit entry point. <i>Note:</i> This function should be provided only for non-bundled conduits.
OpenConduitCarbon	The main conduit entry point. <i>Note:</i> This function should only be provided for bundled conduits.

NOTE: The source for the sample conduits provided by PalmSource, Inc. includes functions `ConduitInit` and `ConduitExit`. These functions are an implementation detail; they are called when the conduit is loaded and unloaded by the operating system to perform low-level setup and teardown of the conduit library. You shouldn't need to change either of these functions.

ConfigureConduit

The HotSync Manager application calls the `ConfigureConduit` function when the user decides to customize your conduit with the following steps:

- chooses **Conduit Settings** from the HotSync Manager application menu
- selects your conduit and clicks **Conduit Settings** in the HotSync Manager application's Conduit Settings dialog box

When the HotSync Manager application calls this function, your conduit responds by displaying a modal configuration dialog box. This dialog box allows the user to specify what actions your conduit

Interfacing with the HotSync Manager Application

The Conduit Entry Points

is to perform. [Figure 7.1](#) shows the configuration dialog box displayed by the To Do conduit that is provided by PalmSource.

Figure 7.1 The settings dialog box for the To Do conduit



[Figure 7.1](#) shows a typical configuration dialog box for a mirror-image synchronization. Your conduit might allow additional configuration options.

GetActionString

The HotSync Manager application calls the `GetActionString` function to retrieve the string to display to the user in the “Next HotSync action” window. If your conduit does not use any settings, you can return a simple string such as “Normal” or “Synchronize.”

NOTE: `GetActionString` should only be implemented if your conduit isn’t bundled. If your conduit is bundled, implement `CopyActionStringAsCFStringRef` instead.

[Listing 7.1](#) shows the implementation of the `GetActionString` function from the template conduit. This function uses the `GetConduitCurrentSetting` function, which is one of many useful functions in the `UConduitUtils` class; the code for this class is found in the `Shared` folder inside of the `Conduits` source code folder of the CDK.

Listing 7.1 An example of the GetActionString function

```
long GetActionString(CSyncPreference& inSyncPrefs, char* ioActionString,
                    WORD inStrLen)
{
    short currentSetting;
    Str255 action;
    short saveRsrcRefNum = ::CurResFile();

    Try_
    {
        currentSetting = UConduitUtils::GetConduitCurrentSetting
            (gRsrcFileRefNum, inSyncPrefs.u.m_UserDirFSSpec);
    }
    Catch_(error)
    {
        action[0] = 0;           // Return empty string if anything goes wrong.
        return error;
    }

    ::UseResFile(gRsrcFileRefNum);

    if(currentSetting == 2)
        ::GetIndString(action, kActionStringResID, kDoNothingString);
    else
        ::GetIndString(action, kActionStringResID, kDoSomethingString);

    if(action[0] > inStrLen) action[0] = inStrLen;
    // copy action to output as a C string
    ::memcpy(ioActionString, &(action[1]), action[0]);
    ioActionString[action[0]] = 0;

    ::UseResFile(saveRsrcRefNum);

    return 0;
}
```

CopyActionStringAsCFStringRef

The HotSync Manager application calls the `CopyActionStringAsCFStringRef` function to retrieve the string to display to the user in the “Next HotSync action” window. If your conduit does not use any settings, you can return a simple string such as “Normal” or “Synchronize.”

Interfacing with the HotSync Manager Application

The Conduit Entry Points

NOTE: CopyActionStringAsCFStringRef should only be implemented if your conduit is bundled. If your conduit isn't bundled, implement GetActionString instead.

[Listing 7.2](#) shows the implementation of the CopyActionStringAsCFStringRef function from the Install conduit. This function uses the GetConduitCurrentSetting function, which is one of many useful functions in the UConduitUtils class; the code for this class is found in the Shared folder inside of the Conduits source code folder of the CDK.

Listing 7.2 An example of the CopyActionStringAsCFStringRef function

```
long CopyActionStringAsCFStringRef(CSyncPreference&
                                inSyncPrefs, CFStringRef* oActionString)
{
    #pragma unused( inSyncPrefs )
    longerr = noErr;
    SInt16currentSetting;

    require_action( oActionString != NULL,
        CopyActionStringAsCFStringRef_ParamErr, err = paramErr );
    *oActionString = NULL;

    try
    {
        currentSetting =
            UConduitUtils::GetConduitCurrentSetting(
                kInstallConduitCreator);
    }
    catch (LException err)
    {
        ::SysBeep(34);
        return err.GetErrorCodes();
    }

    if(currentSetting == eDoNothing)
        *oActionString = CFCopyLocalizedStringFromTableInBundle(
            CFSTR("Do Nothing"),
            CFSTR("Localizable"),
            gBundleRef,
```

```
        CFSTR("Conduit Action") );
else
    *oActionString = CFCopyLocalizedStringFromTableInBundle(
        CFSTR("Install Files"),
        CFSTR("Localizable"),
        gBundleRef,
        CFSTR("Conduit Action") );

CopyActionStringAsCFStringRef_ParamErr:

return err;
}
```

GetConduitName

The HotSync Manager application calls the `GetConduitName` function to retrieve the name of your conduit. Your implementation must fill in the name buffer with your conduit's name, and must return a status value that indicates whether the operation was successful.

NOTE: You should only implement the `GetConduitName` function if your conduit isn't bundled. Bundled conduits should implement the `CopyConduitNameAsCFStringRef` function instead.

For example, the template conduit's version of `GetConduitName` function, which is shown in [Listing 7.3](#) is typical. This function extracts the conduit's name from its 'CInf' resource and returns 0 to indicate success.

The template conduit version of `GetConduitName` uses a utility function named `ExtractUserVisibleConduitName`, which is also shown in [Listing 7.3](#), to retrieve the user name from resource.

Listing 7.3 An example of the `GetConduitName` function

```
long GetConduitName(char* ioConduitName, WORD inStrLen)
{
    Str255          conduitName = "\p";
    short           saveResFile = ::CurResFile();
    ::UseResFile(gRsrcFileRefNum);
```

Interfacing with the HotSync Manager Application

The Conduit Entry Points

```
Handle cinfHandle = ::Get1Resource('CInf', 0);
ExtractUserVisibleConduitName(cinfHandle,  conduitName);

::memcpy(ioConduitName, &conduitName[1], conduitName[0]);
ioConduitName[conduitName[0]] = 0;

::UseResFile(saveResFile);

return 0;
}

void ExtractUserVisibleConduitName(Handle inCinfResource,
                                   StringPtr outConduitName)
{
    SInt8 saveState = ::HGetState(inCinfResource);
    ::HLock(inCinfResource);

    StringPtr conduitName = (StringPtr)((*inCinfResource) +
                                        sizeof(short) + //conduitType
                                        sizeof(long) +  //conduit version
                                        sizeof(long));    //conduit creator
                                        //next byte is beginning of conduit name

    ::memcpy(outConduitName, conduitName, conduitName[0]+1);

    ::HSetState(inCinfResource, saveState);
}
```

CopyConduitNameAsCFStringRef

The HotSync Manager application calls the CopyConduitNameAsCFStringRef function to retrieve the name of your conduit. Your implementation must fill in the name buffer with your conduit's name, and must return a status value that indicates whether the operation was successful.

NOTE: You should only implement the CopyConduitNameAsCFStringRef function if your conduit is bundled. Non-bundled conduits should implement the GetConduitName function instead.

For example, the Install conduit's version of CopyConduitNameAsCFStringRef function, which is shown in [Listing 7.4](#) is typical. This function extracts the conduit's name from its 'CInf' resource and returns 0 to indicate success.

The Install conduit version of CopyConduitNameAsCFStringRef uses a utility function in the UConduitUtils class named ExtractUserVisibleConduitName, which is also shown in [Listing 7.4](#), to retrieve the user name from resource.

Listing 7.4 An example of the CopyConduitNameAsCFStringRef function

```
long CopyConduitNameAsCFStringRef( CFStringRef* oConduitName )
{
    long err = noErr;

    StCurResChainsaveRes( gNonLocalizedRsrcFileRefNum, gLocalizedRsrcFileRefNum
);

    require_action( oConduitName != NULL, CopyConduitNameAsCFStringRef_ParamErr,
err = paramErr );

    Handle cinfHandle = ::GetResource('CInf', 0);
    UConduitUtils::ExtractUserVisibleConduitName(cinfHandle, oConduitName);

    CopyConduitNameAsCFStringRef_ParamErr:

    return err;
}

void UConduitUtils::ExtractUserVisibleConduitName(Handle inCinfResource,
StringPtr outConduitName)
{
    SInt8 saveState = ::HGetState(inCinfResource);
    ::HLock(inCinfResource);

    StringPtr conduitName = (StringPtr)((*inCinfResource) +
sizeof(short) + //conduitType
sizeof(long) + //conduit version
sizeof(long)); //conduit creator
```

Interfacing with the HotSync Manager Application

The Conduit Entry Points

```
        //next byte is beginning of conduit name

::memcpy(outConduitName, conduitName, conduitName[0]+1);

::HSetState(inCinfResource, saveState);
}
```

GetConduitVersion

The HotSync Manager application calls the `GetConduitVersion` function to retrieve the version of your conduit that is running on the desktop computer. Your implementation must pack your major version number into the high byte of the low word in the result, and must pack your minor version number into the low byte of the low word in the result.

For example, the template conduit's version of the `GetConduitVersion` function, which is shown in [Listing 7.5](#) is typical. This function simply returns a constant value that specifies the version number.

Listing 7.5 An example of the `GetConduitVersion` function

```
#define TEMPLATE_CONDUIT_VERSION 0x00000300
// Must be 0x00000300 or higher to work with Palm Desktop 4.0
or later

DWORD GetConduitVersion()
{
    return TEMPLATE_CONDUIT_VERSION;
}
```

OpenConduit

The HotSync Manager application calls the `OpenConduit` function when your conduit is loaded to begin the process of synchronizing data between the desktop computer and the handheld. Your implementation of this function needs to perform operations such as the following:

- register your conduit with the Sync Manager

- perform synchronization operations, either by instantiating a CSynchronizer object, or by calling Sync Manager API functions
- unregister your conduit

NOTE: You should only implement the `OpenConduit` function if your conduit is non-bundled. Bundled conduits should implement the `OpenConduitCarbon` function instead.

[Listing 7.6](#) shows the implementation of the `OpenConduit` function from the template conduit.

Listing 7.6 An example of the `OpenConduit` function

```
long OpenConduit(PROGRESSFN inProgressCallBack,
                  CSyncProperties& inSyncProperties)
{
    long err = 0;

    CONDHANDLE conduitHandle = 0;
    Byte dbHandle = 0;
    WORD recCount = 0;
    CRawRecordInfo rawRecordInfo = {};

    gProgressCallBack = inProgressCallBack;

    err = ::SyncRegisterConduit(conduitHandle);

    err = ::SyncOpenDB(inSyncProperties.m_RemoteName[0], 0, dbHandle);

    if (err == SYNCERR_FILE_NOT_FOUND && inSyncProperties.m_SyncType != eHHtoPC)
    {
        err = ::SyncAddLogEntry("Creating new Sample database on handheld\n");
        CDbCreatedb dbInfo = {};
        dbInfo.m_Creator = inSyncProperties.m_Creator;
        dbInfo.m_Flags = eRecord;
        dbInfo.m_CardNo = ( (BYTE)inSyncProperties.m_CardNo);
        dbInfo.m_Type = inSyncProperties.m_DbType;
        strcat( dbInfo.m_Name, inSyncProperties.m_RemoteName[0]);

        err = ::SyncCreateDB(dbInfo);
        if( err == 0 )
            dbHandle = dbInfo.m_FileHandle;
    }
}
```

Interfacing with the HotSync Manager Application

The Conduit Entry Points

```
err = ::SyncGetDBRecordCount( dbHandle, recCount );
err = SYNCERR_NONE;
while(err == SYNCERR_NONE)
{
    err = ::SyncReadNextModifiedRec(rawRecordInfo);
    // -- perform the work of your conduit here --
}

err = ::SyncCloseDB(dbHandle);
err = ::SyncResetSyncFlags(dbHandle);
err = ::SyncUnRegisterConduit(conduitHandle);

return err;
}
```

OpenConduitCarbon

The HotSync Manager application calls the `OpenConduitCarbon` function of a bundled conduit to begin the process of synchronizing data between the desktop computer and the handheld. Your implementation of this function needs to perform operations such as the following:

- register your conduit with the Sync Manager
- perform synchronization operations, either by instantiating a `CSynchronizer` object, or by calling Sync Manager API functions
- unregister your conduit

NOTE: You should only implement the `OpenConduitCarbon` function if your conduit is bundled. Non-bundled conduits should implement the `OpenConduit` function instead.

[Listing 7.7](#) shows the implementation of the `OpenConduitCarbon` function from the template conduit.

Listing 7.7 An example of the `OpenConduitCarbon` function

```
long OpenConduitCarbon(PROGRESSFN CARBON inProgressCallBack,
                      CSyncProperties& inSyncProperties)
{
    long retval = 0;

    try
    {
        if (inProgressCallBack)
        {
            CInstallConduit installConduit(gBundleRef, gNonLocalizedRsrcFileRefNum,
                                           gLocalizedRsrcFileRefNum, inProgressCallBack,
                                           inSyncProperties);

            retval = installConduit.EngageInstall();
        }
    }
    catch (LException err)
    {
        retval = err.GetErrorCode();
    }
    catch (...)
    {
        retval = SYNCERR_UNKNOWN;
    }

    return retval;
}
```

Logging Entries in the HotSync Log

During synchronization operations, the HotSync Manager application and any conduits that it runs can add messages to the HotSync log. If any unusual messages are added to the log during the synchronization, the HotSync Manager application notifies the user, who can then review the log.

Conduits can also use add messages to the HotSync log for debugging purposes, as described in [Chapter 10, “Debugging Conduits.”](#)

Interfacing with the HotSync Manager Application

Logging Entries in the HotSync Log

The Sync Manager API provides the LogAddEntry function, the declaration of which is shown here, for logging messages.

```
long LogAddEntry(LPCTSTR pszEntry,
                 Activity act,
                 BOOL bTimeStamp);
```

This function can optionally time stamp each message and associate it with an activity type. Here is a typical call to the AddLogEntry function:

```
err = ::AddLogEntry ("Creating new database on handheld.\n");
```

For more information about the logging functions, see [Chapter 3, "HotSync Log API,"](#) on page 155 of *C/C++ Sync Suite Reference for Macintosh*.

Using Expansion Technology

This chapter describes how to work with handheld expansion cards and add-on devices from the desktop using the Expansion and Virtual File System (VFS) Managers. The sections in this chapter are:

- [Expansion Support](#)
- [Architectural Overview](#)
- [Standard Directories](#)
- [Card Insertion and Removal](#)
- [Checking for Expansion Cards](#)
- [Volume Operations](#)
- [File Operations](#)
- [Directory Operations](#)
- [Custom Calls](#)

Expansion Support

Beginning with Palm OS® 4.0, a set of optional system extensions provide a standard mechanism by which Palm OS applications, desktop applications, and conduits can take advantage of the expansion capabilities of various Palm Powered™ handhelds. This capability not only augments the memory and I/O of the handheld, but facilitates data interchange with other Palm Powered handhelds and with devices that are not running Palm OS. These other devices include digital cameras, digital audio players, desktop or laptop computers, and the like.

This section covers the following topics:

- [Primary vs. Secondary Storage](#)
- [Expansion Slot](#)
- [Universal Connector](#)

Primary vs. Secondary Storage

All Palm Powered handhelds contain **primary storage** — directly addressable memory that is used for both long-term and temporary storage. This includes storage RAM, used to hold nonvolatile user data and applications; and dynamic RAM, which is used as working space for temporary allocations.

On most handhelds, primary storage is contained entirely within the handheld itself. The Palm OS memory architecture does not limit handhelds to this, however; handhelds can be designed to accept additional storage RAM. Some products developed by Handspring work this way; memory modules plugged into the Springboard slot are fully-addressable and appear to a Palm OS application as additional storage RAM.

To access primary storage RAM from the desktop, conduits make Sync Manager API calls during a HotSync[®] operation (see [Chapter 6, “Using the Sync Manager API,”](#) on page 57).

Secondary storage, by contrast, is designed primarily to be add-on nonvolatile storage. Although not limited to any particular implementation, most secondary storage media:

- can be inserted and removed from the expansion slot at will
- are based upon a third-party standard, such as Secure Digital (SD), MultiMediaCard (MMC), CompactFlash, Sony’s Memory Stick, and others
- present a serial interface, accessing data one bit, byte, or block at a time

Conduits access primary storage through the Sync Manager during a HotSync operation. To access secondary storage, however, conduits use the Expansion and VFS Managers. These have been designed to support as broad a range of serial expansion architectures as possible.

Desktop applications or installers have, in the past, used the Install Aide API to queue applications and databases for the Install conduit to install in primary storage on the handheld during the next HotSync operation. With Palm OS 4.0 and HotSync Manager 3.0, the new User Manager API lets applications queue files for the Install conduit to copy from the desktop to secondary storage on expansion cards. The User Manager API also enables desktop applications and installers to access information about handheld users on the desktop as well as expansion slots on users' handhelds.

For more information about the User Manager API, see [Chapter 6, "User Manager API,"](#) on page 233 in the *C/C++ Sync Suite Reference for Macintosh*.

Expansion Slot

The expansion slots found on many Palm Powered handhelds vary depending on the manufacturer. While some may accept SD and MMC cards, others may accept Memory Stick or CompactFlash. Note that there is no restriction on the number of expansion slots that handhelds can have.

Depending on the expansion technology used, there can be a wide variety of expansion cards usable with a given handheld:

- Storage cards provide secondary storage and can either be used to hold additional applications and data, or can be used for a specific purpose, for instance as a backup mechanism.
- ROM cards hold dedicated applications and data.
- I/O cards extend the handheld's I/O capabilities. A modem, for instance, could provide wired access, while a Bluetooth transceiver could add wireless capability.
- "Combo" cards provide both additional storage or ROM along with some I/O capability.

Universal Connector

Certain newer Palm Powered handhelds may be equipped with a universal connector that connects the handheld to a HotSync cradle. This connector can be used to connect the handheld to snap-on I/O devices as well. A special slot driver dedicated to this connector allows handheld-to-accessory communication using the serial

Using Expansion Technology

Architectural Overview

portion of the connector. This “plug and play” slot driver presents the peripheral as a card in a slot, even to the extent of providing the card insertion notification when the peripheral is attached.

Because the universal connector’s slot driver makes a snap-on peripheral appear to be a card in a slot, such peripherals can be treated as expansion cards by the handheld application. From a conduit’s perspective, such a peripheral is not accessible during a *cradle-based* HotSync operation, because the universal connector is physically unavailable to a peripheral if it is used by the HotSync cradle. However, if the HotSync operation is via a *wireless* connection — for example, infrared — then a conduit can access such a peripheral. See “[Custom Calls](#)” on page 107 for a discussion of how conduits can make custom calls to access custom file systems or I/O devices.

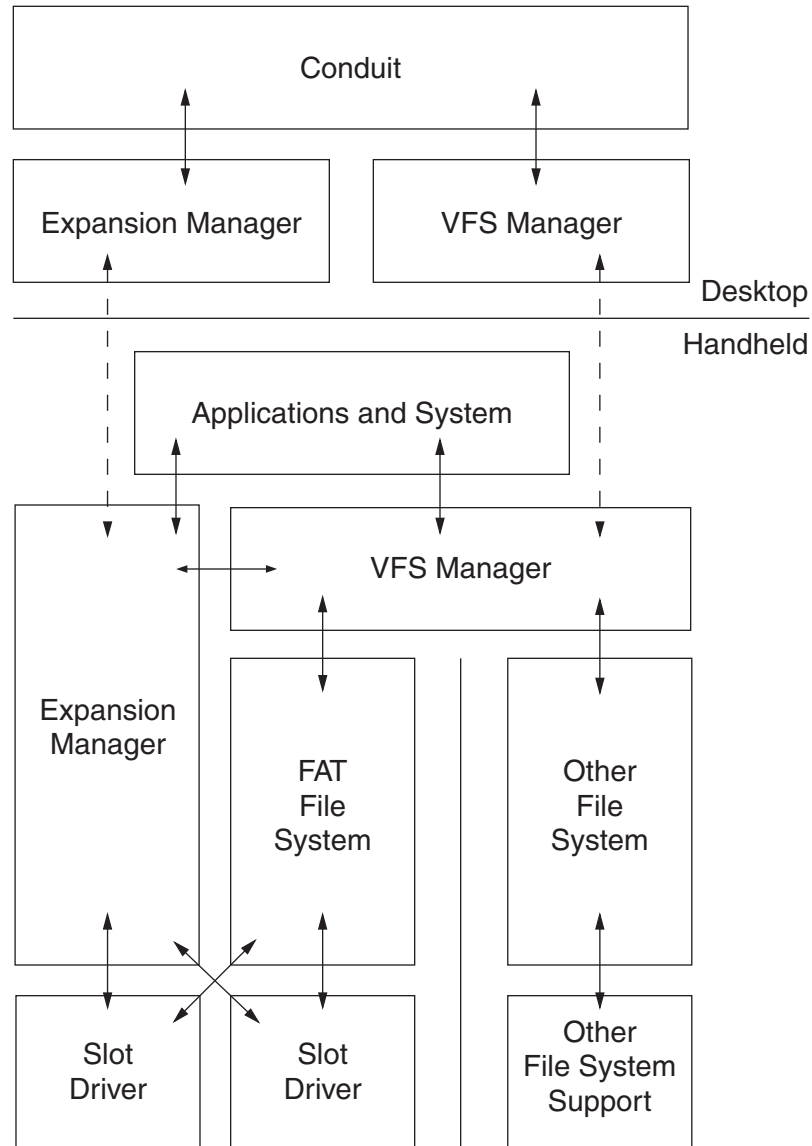
For the remainder of this chapter, wherever an I/O card could be used, the phrase “expansion card” can be taken to mean both “expansion card” and “plug and play peripheral,” but only if the the HotSync operation is not via the universal connector.

Architectural Overview

[Figure 8.1](#) illustrates the Palm OS expansion architecture. It is designed to be flexible enough to support multiple file systems and diverse physical expansion mechanisms while still presenting a consistent set of APIs to applications and to other parts of Palm OS. The following sections describe the major components of the Palm OS expansion architecture. Working from the bottom up, those components are:

- [Slot Drivers](#)
- [File Systems](#)
- [VFS Manager](#)
- [Expansion Manager](#)

Figure 8.1 Palm OS expansion architecture



Slot Drivers

A slot driver is a standard Palm OS shared library on the handheld. It is a special library that encapsulates direct access to the hardware and provides a standard set of services to the Expansion Manager and, optionally, to file system libraries. As illustrated in [Figure 8.1](#), neither handheld applications nor conduits normally interact directly with slot drivers.

Each expansion slot has a slot driver associated with it. Slots are identified by a unique **slot reference number**, which is assigned by the Expansion Manager on the handheld. Expansion cards themselves are not numbered individually; applications typically reference the slot into which a card is inserted. Note, however, that a slot may or may not have a card in it at any given time, and that a card can be inserted and removed while an application is running.

NOTE: “Card number” is a Sync Manager term and is not to be confused with “slot reference number.”

The Expansion Manager on the handheld identifies slots by slot reference numbers. These slot reference numbers may change depending on the order in which slot drivers are loaded by the Expansion Manager. Moreover, slot reference numbers are available only to conduits during a HotSync operation via the Expansion Manager API. Therefore HotSync Manager assigns **slot IDs** to slots on the handheld at the beginning of each HotSync operation and saves them for the corresponding user in the user data store on the desktop.

The User Manager API, which is not used during a HotSync operation, uses slot IDs to identify slots instead of slot reference numbers. This API simply returns the information saved on the desktop during the last HotSync operation, so this information might not be accurate for the next HotSync operation because the user might have changed or updated the handheld between HotSync operations.

File Systems

The Palm OS expansion architecture defines a common interface for all file system implementations on Palm OS. This interface consists of a complete set of APIs for interacting with the file system, including the ability to open, close, read, write, and delete both files and directories on named volumes. Almost all of these handheld APIs are available on the desktop so that conduits can interact with file systems almost as completely as handheld applications can.

File system implementations are packaged as shared libraries on the handheld. They are modular plug-ins that add support for a particular type of file system, such as VFAT, HFS, or NFS. The Palm OS expansion architecture allows multiple **file system libraries** to be installed at any given time. Typically, an implementation of the VFAT file system is present.

VFAT is the industry standard for flash memory cards of all types. It enables easy transfer of data and or applications to desktops and other devices. The VFAT file system library included with Palm OS software versions 4.0 and later natively supports VFAT file systems on secondary storage media. It is able to recognize and mount FAT and VFAT file systems, and offers to reformat unrecognizable or corrupted media.

Because the VFAT file system requires long filenames to be stored in Unicode/UCS2 format, the standard VFAT file system library supports conversion between UCS2 and Shift-JIS (the standard Palm OS multi-byte character encoding), and between UCS2 and the Palm OS/Latin encoding.

VFS Manager

The Virtual File System (VFS) Manager provides a unified API that gives handheld applications and desktop conduits access to many different file systems on many different media types. It abstracts the underlying file systems so that applications and conduits can be written without regard to the actual file system in use. The VFS Manager provides conduits many API functions for manipulating files, directories, and volumes *during* a HotSync operation.

The VFS Manager and the Sync Manager APIs

With the addition of the VFS Manager to the C/C++ Sync Suite, there are now two distinct ways that conduits can store and retrieve Palm OS user data:

- The Sync Manager accesses resource and record databases in the handheld's primary storage RAM. It effectively calls the handheld's Data Manager, which was specifically designed to make the most of the limited dynamic RAM and the nonvolatile RAM used instead of disk storage on most handhelds. Use the Sync Manager to store and retrieve Palm OS user data when storage on the handheld is all that is needed, or when efficient access to data is paramount.
- The VFS and Expansion Managers were designed specifically to support many types of expansion memory as secondary storage. The VFS Manager API presents a consistent interface to many different types of file systems on many types of external media. Conduits that use the VFS Manager API can support the widest variety of file systems. Use the VFS Manager when your conduit needs to read and write data stored on external media.

Conduits should use the appropriate APIs for each given situation. The Sync Manager, being an efficient manager of storage in the storage heap, should be used whenever access to external media is not absolutely needed. Use the VFS Manager API when interoperability and file system access is needed. Note, however, that the VFS Manager adds the extra overhead of buffering all reads and writes in the handheld's memory when accessing data, so only conduits that specifically need this functionality should use the VFS Manager.

For more information on the Sync Manager see [Chapter 6, "Using the Sync Manager API,"](#) on page 57. For details of the API presented by the VFS Manager, see [Chapter 5, "Virtual File System Manager API,"](#) on page 173 in the *C/C++ Sync Suite Reference for Macintosh*.

Expansion Manager

The Expansion Manager is a software layer that manages slot drivers on Palm Powered handhelds. Supported expansion card types include, but are not limited to, Memory Stick and SD cards.

The Expansion Manager does not support these expansion cards directly; rather, it provides an architecture and higher level set of API functions that, with the help of low-level slot drivers and file system libraries, support these types of media.

The Expansion Manager on the handheld:

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

The Expansion Manager API for conduits provides an interface to the Expansion Manager on the handheld *during* a HotSync operation. Through this interface, conduits can determine whether an expansion card is present in a slot and get information about those cards.

For details of the API presented by the Expansion Manager, see [Chapter 4, “Expansion Manager API,”](#) on page 161 in the *C/C++ Sync Suite Reference for Macintosh*.

Standard Directories

The user experience presented by Palm OS is simpler and more intuitive than that of a typical desktop computer. Part of this simplicity arises from the fact that Palm OS does not present a file system to the user. Users do not have to understand the complexities of a typical file system; applications are readily available with one or two taps of a button or icon, and data associated with those applications is accessible only through each application. Maintaining this simplicity of user operation while supporting a file system on an expansion card is made possible through a standard set of directories on the expansion card.

[Table 8.1](#) lists the standard directory layout for all “standards compliant” Palm OS secondary storage. All Palm OS relevant data should be in the /PALM directory (or in a subdirectory of the /PALM directory), effectively partitioning off a private name space.

Using Expansion Technology

Card Insertion and Removal

Table 8.1 Standard directories

Directory	Description
/	Root of the secondary storage.
/PALM	Most data written by Palm OS applications lives in a subdirectory of this directory. <code>start.prc</code> lives directly in /PALM. This optional file is automatically run when the secondary storage volume is mounted. Other applications may also reside in this directory.
/PALM/Backup	Reserved by Palm OS for backup purposes.
/PALM/Programs	Catch-all for other applications and data.
/PALM/Launcher	Home of Launcher-visible applications.

In addition to these standard directories, the VFS Manager supports the concept of a **default directory**; a directory in which data of a particular type is typically stored. See “[Determining the Default Directory for a Particular File Type](#)” on page 103 for more information.

Card Insertion and Removal

The Expansion Manager supports the insertion and removal of expansion media at any time. The handheld continues to run as before, though an application switch may occur upon card insertion. The handheld need not be reset or otherwise explicitly informed that a card has been inserted or removed.

WARNING! Because of the way certain expansion cards are constructed, if the user removes an expansion card while an application or conduit is *writing* to it, in certain rare circumstances it is possible for the card to become damaged to the point where either it can no longer be used or it must be reformatted. To the greatest extent possible, applications and conduits should write to the card only at well-defined points. The card can be removed without fear of damage while an application or conduit is *reading* from it, however.

If the user removes a card while your conduit is accessing it during a HotSync operation, the Expansion Manager or VFS Manager function fails. PalmSource, Inc. recommends that your conduit always check errors returned by all API functions and add helpful messages to the user in the HotSync log.

Checking for Expansion Cards

Before looking for an expansion card, your conduit should first make sure that the handheld supports expansion by verifying the presence of the Expansion and VFS Managers. It can then query for mounted volumes. Finally, your conduit may want to ascertain the capabilities of the card; whether it has memory, whether it does I/O, and so on. The following sections describe each of these steps:

- [Verifying Handheld Compatibility](#)
- [Checking for Mounted Volumes](#)
- [Enumerating Slots](#)
- [Determining a Card's Capabilities](#)

Verifying Handheld Compatibility

There are many different types of Palm Powered handhelds, and in the future there will be many more. Some will have expansion slots to support secondary storage, and some will not. Hardware to support secondary storage is optional, and may or may not be present on a given type of handheld. Because the Expansion and

Using Expansion Technology

Checking for Expansion Cards

VFS Managers are of no use on a handheld that has no physical expansion capability, they are optional system extensions that are not present on every type of Palm Powered handheld.

Because of the great variability both in handheld configuration and in the modules that can be plugged into or snapped onto the handheld, conduits should not attempt to detect the manufacturer or model of a specific handheld when determining whether it supports secondary storage. Instead, check for the presence and capabilities of the underlying operating system.

The VFS Manager and the Expansion Manager on the handheld are individual system extensions that are both optional. They both make use of other parts of the operating system that were introduced in Palm OS software version 4.0.

NOTE: Although your conduit could check for the presence of both the VFS and Expansion Managers, it can take advantage of the fact that the VFS Manager relies on the Expansion Manager and is not present without it. Therefore, if the VFS Manager is present, you can safely assume that the Expansion Manager is present as well.

Therefore your conduit must check for the presence of the VFS Manager on the handheld before calling any VFS or Expansion Manager API functions. This section presents two methods of verifying the presence of the VFS Manager on the handheld:

- [Using the VFSSupport Function](#)
- [Using the SyncReadFeature Function](#)

Using the VFSSupport Function

The desktop VFS Manager API provides the [VFSSupport](#) function to verify whether the handheld has an expansion slot and whether any file systems are present.

When you call `VFSSupport`, it passes back the version of the Expansion Manager on the handheld — which is identical to the VFS Manager version number — or zero, if no expansion slot is present. In a second parameter, it passes back the number of

volumes present on the card or zero, if there is no file system present on the card or no card in the slot.

NOTE: The `VFSSupport` function is the primary method for conduits to determine whether a handheld has expansion slots. This information has already been obtained by the desktop VFS Manager, so no additional calls are made to the handheld at the time your conduit calls `VFSSupport` — which makes it more efficient than using `SyncReadFeature`.

Using the `SyncReadFeature` Function

The Sync Manager API provides [SyncReadFeature](#) to verify the presence of defined feature sets on the handheld. To check for the VFS Manager's system feature, do the following:

- call [SyncReadFeature](#)
- supply `sysFileCVFSMgr` for the feature creator
- supply `vfsFtrIDVersion` for the feature number

[Listing 8.1](#) shows how to use `SyncReadFeature` to check for the presence and proper version of the VFS Manager. Note that `expectedVFSMgrVersionNum` should be replaced by the actual version number you expect.

Listing 8.1 `SyncReadFeature` to verify the presence of the VFS Manager

```
UINT32 vfsMgrVersion;
long err;

err = SyncReadFeature(sysFileCVFSMgr, vfsFtrIDVersion,
                     &vfsMgrVersion);
if(err){
    // The VFS Manager is not installed.
}
else {
    // Check the version number of the VFS Manager,
    // if necessary.
    if(vfsMgrVersion == expectedVFSMgrVersionNum)
        // Everything is OK.
}
```

Using Expansion Technology

Checking for Expansion Cards

Even though presence of the VFS Manager implies presence of the Expansion Manager, you can also check for the Expansion Manager's system feature as follows:

- call [SyncReadFeature](#)
- supply sysFileCExpansionMgr for the feature creator
- supply expFtrIDVersion for the feature number

[Listing 8.2](#) shows how to use SyncReadFeature to check for the presence and proper version of the Expansion Manager. Note that expectedExpMgrVersionNum should be replaced by the actual version number you expect.

Listing 8.2 SyncReadFeature to verify the presence of the Expansion Manager

```
UINT32 expMgrVersion;
long err;

err = SyncReadFeature(sysFileCExpansionMgr, expFtrIDVersion,
                     &expMgrVersion);
if(err){
    // The Expansion Manager is not installed.
}
else {
    // Check version number of the Expansion Manager,
    // if necessary.
    if(expMgrVersion == expectedExpMgrVersionNum)
        // Everything is OK.
}
```

Checking for Mounted Volumes

Conduits rely on the fact that Palm OS automatically mounts any recognized volumes inserted into or snapped onto the handheld. Therefore conduits can simply enumerate the mounted volumes and select one as appropriate. [Listing 8.3](#) illustrates how to do this.

Listing 8.3 Enumerating mounted volumes

```
WORD numVolumes = 0;
WORD *pwVolRefNumList;

// The first call returns only the number of mounted volumes,
// not their reference numbers.
VFSVolumeEnumerate (&numVolumes, NULL);
if (numVolumes)
{
    // Allocate buffer for volume reference list.
    pwVolRefList = new WORD [numVolumes];
    if (pwVolRefList != NULL)
    {
        // Get the volume reference numbers.
        VFSVolumeEnumerate (&numVolumes, pwVolRefList);
    }
}
```

The volume reference numbers obtained from [VFSVolumeEnumerate](#) can then be used with many of the volume, directory, and file operations that are described later in this chapter.

Occasionally a conduit needs to know more than that there is secondary storage available for use. Those conduits likely need to take a few extra steps, beginning with checking each of the handheld's slots, as described in the next section.

Enumerating Slots

Before you can determine which expansion modules are attached to a handheld, you must first determine how those modules could be attached. Expansion cards and some I/O devices could be plugged into physical slots, and snap-on modules could be connected through the handheld's universal connector. Irrespective of how they are physically connected, the Expansion Manager presents these to the developer as slots. The [ExpSlotEnumerate](#) function makes it simple to enumerate these slots. [Listing 8.4](#) illustrates the use of this function.

Using Expansion Technology

Checking for Expansion Cards

Listing 8.4 Enumerate a handheld's expansion slots

```
WORD wSlotRefList[32]; // Buffer for slot reference numbers.
WORD wSlotRefCount;    // Number of entries allocated for list.
long retval;

// Allocate enough space for buffer.
wSlotRefCount = sizeof (wSlotRefList) / sizeof (wSlotRefList[0]);
retval = ExpSlotEnumerate(&wSlotRefCount, wSlotRefList, NULL);
```

The array of slot reference numbers passed back by `ExpSlotEnumerate` uniquely identify all slots. A slot reference number can be supplied to various Expansion Manager functions to obtain information about the slot, such as whether there is a card or other expansion module present in the slot.

Checking a Slot for the Presence of a Card

Use the [ExpCardPresent](#) function to determine whether a card is present in a given slot. Given the slot reference number, this function returns `SYNCERR_NONE` if there is a card in the slot, or an error if either there is no card in the slot or there is a problem with the specified slot.

Determining a Card's Capabilities

Just knowing that an expansion card is inserted into a slot or connected to the handheld is not enough; your conduit needs to know something about the card to ensure that the operations it needs to perform are compatible with the card. For instance, if your conduit needs to write data to the card, its important to know whether writing is permitted.

The capabilities available to your conduit depend not only on the card but on the slot driver as well. Handheld manufacturers will provide one or more slot drivers that define standard interfaces to certain classes of expansion hardware. Card and device manufacturers may also choose to provide card-specific slot drivers, or they may require that applications use the slot custom control function and a registered creator code to access and control certain cards.

The slot driver is responsible for querying expansion cards for a standard set of capabilities. When a slot driver is present for a given expansion card, you can use the [ExpCardInfo](#) function to determine the following:

- the name of the expansion card's manufacturer
- the name of the expansion card
- the "device class," or type of expansion card. Values returned here might include "Ethernet" or "Backup"
- a unique identifier for the device, such as a serial number
- whether the card supports both reading and writing, or whether it is read-only

Note that the existence of the `ExpCardInfo` function does not imply that all expansion cards support these capabilities. It means only that the slot driver is able to assess a card and report its findings up to the Expansion Manager.

Volume Operations

If an expansion card supports a file system, the VFS Manager allows you to perform a number of standard volume operations. To determine which volumes are currently mounted and available, use [VFSVolumeEnumerate](#). This function, the use of which is illustrated in "[Checking for Mounted Volumes](#)" on page 90, returns a list of volume reference numbers, which you select from and then supply to the remainder of the volume operations.

NOTE: Volume reference numbers can change each time the handheld mounts a given volume. To keep track of a particular volume, save the volume's label rather than its reference number.

When the user inserts a card containing a mountable volume into a slot, the VFS Manager attempts to mount the volume automatically. Conduits cannot otherwise mount a volume.

Use [VFSVolumeFormat](#) to format a volume. Once the card has been formatted, the VFS Manager automatically mounts it and `VFSVolumeFormat` passes back a new volume reference number.

Using Expansion Technology

Volume Operations

The [VFSVolumeGetLabel](#) and [VFSVolumeSetLabel](#) functions get and set the volume label, respectively. Because the file system is responsible for verifying the validity of strings, you can try to set the volume label to any desired value. If the file system does not natively support the name given, the VFS Manager creates the /VOLUME.NAM file used to support long volume names (see “[Naming Volumes](#)” on page 96 for more information) or you get an error back if the file system does not support the supplied string.

NOTE: Most conduits should not need to call the [VFSVolumeSetLabel](#) function. This function may create or delete a file in the root directory, which would invalidate any current calls to [VFSDirEntryEnumerate](#).

Additional information about the volume can be obtained through the use of [VFSVolumeSize](#) and [VFSVolumeInfo](#). As the name implies, [VFSVolumeSize](#) returns size information about the volume. In particular, it returns both the total amount of space on the volume, in bytes, and the amount of that volume’s space that is currently in use, again in bytes. [VFSVolumeInfo](#) returns various pieces of information about the volume, including:

- whether the volume is hidden
- whether the volume is read-only
- whether the volume is supported by a slot driver, or is being simulated by Palm OS Emulator
- the type and creator of the underlying file system
- the slot with which the volume is associated, and the reference number of the slot driver controlling the slot
- the type of media on which this volume is located, such as SD, CompactFlash, or Memory Stick

All of the above information is returned encapsulated within a [VolumeInfoType](#) structure. Whether the volume is hidden or read-only is further encoded into a single field within this structure; see “[Volume Attributes](#)” on page 177 in the *C/C++ Sync Suite Reference for Macintosh* for the bits that make up this field.

The rest of this section covers the following topics:

- [Hidden Volumes](#)
- [Matching Volumes to Slots](#)
- [Naming Volumes](#)

Hidden Volumes

Included among the volume attributes is a “hidden” bit, `vfsVolumeAttrHidden`, that indicates whether the volume on the card is to be visible or hidden. Hidden volumes are typically not meant to be directly available to the user; the Launcher and the CardInfo application both ignore all hidden volumes.

To make a volume hidden, simply create an empty file named `HIDDEN.VOL` in the `/PALM` directory. The [VFSVolumeInfo](#) function looks for this file and, if found, returns the `vfsVolumeAttrHidden` bit along with the volume’s other attributes.

NOTE: The `vfsVolumeAttrHidden` attribute is not defined in the desktop VFS Manager header file but can be passed back in a [VolumeInfoType](#) structure. It is defined in the handheld’s VFS Manager header file available in the Palm OS SDK, so you can copy it from that file to the desktop `VFSMgr.h` file.

Matching Volumes to Slots

In most cases, a conduit does not need to know the specifics of an expansion card as provided by the [ExpCardInfo](#) function. Often, the information provided by the [VFSVolumeInfo](#) function is enough. Some applications need to know more about a particular volume, however. The name of the manufacturer or the type of card, for instance, may be important.

The [VolumeInfoType](#) structure returned from `VFSVolumeInfo` contains a `slotRefNum` field that can be passed to `ExpCardInfo`. This allows you to obtain specific information about the card on which a particular volume is located.

Obtaining volume information that corresponds to a given slot reference number is not quite so simple, because there is no function that returns the volume reference number given a slot reference number. If the volumes are slot-based, you can, however, iterate through the mounted volumes and check each volume's slot reference number. (To determine whether a volume is slot-based, in the `VolumeInfoType` structure check whether `mountClass` is set to `sysFileTSlotDriver` before accessing the volume specified by `slotRefNum`.)

Naming Volumes

Different handheld file system libraries support volume names of different maximum lengths and have different restrictions on character sets. The file system library is responsible for verifying whether or not a given volume name is valid, and returns an error if it is not. From a conduit developer's standpoint, volume names can be up to 255 characters long, and can include any printable character.

The file system library is responsible for translating the volume name into a format that is acceptable to the underlying file system. For example, when the 8.3 naming convention is used to translate a long volume name, the first eleven valid, non-space characters are used. Valid characters in this instance are A-Z, 0-9, \$, %, ', -, _, @, ~, ', !, (,), ^, #, and &.

For more information on naming volumes on handheld expansion cards, see the *Palm OS Programmer's Companion*.

File Operations

All of the familiar operations you use to operate on files in a desktop application are supported by the VFS Manager for handheld expansion cards; these are listed in "[Common File Operations](#)." In addition, the VFS Manager includes a set of functions that simplify the way you work with files that represent Palm OS record databases (PDB) or resource databases (PRC). These are covered in "[Working with Palm OS Databases](#)" on page 99.

This section covers the following topics:

- [Common File Operations](#)
- [Naming Files](#)
- [Working with Palm OS Databases](#)

Common File Operations

As shown in [Table 8.2](#), the VFS Manager provides all of the standard file operations that should be familiar from desktop and larger computer systems. Because these functions work largely as you would expect, their use is not detailed here. See the descriptions of each individual function in [Chapter 5, “Virtual File System Manager API,”](#) on page 173 in the *C/C++ Sync Suite Reference for Macintosh* for the parameters, return values, and side effects of each.

Note that some of these functions can be applied to both files and directories, while others work only with files.

Table 8.2 Common file operations

Function	Description
VFSFileOpen	Opens a file, given a volume reference number and a file path.
VFSFileClose	Closes an open file.
VFSFileRead	Reads data from a file into the specified buffer.
VFSFileWrite	Writes data to an open file.
VFSFileSeek	Sets the position within an open file from which to read or write.
VFSFileTell	Gets the current position of the file pointer within an open file.
VFSFileEOF	Gets the end-of-file status for an open file.
VFSFileCreate	Creates a file, given a volume reference number and a file path.

Table 8.2 Common file operations (*continued*)

Function	Description
VFSFileDelete	Deletes a closed file.
VFSFileRename	Renames a closed file.
VFSFileSize	Gets the size of an open file.
VFSFileResize	Changes the size of an open file.
VFSFileGetAttributes	Gets the attributes of an open file, including hidden, read-only, system, and archive bits. See “ File and Directory Attributes ” on page 176 in the <i>C/C++ Sync Suite Reference for Macintosh</i> for the bits that make up the attributes field.
VFSFileSetAttributes	Sets the attributes of an open file, including hidden, read-only, system, and archive bits.
VFSFileGetDate	Gets the created, modified, and last accessed dates for an open file.
VFSFileSetDate	Sets the created, modified, and last accessed dates for an open file.

Once a file has been opened, it is identified by a unique **file reference number**: a [FileRef](#). Functions that work with open files take a file reference. Others, such as `VFSFileOpen`, require a volume reference and a path that identifies the file within the volume. Note that all paths are volume relative, *and absolute within that volume*: the VFS Manager has no concept of a “current working directory,” so relative path names are not supported. The directory separator character is the forward slash: “/”. The root directory for the specified volume is specified by a path of “/”.

Naming Files

Different file systems support filenames and paths of different maximum lengths. The file system library is responsible for verifying whether or not a given path is valid and returns an error if it is not valid. From a conduit developer's standpoint, filenames can be up to 255 characters long and can include any normal character including spaces and lowercase characters in any character set. They can also include the following special characters:

`$ % ' - _ @ ~ ' ! () ^ # & + , ; = []`

The file system library is responsible for translating each filename and path into a format that is acceptable to the underlying file system. For example, when the 8.3 naming convention is used to translate a long filename, the following guidelines are used:

- The name is created from the first six valid, non-space characters which appear before the last period. The only valid characters are A-Z, 0-9, \$, %, ', -, _, @, ~, '!, (,), ^, #, and &.
- The extension is the first three valid characters after the last period.
- The end of the six byte name has "~1" appended to it for the first occurrence of the shortened filename. Each subsequent occurrence uses the next unique number, so the second occurrence would have "~2" appended, and so on.

The standard VFAT file system library provided with all Palm Powered handhelds that support expansion uses the above rules to create FAT-compliant names from long filenames.

Working with Palm OS Databases

Expansion cards are often used to hold Palm OS applications and data in PRC and PDB format. Because of the way that secondary storage media are connected to the Palm Powered handheld, applications cannot be run directly from the expansion card, nor can conduits manipulate databases using the Sync Manager without first transferring them to main memory with the VFS Manager. Conduits written to use the VFS Manager, however, can operate directly on files located on an expansion card.

Using Expansion Technology

Directory Operations

NOTE: Whenever possible give the same name to the PRC/PDB file and to the database. If the PRC/PDB filename differs from the database name, and the user copies your database from the card to the handheld and then to another card, the filename may change. This is because the database name is used when a database is copied from the handheld to the card.

Transferring Palm OS Databases to and from Expansion Cards

The [VFSEExportDatabaseToFile](#) function converts a database from its internal format on the handheld to its equivalent PRC or PDB file format and transfers it to an expansion card. The [VFSImportDatabaseFromFile](#) function does the reverse; it transfers the .prc or .pdb file to primary storage memory and converts it to the internal format used by Palm OS. Use these functions when moving Palm OS databases between main memory and an expansion card.

The VFSEExportDatabaseToFile and VFSImportDatabaseFromFile routines are atomic and, depending on the size of the database and the mechanism by which it is being transferred, can take some time.

Directory Operations

All of the familiar operations you would use to operate on directories are supported by the VFS Manager; these are listed in “[Common Directory Operations](#)” on page 101. One common operation — determining the files that are contained within a given directory — is covered in some detail in “[Enumerating the Files in a Directory](#)” on page 102. To improve data interchange with devices that are not running Palm OS, expansion card manufacturers have specified default directories for certain file types. “[Determining the Default Directory for a Particular File Type](#)” on page 103 discusses how you can both determine and set the default directory for a given file type.

This section covers the following topics:

- [Directory Paths](#)
- [Common Directory Operations](#)
- [Enumerating the Files in a Directory](#)
- [Determining the Default Directory for a Particular File Type](#)
- [Default Directories Registered at Initialization](#)

Directory Paths

All paths are volume relative, *and absolute within that volume*: the VFS Manager has no concept of a “current working directory,” so relative path names are not supported. The directory separator character is the forward slash: “/”. The root directory for the specified volume is specified by a path of “/”.

See “[Naming Files](#)” on page 99 for details on valid characters to use in file and directory names.

Common Directory Operations

As shown in [Table 8.3](#), the VFS Manager provides all of the standard directory operations that should be familiar from desktop and larger computer systems. Because these functions work largely as you would expect, their use is not detailed here. See the descriptions of each individual function [Chapter 5, “Virtual File System Manager API,”](#) on page 173 in the *C/C++ Sync Suite Reference for Macintosh* for the parameters, return values, and side effects of each.

Note that most of these functions can be applied to files as well as directories.

Table 8.3 Common directory operations

Function	Description
VFSDirCreate	Creates a new directory.
VFSFileDelete	Deletes a directory, given a path.
VFSFileRename	Renames a directory.
VFSFileOpen	Opens the specified directory.

Table 8.3 Common directory operations (*continued*)

Function	Description
VFSFileClose	Closes the specified directory.
VFSFileGetAttributes	Gets the attributes of an open directory, including hidden, read-only, system, and archive bits. See “ File and Directory Attributes ” on page 176 in the <i>C/C++ Sync Suite Reference for Macintosh</i> for the bits that make up the attributes field.
VFSFileSetAttributes	Set the attributes of an open directory, including hidden, read-only, system, and archive bits.
VFSFileGetDate	Get the created, modified, and last accessed dates for an open file.
VFSFileSetDate	Set the created, modified, and last accessed dates for an open file.

Enumerating the Files in a Directory

Enumerating the files within a directory is made with the [VFSDirEntryEnumerate](#) function. [Listing 8.5](#) illustrates the use of this function. Note that `volRefNum` must be declared and initialized prior to the following code.

Listing 8.5 Enumerating a directory’s contents

```
// Open the directory and iterate through the files in it.
// volRefNum must have already been defined.
FileRef dirRef;

err = VFSFileOpen (volRefNum, "/", vfsModeRead, &dirRef);
if(err == errNone) {
    // Iterate through all the files in the open directory
    UInt32 fileIterator;
    FileInfoType fileInfo;
    char *fileName = new char[256];    // Should check for err.
```

```
fileInfo.nameP = fileName;           // Point to local buffer.
fileInfo.nameBufLen = sizeof(fileName);
fileIterator = expIteratorStart;
while (fileIterator != expIteratorStop) {
    // Get the next file
    err = VFSDirEntryEnumerate(dirRef, &fileIterator,
                               &fileInfo);

    if(err == errNone) {
        // Process the file here.
    }
} else {
    // Handle directory open error here.
}
delete [] fileName;
}
```

Each time through the while loop, `VFSDirEntryEnumerate` sets the [FileInfoType](#) structure as appropriate for the file currently being enumerated. Note that if you want the filename, it is not enough to simply allocate space for the `FileInfoType` structure; you must also allocate a buffer for the filename, set the appropriate pointer to it in the `FileInfoType` structure, and specify your buffer's length. Because the only other information encapsulated within `FileInfoType` is the file's attributes, most conduits also want to know the file's name.

Determining the Default Directory for a Particular File Type

As explained in “[Standard Directories](#)” on page 85, the expansion capabilities of Palm OS include a mechanism to map MIME types or file extensions to specific directory names. This mechanism is specific to the slot driver: where an image might be stored in the “/Images” directory on a Memory Stick, on an MMC card it may be stored in the “/DCIM” directory. The VFS Manager includes a function that enables your conduit to get the default directory on a particular volume for a given file extension or MIME type; unlike handheld applications, conduits cannot register and unregister their own default directories.

The [VFSGetDefaultDirectory](#) function takes a volume reference and a string containing the file extension or MIME type and returns a string containing the full path to the corresponding

default directory. When specifying the file type, either supply a MIME media type/subtype pair, such as “image/jpeg”, “text/plain”, or “audio/basic”; or a file extension, such as “.jpeg”. As with most other such VFS Manager functions, you must pre-allocate a buffer to contain the returned path. Supply a pointer to this buffer along with the buffer’s length. The length is updated upon return to indicate the actual length of the path, which will not exceed the originally-specified buffer length.

The default directory registered for a given file type is intended to be the “root” default directory. If a given default directory has one or more subdirectories, conduits should also search those subdirectories for files of the appropriate type.

VFSGetDefaultDirectory allows you to determine the directory associated with a particular file suffix. However, there is no way to get the entire list of file suffixes that are mapped to default directories.

Default Directories Registered at Initialization

The VFS Manager registers the file types in [Table 8.4](#) under the `expMediaType_Any` media type, which [VFSGetDefaultDirectory](#) reverts to when there is no default registered by the slot driver for a given media type.

Table 8.4 Default directory registrations

File Type	Path
.prc	/PALM/Launcher/
.pdb	/PALM/Launcher/
.pqa	/PALM/Launcher/
application/vnd.palm	/PALM/Launcher/
.jpg	/DCIM/
.jpeg	/DCIM/
image/jpeg	/DCIM/
.gif	/DCIM/

Table 8.4 Default directory registrations (*continued*)

File Type	Path
image/gif	/DCIM/
.qt	/DCIM/
.mov	/DCIM/
video/quicktime	/DCIM/
.avi	/DCIM/
video/x-msvideo	/DCIM/
.mpg	/DCIM/
.mpeg	/DCIM/
video/mpeg	/DCIM/
.mp3	/AUDIO/
.wav	/AUDIO/
audio/x-wav	/AUDIO/

The SD slot driver provided by PalmSource, Inc. registers the file types in [Table 8.5](#), because it has an appropriate specification for these file types.

Table 8.5 Directories registered by the SD slot driver

File Type	Path
.jpg	/DCIM/
.jpeg	/DCIM/
image/jpeg	/DCIM/
.qt	/DCIM/
.mov	/DCIM/
video/quicktime	/DCIM/

Table 8.5 Directories registered by the SD slot driver

File Type	Path
.avi	/DCIM/
video/x-msvideo	/DCIM/

Although the directories registered by this SD slot driver all happen to be duplicates of the default registrations made by the VFS Manager, they are also registered under the SD media type because the SD specification explicitly includes them.

Slot drivers written by other Palm Powered handheld manufacturers that support different media types, such as Memory Stick, register default directories appropriate to their media's specifications. In some cases these registrations may override the `expMediaType_Any` media type registration, or in some cases augment the `expMediaType_Any` registrations with file types not previously registered.

These registrations are intended to aid applications developers, but you are not required to follow them. Although you can choose to ignore these registrations, by following them you improve interoperability between applications and other devices. For example, a digital camera which conforms to the media specifications puts its pictures into the registered directory (or a subdirectory of it) appropriate for the image format and media type. By looking up the registered directory for that format, an image viewer application on the handheld can easily find the images without having to search the entire card. These registrations also help prevent different developers from hard-coding different paths for specific file types. Thus, if a user has two different image viewer applications, both look in the same location and find the same set of images.

Registering these file types at initialization allows the default install conduit to transfer files of these types to an expansion card. During the HotSync process, files of the registered types are placed directly in the specified directories on the card.

Custom Calls

Recognizing that some file systems may implement functionality not covered by the APIs included in the VFS and Expansion Managers, the VFS Manager includes a single function that exists solely to give developers access to the underlying file system. This function, [VFSCustomControl](#), takes a registered creator ID and a selector that together identify the operation that is to be performed. `VFSCustomControl` can either request that a specific file system perform the specified operation, or it can iterate through all of the currently-registered file systems in an effort to locate one that responds to the desired operation.

Parameters are passed to the file system's custom function through a single `VFSCustomControl` parameter. This parameter, `pDataBuf`, is declared as a `void *` so you can pass a pointer to a structure of any type. A second parameter, `valueLenP`, allows you to specify the length of `pDataBuf`. Note that these values are simply passed to the file system and are in reality dependent upon the underlying file system. See the description of [VFSCustomControl](#) in the *C/C++ Sync Suite Reference for Macintosh* for more information.

Because `VFSCustomControl` is designed to allow access to nonstandard functionality provided by a particular file system, see the documentation provided with that file system for a list of any custom functions that it provides.

The rest of this section discusses "[Custom I/O](#)."

Custom I/O

While the Expansion and VFS Managers provide higher-level OS support for secondary storage, they do not attempt to present anything more than a raw interface to custom I/O applications. Because it is not possible to envision all uses of an expansion mechanism, the Expansion and VFS Managers simply try to get out of the way of custom hardware.

The Expansion Manager provides insertion and removal notification and can load and unload drivers. Everything else is the responsibility of the application developer. PalmSource, Inc. has defined a common expansion slot driver API which is extensible by

Using Expansion Technology

Custom Calls

Palm OS licensees. This API is designed to support all of the needs of the Expansion Manager, the VFS Manager, and the file system libraries. Conduits that need to communicate with an I/O device, however, may need to go beyond the provided APIs. Such conduits should wherever possible use the slot custom call, which provides direct access to the expansion slot driver. See the developer documentation provided to licensees for more information on slot drivers and the slot custom call. For documentation on functions made available by a particular I/O device, along with how you access those functions, contact the I/O device manufacturer.

NOTE: If a custom I/O device connects to the handheld via the same connector as the HotSync cradle, the custom I/O device is, of course, not accessible during a *cradle-based* HotSync operation. However, if the HotSync operation is via a *wireless* connection — for example, infrared — then a conduit can access such a device.

Summary of Expansion and VFS Managers

Expansion Manager Functions

ExpCardInfo	ExpSlotEnumerate
ExpCardPresent	ExpSlotMediaType

VFS Manager Functions

Working with Files

VFSFileClose	VFSFileRead
VFSFileCreate	VFSFileRename
VFSFileDelete	VFSFileResize
VFSFileEOF	VFSFileSeek
VFSFileGet	VFSFileSetAttributes
VFSFileGetAttributes	VFSFileSetDate
VFSFileGetDate	VFSFileSize
VFSFileOpen	VFSFileTell
VFSFilePut	VFSFileWrite

Working with Directories

VFSDirCreate	VFSFileOpen
VFSDirEntryEnumerate	VFSFileRename
VFSFileClose	VFSFileSetAttributes
VFSFileDelete	VFSFileSetDate
VFSFileGetAttributes	VFSGetDefaultDirectory
VFSFileGetDate	

Working with Volumes

VFSVolumeEnumerate	VFSVolumeSetLabel
VFSVolumeFormat	VFSVolumeSize
VFSVolumeGetLabel	VFSSupport
VFSVolumeInfo	

Miscellaneous Functions

VFSCustomControl	VFSGetAPIVersion
VFSExportDatabaseToFile	VFSSupport
VFSImportDatabaseFromFile	

Installing Conduits

For your conduit to run on a Macintosh computer, you need to create a conduit info ('CInf') resource for your conduit. This resource provides the information about your conduit that the HotSync® Manager application needs to run it.

The Conduit Information Resource

[Listing 9.1](#) shows the declaration for the conduit information resource.

The CDK includes a Resourceror template for the 'CInf' resource, in the Extras folder.

Listing 9.1 The conduit information resource

```
type 'CInf' {
    integer;                // conduit type
    longint;                // conduit version number
    literal longint;        // conduit creator ID
    pstring;                // conduit name shown to user
    pstring;                // conduit settings directory name
    integer;                // conduit priority
    integer;                // conduit has a user interface if nonzero
    integer = $$Countof(Stringarray);
    array StringArray{
        pstring;            // handheld database names
    };
    integer = $$Countof(Stringarray);
    array StringArray{
        pstring;            // desktop computer database names
    };
};
```

Installing Conduits

The Conduit Information Resource

The fields of the resource are used as shown below.

conduit type This is the type of conduit you are defining. Use one of the following values:

`nativeConduit = 1`

 This is a built-in conduit.

`nonNativeConduit = 1`

 This is a user-supplied conduit.

version number The long integer version number for your conduit. Pack your major version number into the high byte of the low word, and pack your minor version number into the low byte of the low word. *Note:* This must be 0x0300 or greater if your conduit is Carbonized.

conduit name A `pstring` that provides the name of the conduit shown to the user during synchronization operations.

creator ID The long integer ID associated with your conduit and with its companion application on the handheld.

directory name A `pstring` that specifies the name of the folder inside the user folder in which settings are stored for the conduit.

priority The priority of the conduit. The HotSync Manager application runs higher priority conduits before lower priority ones. Use one of the following values:

`lowestPriority = 0`

`lowPriority = 1`

`defaultPriority = 2`

`highPriority = 3`

`highestPriority = 4`

user interface Specifies whether your conduit provides a user interface (configuration) function. Use one of the following values:

`doesntWantUserInterface = 0`

`wantsUserInterface = 1`

handheld database names

An array of `pstrings`; one string for the name of each database on the handheld that the conduit uses.

desktop database names

An array of `pstrings`; one string for the file name of each database on the desktop computer that the conduit uses.

[Listing 9.2](#) shows an example of a conduit information resource. The conduit that uses this resource definition provides a user interface, runs at the default priority, and uses the specified data source on the handheld and desktop computer.

Listing 9.2 An example of a conduit information resource

```
resource 'CInf' (0)
{
    nonNativeConduit,
    0x0300,                // version number
    'Samp',                // creator ID
    "Memo Conduit",        // user visible conduit name
    "Memo",                // settings directory
    defaultPriority,        // conduit priority
    WantsUserInterface,    // conduit does have an interface
    {
        "MemoDB",          // remote (handheld) database names
    },
    {
        "Memo.dat",        // local (desktop) database names
    },
};
```

Conduit Domains

HotSync® Manager supports the new concept of “domains” under Mac OS X and Mac OS 9. There are three domains:

- the user domain
- the local domain
- the network domain

Carbon defines constants for these three domains in `Folders.h`:

- `kUserDomain`
- `kLocalDomain`
- `kNetworkDomain`

Currently, only the local domain is supported by HotSync Manager. Under Mac OS X, conduits are placed in `/Library/Application Support/Palm HotSync/Conduits`. Under Mac OS 9, conduits are installed in `<Volume Name>:System Folder:Application Support:Palm HotSync:Conduits`. However, you should use the `UmGetGlobalConduitsDirectory` function to determine where conduits should be installed.

NOTE: As a side effect of this architecture, conduits installed under Mac OS X are in a different location than conduits installed under Mac OS 9, so they won't be available on both operating systems unless they're installed in both locations.

Using the User Manager API

The User Manager API is the preferred way to access information in the users data file on the desktop. The users data file stores name, synchronization preferences, directory, and password information about users who have synchronized Palm Powered™ handhelds with the desktop computer.

NOTE: The Install Aide API has been deprecated in favor of the User Manager API. The User Manager API is available beginning with Palm Desktop 4.0, while the Install Aide API is not available with Palm Desktop 4.0 or later.

Installing Conduits

To determine where to install your conduit, use the `UmGetGlobalConduitsDirectory` function.

Installing Handheld Applications

To determine where your installer should place handheld applications to be installed the next time the user performs a HotSync operation, use the `UmGetFilesToInstallFolderSpec` function.

For details on the User Manager API functions, see [Chapter 6, “User Manager API,”](#) on page 233 in the *C/C++ Sync Suite Reference for Macintosh*.

Testing Your Conduit

Since one of the most important goals of conduits is simplicity for users, you need to ensure that your conduit does not interfere with other conduits. PalmSource, Inc. strongly recommends that, at a minimum, you test the following conduit installation and removal conditions:

- after installing your conduit, the user can press the HotSync button and have synchronization operations proceed correctly and without required intervention
- after removing (uninstalling) your conduit, the user can press the HotSync button and have synchronization operations proceed correctly and without required intervention

Conduit Cautions

In addition to verifying that your conduit installation and removal operations leave the user's desktop computer in fully operational condition, you need to be aware of the following situations that can cause your conduit to generate unpleasant side effects for users of Palm Powered™ handhelds:

- If your conduit stores state information on the handheld, you must remember to clean up the state information upon removal of your conduit.
- If you create data for a creator ID that you do not own, your conduit can generate very unpleasant results. Do not use a creator ID that does not belong to you.

Debugging Conduits

This chapter provides information about debugging your conduit, including the following:

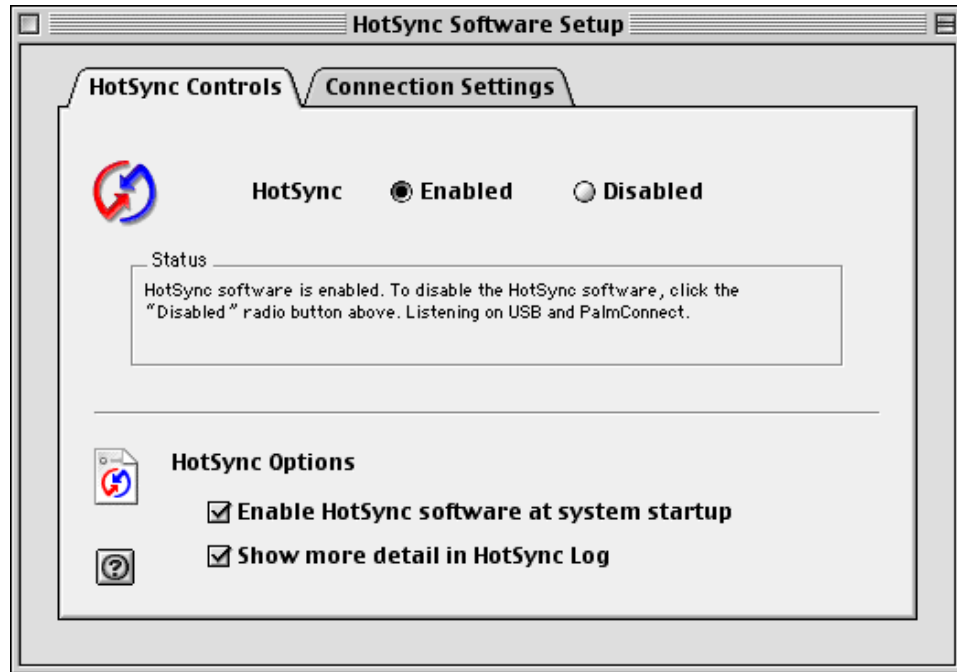
- how to use HotSync[®] Manager application to provide additional debugging information
- troubleshooting tips for common conduit problems
- shortcuts on the handheld that you can use for debugging
- how to disable time-outs and other conduits to help your debugging efforts
- information about additional tools you can use for debugging on Palm Powered[™] handhelds

Using the HotSync Manager Application Log

You can add entries to the HotSync Manager application's log to provide you with debugging information. You can also set the additional logging information option in the HotSync Manager application to provide additional information in the log. To do so, follow these steps:

1. Select the **Setup** command from the HotSync menu.
2. Select the **Show more detail** option in the dialog box, as shown in [Figure 10.1](#).

Figure 10.1 Enabling additional log information



Location of the Log File

The HotSync Manager application log file is named `HotSync Log` and is normally generated in the user's directory.

Debugging Tips and Tricks

This section provides several hints to help you in debugging your conduits, including the following:

- troubleshooting tips for common conduit problems
- how to use shortcut numbers
- how to avoid power-off timeouts
- how to disable other conduits

Common Troubleshooting Help

This section describes solutions to several common problems that developers have with getting their conduits working.

When Your Conduit Doesn't Appear in The Custom Dialog Box

You do not see your conduit in the Custom dialog box, it means that the HotSync Manager application has not loaded your conduit. This is usually because HotSync Manager can't find your conduit.

Also, all Carbonized conduits must have a 'carb' 0 resource.

When Your Conduit Doesn't Run

If your conduit displays in the HotSync Manager application's Conduit Settings dialog box, check to be sure the version number of your conduit is in the range specified by the constants `MIN_CONDUIT_VERSION` (0x00000101) and `MAX_CONDUIT_VERSION` (0x00000300).

The value that you return from your `GetConduitVersion` entry point must be in the range `MIN_CONDUIT_VERSION` to `MAX_CONDUIT_VERSION`. [Listing 7.5](#) shows an example of the `GetConduitVersion` function.

You Don't Hit Your Breakpoint

On Mac OS X, if your breakpoint is in `OpenConduit` or `OpenConduitCarbon`, your debugging host application must be Conduit Manager rather than HotSync Manager. For the other entry points, you must use HotSync Manager as your debugging host instead of Conduit Manager. This is due to the memory protection feature offered by Mac OS X.

On Mac OS 9, it doesn't matter.

Using Shortcut Numbers

The Palm OS® responds to a number of "hidden" shortcuts for debugging your programs. You generate each of these shortcuts by drawing characters on your Palm Powered handheld, or by drawing them in the emulator program, if you are using that for debugging your conduit.

Debugging Conduits

Debugging Tips and Tricks

To enter a shortcut number, follow these steps:

1. On your handheld, or in the emulator program, draw the shortcut symbol. This is a lowercase, cursive “L” character, drawn as follows:



2. Next, tap the stylus twice, to generate a dot (a period).
3. Next, draw a number character in the number entry portion of the handheld’s text entry area. [Table 10.1](#) shows the different shortcut numbers that you can use.

For example, to disable the automatic power-off feature of the handheld, enter the follow sequence:



Table 10.1 Shortcut Numbers

Number	Description	Notes
1	The handheld enters debugger mode, and waits for a low-level debugger to connect. A flashing square appears in the top left corner of the handheld.	This mode opens a serial port, which drains power over time. You must perform a soft reset to exit this mode.
2	The handheld enters console mode, and waits for communication, typically from a high-level debugger.	This mode opens a serial port, which drains power over time. You must perform a soft reset to exit this mode.

Table 10.1 Shortcut Numbers (*continued*)

Number	Description	Notes
3	The handheld's automatic power-off feature is disabled.	You can still use the handheld's power button to power it on and off. You must perform a soft reset to exit this mode.
4	Displays the user's name	None
5	Erases the user's name and User ID.	WARNING! When the handheld is next synchronized after using this shortcut, the HotSync Manager application thinks that it has never been synchronized before. This means that records will be duplicated unless you first perform a hard reset (press the reset button while holding the power key).
6	Displays the ROM build date and build time.	
7	Cycles through battery curves to allow adjustment of when the battery warnings appear.	Of limited effectiveness. Low battery warnings do not work well with NiCd batteries.

NOTE: Many of the debugging shortcuts leave the handheld in a mode that requires a soft reset. To perform a soft reset, press the reset button on the back of the handheld with a blunt instrument, such as a paper clip.

Disabling Timeouts

There are two timeouts that you can disable when debugging your conduits. [Table 10.2](#) provides a summary of each timeout.

Table 10.2 Disabling timeouts on the handheld

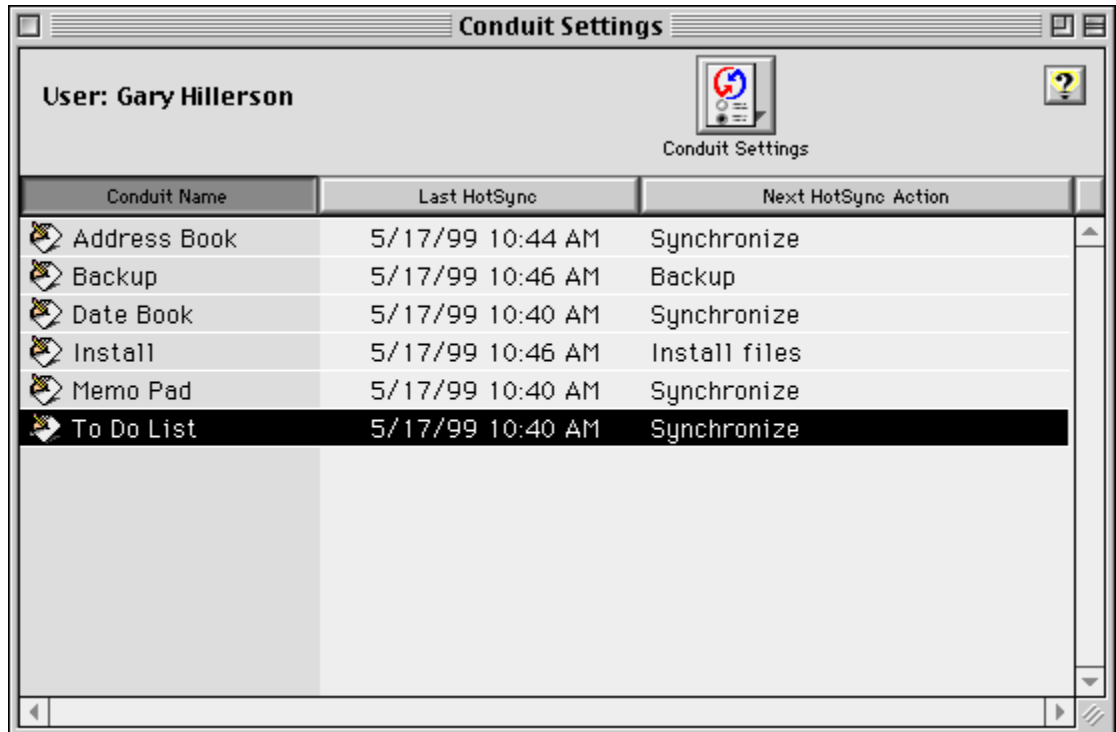
Timeout type	How to disable	Reason to disable
Automatic power-off	Use shortcut number 3, as described in Table 10.1 .	If you are stepping through your code and the handheld powers down, you lose your debugging session.
Transport link	Hold down the scroll-up button (on a Palm OS 3.0, or later, handheld) or the scroll-up and scroll-down buttons (on a pre-Palm OS 3.0 handheld) on the handheld and simultaneously tap in the top-right corner of the screen. <i>Note:</i> This doesn't work on devices with the universal connector.	If you are stepping through your code while synchronizing, the HotSync Manager application thinks the connection has been lost and terminates.

Disabling Other Conduits

To speed up your debugging efforts, you can disable all of the other conduits. To disable a conduit, follow these steps:

1. Choose the **Conduit Settings** menu choice in the HotSync Manager application
2. Select the conduit you want to disable. [Figure 10.2](#) shows selecting the To Do List conduit for disabling.

Figure 10.2 Selecting a conduit to disable



3. Click **Conduit Settings**.
4. Select the **Do Nothing** radio button, as shown in [Figure 10.3](#).
5. Also click the **Make Default** button, as shown in [Figure 10.3](#). This ensures that "Do Nothing" will remain effective until you unset the option.
6. Click **OK**.

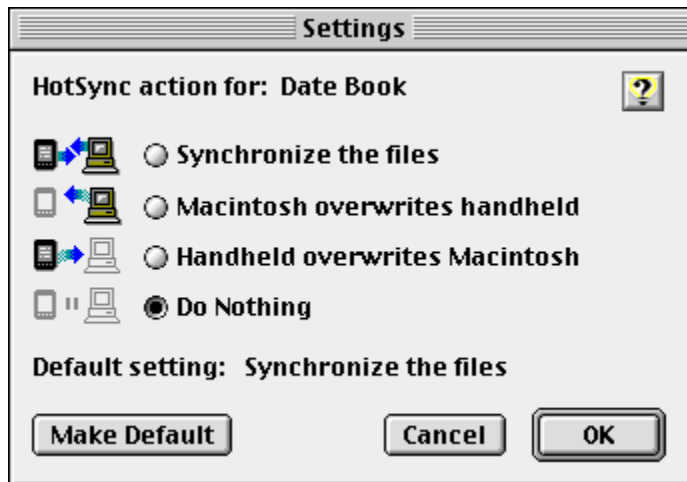
Perform this sequence of actions for each of your installed conduits.

IMPORTANT: Remember to reset each conduit after you've completed your debugging.

Debugging Conduits

Debugging Tips and Tricks

Figure 10.3 Changing the conduit action to Do Nothing



Glossary

This glossary defines terms used throughout the Conduit Development Kit documentation.

application conduit	A conduit associated with a third-party application, not an application integrated within the Palm™ Desktop software. Contrast with component conduit.
archiving	Saving deleted records on the desktop computer during synchronization operations. Archived records are not synchronized even if the user modifies them.
backup conduit	A built-in conduit that copies to the desktop any handheld databases and applications that have their backup bit set. The HotSync® Manager includes a backup conduit by default. Backup conduits are intended for handheld databases and applications that do not have synchronization conduits associated with them.
built-in applications	ROM-based Palm OS® applications that are built into the handheld. For example, the Memo Pad, Date Book, Mail, To do list, and Address book applications.
C API-based conduit	A conduit that is written in the C/C++ language with the C/C++ Sync Suite.
C/C++ Sync Suite	The components of the CDK for Windows used to create a C API-based conduit. It includes APIs, the C++ Generic Conduit framework, samples, documents, and utilities. It also includes APIs to access user information, to install databases and files on a handheld, and to perform other actions that do not necessarily happen during a HotSync operation.

CDK	Conduit Development Kit. PalmSource, Inc. provides one for Macintosh and one for Windows conduit development.
conduit	A plug-in module for the HotSync Manager application. Each conduit copies or synchronizes data for a certain application on the handheld with data on the desktop computer. In most instances, “conduit” refers to a synchronization conduit.
Conduit Configuration utility	A developer utility for registering and unregistering conduits and notifiers with HotSync Manager.
conduit entry points	Functions implemented in a conduit that HotSync Manager calls to start, to get information from, and otherwise to communicate with a conduit.
configuration entries	The configuration information stored on a Macintosh desktop computer about HotSync Manager and each conduit it runs during a HotSync operation. When an installer registers its conduit with HotSync Manager, it must use the APIs documented in the CDK to add a configuration entry.
cradle	The special holder that interfaces a Palm OS handheld with a desktop computer, usually via a serial connection. When you press the HotSync button on the cradle, HotSync Manager on the desktop begins synchronizing.
creator ID	The unique ID associated with each application on the handheld. Each conduit is associated with a specific creator ID.
database type	An application-specific value assigned to handheld databases to assist the Palm OS software or an application in identifying the database.
default conduits	Conduit supplied with the HotSync Manager application.
desktop application	An application program that runs on a Windows-based desktop computer.

desktop computer	A Windows-based or Macintosh computer. In conduit documentation, this term is often shortened to “desktop” and is used to distinguish it from the handheld.
direct cable connection	A connection between a desktop computer and a cradle that is achieved by connecting the cradle’s cable to a serial port on the desktop computer.
fast sync	A form of synchronization that is used when the most recent synchronization for the handheld was performed with the current desktop computer. In fast sync operations, only modified records should be copied between the desktop computer and handheld. A slow sync is performed when a single handheld is synchronized with multiple desktop computers. See slow sync.
Generic Conduit Framework	C++ classes provided in the CDK that implement synchronization logic more easily. PalmSource, Inc. highly recommends that you use the Generic Conduit for any mirror-image synchronization conduit.
handheld	A handheld computer (often used synonymously with “Palm Powered™ handheld.”
HotSync button	The button on the cradle that the user presses to initiate synchronization operations.
HotSync Manager	The desktop application that performs synchronization between a desktop computer and a handheld. Conduits are plug-in modules for this application.
install	When referring to a conduit, includes at least registering the conduit with HotSync Manager and copying the conduit file(s) to the desktop computer. The same installer could also install a desktop application and queue a handheld application (and databases) to be installed on the handheld during the next HotSync operation.
install conduit	A conduit that installs Palm OS applications and databases in a handheld’s main memory or transfers files between the desktop and handheld expansion cards. HotSync Manager includes an install conduit by default.

installed user	A user who has completed a HotSync operation so that the user's user ID is on both the desktop computer and a handheld.
mirror-image conduit	A conduit that performs mirror-image synchronization, which results in identical data for an application on the desktop computer and the handheld.
modem connection	One of the communications connections through which a user can perform a HotSync operation. In this connection, the handheld connects to the desktop computer using a modem.
multithreading	Multithreading makes it possible to run multiple threads, or client processes, that all connect with the same COM-based conduit.
native application	An application that is built into the handheld. For example, the built-in Date book application. Same as "built-in application."
network connection	One of the communications connections through which a user can perform a HotSync operation. In this connection, the handheld connects to the desktop computer using a local area network.
one-directional conduit	A conduit that copies data in one direction: either from the desktop computer to the handheld, or from the handheld to the desktop computer.
Palm™ Desktop software	The desktop computer application software supplied with Palm OS handhelds.
Palm OS	The operating system and services that run on a Palm Powered handheld.
Palm OS application	An application program that runs on a Palm OS handheld.
Palm OS platform	Includes the Palm OS, built-in applications, hardware architecture, conduits, and development tools.
Palm Powered handheld	Any handheld based on the Palm OS platform. Often shortened to "handheld" in Palm OS platform documentation.

PC ID	A pseudo-random number that HotSync Manager generates and uses to uniquely identify a desktop computer.
portable data entry	Data entry that the user performs on the portable handheld.
profile	See user profile.
record status flags	Flags in databases on the handheld that indicate whether an individual record has been modified since the most recent synchronization. Used for fast sync operations.
remote transactions	Data transactions performed by a user on a handheld.
slow sync	A form of synchronization that is used when the most recent synchronizations for the handheld and desktop computer were not performed with each other. In slow sync operations, a comprehensive comparison of handheld and desktop computer records is performed.
Sync Manager API	The collection of functions and data structures through which the desktop computer interacts with the handheld during synchronization operations. This lower-level API is called by the higher-level methods provided in the Generic Conduit Framework. The JSync and COM Sync software provide a bridge between the Sync Manager and a Java-based or COM-based conduit, respectively.
synchronization conduit	Performs whatever tasks a conduit developer designs it to do, typically synchronizing databases on the handheld with their desktop counterparts. Contrast with “install conduit” and “backup conduit.”
synchronization logic	Code that implements the synchronization actions you need your conduit to perform. The CDK provides the Generic Conduit Framework, which performs full, mirror-image synchronization. PalmSource, Inc. highly recommends that you use the Generic Conduit Framework, especially for any mirror-image synchronization conduit.

transaction-based conduit

A conduit that performs further logic-based operations beyond copying data between the handheld and the desktop computer. For example, a conduit that sends different data to the handheld based on data, or requests, that are found on the handheld. Another example is an email conduit that performs more complex operations than simply creating a mirror image.

user data

Information about desktop users' HotSync Manager settings. Use the User Manager API to access this information programmatically. Developers can use the User Info utility to view or modify user data during development.

user ID

An identification number associated with a handheld. This is a pseudo-random number generated and assigned to the handheld by HotSync Manager during the handheld's initial synchronization. The user ID value is independent of the user name. HotSync Manager assigns a user ID to a handheld when it synchronizes to either a handheld that has just been hard reset or to a handheld that has never been synchronized. When you synchronize such a handheld to a desktop computer that already has a user name and user ID defined, HotSync Manager displays a list of existing user names. After you select an existing name, HotSync Manager assigns that user name and its already associated user ID to the handheld. Any additional such handhelds synchronized with this desktop will undergo the same assignment process.

user profile

A special account created on the desktop computer for which someone — IT staff who support many handheld users — can enter data (company address book, for example) and queue applications and databases to be installed. When a handheld is synchronized to a user profile, the data is transferred to and applications installed on the handheld as with a regular user account — except that no user ID or PC ID is assigned to the handheld, nor are installed files cleared from the queue on the desktop. This behavior makes it easy to put the same data and applications on handhelds to be deployed to a group of users, who can each assign their own user names when they synchronize their handhelds the first time.

Index

A

- AppInfo block 43, 44
- application conduit, defined 125
- application name
 - on expansion cards 100
- applications
 - desktop 13
 - handheld 13
- architecture
 - expansion 80
- Archive bit 45, 46
- archiving 125

B

- Backup conduit 26
- backup conduit, defined 125
- built-in applications, defined 125

C

- C API-based conduit, defined 125
- C/C++ Sync Suite, defined 125
- card insertion and removal 86
- CardInfo application 95
- cards
 - expansion 79
- categories
 - CSynchronizer object, supported in 55
- Category 54
- Category classes 42
- CDK
 - defined 126
 - documentation x
 - handheld compatibility xiii
 - installing 1
 - version compatibility xii
- CInf resource 4, 35, 111
 - conduit name field 112
 - conduit type field 112
 - creator ID field 112
 - declaration of 111
 - desktop database names field 113
 - directory name field 112
 - example of 113
 - handheld database names field 113

- priority field 112
 - user interface field 113
 - version number field 112
- class hierarchy 42
- cleaning up records 61
- closing databases 62
- communications API 10
- communications between handheld and desktop 57
- CompactFlash 78
- compatibility
 - CDK/handheld xiii
- conduit
 - operation 39–40
- Conduit Configuration utility, defined 126
- conduit design goals 7
- conduit development 31
 - basics 9, 34
 - cleaning up records 61
 - closing databases 62
 - customizing the CInf resource 4
 - customizing the entry points 5
 - design goals 7
 - entry points 63
 - environments xiii
 - naming your project 2
 - opening databases in 58
 - project window 3
 - quick start 1
 - reading modified records 61
 - reading records 59
 - registering conduit with the Sync Manager 58
 - tools 14
 - troubleshooting 119
 - using stationery 2
 - working with databases 59
 - writing database records 61
- conduit development environments xiii
- Conduit Domains 114
- conduit entry points 5, 63
- conduit entry points, defined 126
- Conduit Information resource. *See* CInf resource
- conduit manager component xii
- conduit name field 112
- conduit tasks 32

- conduit type field 112
- conduit types 31
- conduit, defined 126
- conduits
 - about 31
 - and HotSync 21
 - brief description of 14
 - cautions 116
 - creating new project 2
 - debugging 117
 - design decisions 36
 - design goals 34
 - design questions 36
 - determining which to run 25
 - developing 31
 - development basics 1, 9, 34
 - development tools xiii, 14
 - entry points 5, 15
 - installing 111
 - interfacing with HotSync Manager
 - application 63
 - mirror-image 32
 - not appearing in custom dialog box 119
 - not running during synchronization 119
 - one directional 33
 - programming 31
 - project window 3
 - quick start to developing 1
 - resources 4, 15
 - running 25
 - stationery 2
 - synchronization types 32
 - tasks 32
 - testing 3, 115
 - transaction 33
 - troubleshooting 119
- ConfigureConduit 35, 65
- connection types 9, 30
 - direct cable 30
 - modem 30
- CopyActionStringAsCFStringRef 67
- CopyConduitNameAsCFStringRef 70
- CPalmRecord class 43
- CPcMgr 54
- CPcMgr class 43
- CPCustomRecord class 43

- cradle, defined 126
- creator ID 22
 - defined 126
- creator ID field 112
- CSynchronizer 44

D

- database creator ID 25
- Database Manager classes 42
- database type, defined 126
- databases
 - closing 62
 - on expansion cards 99
 - opening 58
 - reading modified records from 61
 - reading records from 59
 - working with 59
 - writing records to 61
- debugging conduits 117
- disabling other conduits 122
- disabling time outs 122
- hints 118
- HotSync flags 117
- log file 118
- shortcut numbers 119
- default conduits, defined 126
- default directories 86
 - determining by file type 103
 - for SD slot driver 105
 - registered upon initialization 104
- Delete bit 45, 46
- design decisions 36
- design goals 34
- desktop
 - synchronizing to more than one 48
- desktop application, defined 126
- desktop applications 13
- desktop computer, defined 127
- desktop database names field 113
- development environments xiii
- development tools xiii
- direct cable connection 10, 30
 - defined 127
- directories
 - basic operations 100

- default for file type 103
- enumerating files within 102
- directory name field 112
- disabling conduits 122
- disabling time outs 122
- double modify 46, 47

E

- entry points
 - ConfigureConduit 65
 - GetActionString 66
 - GetConduitName 69
 - GetConduitVersion 72
 - OpenConduit 72
- expansion 77–109
 - and Palm OS databases 99
 - architecture 80
 - auto-start PRC 86
 - custom calls 107
 - custom I/O 107
 - default directories 86
 - enumerating slots 91
 - file system operations 96
 - file systems 83
 - mounted volumes 90
 - naming applications on expansion cards 100
 - ROM 79
 - slot driver 82
 - slot reference number 82
 - standard directories 85
 - standard directory layout 85
 - volume operations 93
- expansion cards 79
 - capabilities 92
 - checking for 87
 - in slots 92
 - insertion and removal 86
- Expansion Manager 78, 84
 - checking card capabilities 92
 - custom I/O 107
 - enumerating slots 91
 - purpose 85
 - slot reference number 82
 - verifying presence of 87
- expansion slot 79
- ExpCardInfo 93

- ExpCardPresent 92
- expMediaType_Any 104
- ExpSlotEnumerate 91
- ExtractCategories method 55

F

- Fast Sync
 - defined 50
- fast sync, defined 127
- fast synchronization 28
- FastSync 28
- FAT 83
- features 40
- file systems 83
 - and filenames 99
 - and volume names 96
 - basic operations 96
 - custom calls to 107
 - FAT 83
 - implementation 83
 - long filenames 83
 - multiple 83
 - nonstandard functionality 107
 - VFAT 83, 99
- filenames
 - long 83
- files
 - enumerating 102
 - naming 83, 96, 99
 - paths to 98
 - referencing 98
- flags 117
- formatting volumes 93

G

- GenCn 39, 40
- Generic Conduit
 - class hierarchy 42
 - features 40
- Generic Conduit Framework 39, 40
- GENERIC_FLAG_CATEGORY_SUPPORTED
 - flag 55
- GetActionString 35, 66
- GetConduitName 35, 69
- GetConduitVersion 35, 72

H

- handheld applications 13
- handheld compatibility xiii
- handheld database names field 113
- handheld, defined 127
- HotSync button, defined 127
- HotSync flags 117
- HotSync log 75
 - enabling additional information 118
- HotSync Manager
 - defined 127
- HotSync Manager application xii, 31
 - brief description of 13
 - conduit manager component xii
 - defined 21
 - determining configuration 24
 - determining type 28
 - interfacing with conduits 63
 - list of conduits 25
 - running conduits 25
 - serial monitor component xii
- HotSync operations 22
- HotSync process 39–40
 - control flow 44

I

- install conduit, defined 127
- install, defined 127
- installed user, defined 128
- installing conduits 111

L

- log file 118
- LogAddEntry 76
- Logging class 43
- logging entries 75
- logging functions 75

M

- manager classes 42
- mapping file types to directories 103
- Memory Stick 78, 103
- MIME types 103
- mirror-image conduit 32

- mirror-image conduit, defined 128
- modem connection 10, 30
 - defined 128
- Modify bit 45, 46
- MultiMediaCard (MMC) 78
- multithreading, defined 128

N

- naming conventions 96, 99
- native application, defined 128
- network connection 10
- network connection, defined 128

O

- one directional conduit 33
- one directional synchronization 33
- one-directional conduit, defined 128
- OpenConduit 35, 72
- OpenConduit entry point 44
- OpenConduitCarbon 74

P

- Palm Desktop software, defined 128
- Palm OS application, defined 128
- Palm OS Platform 9, 11
- Palm OS platform 39–40
- Palm OS platform, defined 128
- Palm OS, defined 128
- Palm Powered handheld, defined 128
- PC ID 22
- PC ID, defined 129
- PCMgr class
 - custom file format 43
- PDB files
 - on expansion cards 99
- plug and play slot driver 80
- portable data entry, defined 129
- PRC files
 - on expansion cards 99
- primary storage 78
- priority field 112
- profile, defined 129
- project window 3

Q

quick start 1

R

RAM

- expansion 78

reading modified records 61

reading records 59

Record classes 42

record status flags, defined 129

record-level synchronization 28

records

- reading 59, 61

records, synchronization logic 46

remote transactions, defined 129

RetrieveDB method 56

ROM

- expansion 79

S

SD slot driver 105

secondary storage 78

Secure Digital (SD) 78

serial monitor component xii

shortcut numbers 119

slot custom call 108

slot driver 82, 106

- accessing directly 108

- plug and play 80

slot ID 82

slot reference number 92

slots 79

- and volumes 94, 95

- checking for a card 92

- enumerating 91

- referring to 82

Slow Sync

- defined 50

slow sync, defined 129

slow synchronization 28

SlowSync 28

soft reset 121

software components 11

standard directories on expansion media 85

start.prc 86

stationery 2

storage

- primary 78

- secondary 78

Sync Manager

- and the VFS Manager 84

Sync Manager API 16

- capabilities 57

- communications 57

- defined 129

- logging functions 75

- using 57

synchronization

- fast 50

- logic, records 46

- slow 50

synchronization conduit, defined 129

synchronization logic, defined 129

synchronization types 32

- one directional 33

- transaction-based 33

Synchronizer classes 43

T

testing your conduit 115

time outs 122

tools for conduit development 14

transaction-based conduit 33

transaction-based conduit, defined 130

transaction-based synchronization 33

troubleshooting 119

U

universal connector 79

user data, defined 130

user ID 22, 24

- defined 130

user interface field 113

user profile, defined 130

users

- synchronizing multiple 48

V

- version compatibility xii
- version number field 112
- VFAT 83
- VFS Manager 78, 84
 - and the Sync Manager 84
 - custom calls 107
 - custom I/O 107
 - directory operations 100
 - enumerating files 102
 - file paths 98
 - file system operations 96
 - filenames 99
 - verifying presence of 87
 - volume operations 93
- VFSCustomControl 107
- VFSDirCreate 101
- VFSDirEntryEnumerate 102
- VFSExportDatabaseToFile 100
- VFSFileClose 97, 102
- VFSFileCreate 97
- VFSFileDelete 98, 101
- VFSFileEOF 97
- VFSFileGetAttributes 98, 102
- VFSFileGetDate 98, 102
- VFSFileOpen 97, 101
- VFSFileRead 97
- VFSFileRename 98, 101
- VFSFileResize 98
- VFSFileSeek 97
- VFSFileSetAttributes 98, 102

- VFSFileSetDate 98, 102
- VFSFileSize 98
- VFSFileTell 97
- VFSFileWrite 97
- VFSGetDefaultDirectory 103
- VFSImportDatabaseFromFile 100
- VFSVolumeEnumerate 91
- VFSVolumeFormat 93
- VFSVolumeGetLabel 94
- VFSVolumeInfo 94, 95
- VFSVolumeSetLabel 94
- VFSVolumeSize 94
- Virtual File System. *See* VFS Manager
- volumes
 - and slots 94
 - automatically mounted 93
 - basic operations 93
 - enumerating 90
 - formatting 93
 - hidden 94, 95
 - labeling 94
 - matching to slots 95
 - mounted 90
 - mounting 93
 - naming 96
 - read-only 94
 - size 94
 - space available 94
 - unmounting 93

W

- writing records 61