



# **COM Sync Suite Companion**

**Palm OS® Conduit Development Kit for  
Windows, Version 6.0.1**

Written by Brent Gossett.

Technical assistance from Ravikumar Duggaraju and Giovanni Marais.

Copyright © 2000-2004, PalmSource, Inc. and its affiliates. All rights reserved. This technical documentation contains confidential and proprietary information of PalmSource, Inc. ("PalmSource"), and is provided to the licensee ("you") under the terms of a Nondisclosure Agreement, Product Development Kit license, Software Development Kit license or similar agreement between you and PalmSource. You must use commercially reasonable efforts to maintain the confidentiality of this technical documentation. You may print and copy this technical documentation solely for the permitted uses specified in your agreement with PalmSource. In addition, you may make up to two (2) copies of this technical documentation for archival and backup purposes. All copies of this technical documentation remain the property of PalmSource, and you agree to return or destroy them at PalmSource's written request. Except for the foregoing or as authorized in your agreement with PalmSource, you may not copy or distribute any part of this technical documentation in any form or by any means without express written consent from PalmSource, Inc., and you may not modify this technical documentation or make any derivative work of it (such as a translation, localization, transformation or adaptation) without express written consent from PalmSource.

PalmSource, Inc. reserves the right to revise this technical documentation from time to time, and is not obligated to notify you of any revisions.

THIS TECHNICAL DOCUMENTATION IS PROVIDED ON AN "AS IS" BASIS. NEITHER PALMSOURCE NOR ITS SUPPLIERS MAKES, AND EACH OF THEM EXPRESSLY EXCLUDES AND DISCLAIMS TO THE FULL EXTENT ALLOWED BY APPLICABLE LAW, ANY REPRESENTATIONS OR WARRANTIES REGARDING THIS TECHNICAL DOCUMENTATION, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION ANY WARRANTIES IMPLIED BY ANY COURSE OF DEALING OR COURSE OF PERFORMANCE AND ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, ACCURACY, AND SATISFACTORY QUALITY. PALMSOURCE AND ITS SUPPLIERS MAKE NO REPRESENTATIONS OR WARRANTIES THAT THIS TECHNICAL DOCUMENTATION IS FREE OF ERRORS OR IS SUITABLE FOR YOUR USE. TO THE FULL EXTENT ALLOWED BY APPLICABLE 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, EXEMPLARY OR PUNITIVE DAMAGES OF ANY KIND ARISING OUT OF OR IN ANY WAY RELATED TO THIS TECHNICAL DOCUMENTATION, INCLUDING WITHOUT LIMITATION DAMAGES FOR LOST REVENUE OR PROFITS, LOST BUSINESS, LOST GOODWILL, LOST INFORMATION OR DATA, BUSINESS INTERRUPTION, SERVICES STOPPAGE, IMPAIRMENT OF OTHER GOODS, COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR OTHER FINANCIAL LOSS, EVEN IF PALMSOURCE, INC. OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR IF SUCH DAMAGES COULD HAVE BEEN REASONABLY FORESEEN.

PalmSource, the PalmSource logo, BeOS, Graffiti, HandFAX, HandMAIL, HandPHONE, 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, Palm Pack, PalmPak, PalmPix, PalmPower, PalmPrint, Palm.Net, Palm Reader, Palm Talk, Simply Palm and ThinAir 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 TECHNICAL DOCUMENTATION IS PROVIDED ON A COMPACT DISC, THE SOFTWARE AND OTHER DOCUMENTATION ON THE COMPACT DISC ARE SUBJECT TO THE LICENSE AGREEMENTS ACCOMPANYING THE SOFTWARE AND OTHER DOCUMENTATION.

COM Sync Suite Companion

Document Number 3021-006

May 14, 2004

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](http://www.palmsource.com)

# Table of Contents

---

<b>About This Document</b>	<b>vii</b>
Related Documentation . . . . .	viii
What this Document Contains . . . . .	ix
Changes to This Document . . . . .	ix
Document 3021-006 for CDK 6.0.1 . . . . .	x
Document 3021-005 for CDK 6.0 . . . . .	x
Additional Resources . . . . .	xii
Conventions Used in this Document . . . . .	xii
Programming Style in This Manual . . . . .	xiii
 <b>1 Introduction</b>	 <b>1</b>
COM-based Conduit Fundamentals. . . . .	2
Features . . . . .	3
Programming Languages . . . . .	5
COM Sync Suite Files . . . . .	6
Uninstalling the COM Sync Suite . . . . .	6
 <b>2 Development Basics</b>	 <b>7</b>
HotSync Process . . . . .	7
COM Sync Architecture . . . . .	9
COM Sync Client/Server DLLs. . . . .	10
Client Module Types . . . . .	12
Multithreading Support . . . . .	13
COM Sync Object Model . . . . .	14
System Information Objects . . . . .	15
Schema Database Objects . . . . .	16
Extended Database Objects . . . . .	18
Classic Database Objects . . . . .	19
Installation and Support Objects . . . . .	22
Expansion Card Objects. . . . .	24
COM ProgIDs . . . . .	25
COM Error Handling . . . . .	26
Database Support and Extensibility . . . . .	28

---

<b>3 Building a Conduit</b>	<b>31</b>
Requirements . . . . .	33
Create the Debug Environment . . . . .	33
Synchronizing with Palm OS Simulator . . . . .	36
Start a Visual Basic Project . . . . .	36
Create the Conduit . . . . .	39
Debug the Conduit . . . . .	43
Convert Your Conduit to an ActiveX Client . . . . .	43
Implement the IPDClientNotify Interface . . . . .	45
Developing a Conduit with Visual Basic .NET . . . . .	48
Registration Steps . . . . .	49
 <b>4 Writing an Installer</b>	 <b>51</b>
Installer Tasks . . . . .	52
Files to Install . . . . .	53
Sample for Registering a COM-based Conduit . . . . .	55
Installing Files on the Handheld with PDInstall. . . . .	58
How PDInstall Works. . . . .	58
Install Directory Terminology . . . . .	60
Expansion Slot Support in PDInstall . . . . .	62
Controlling HotSync Manager with PDHotSyncUtility. . . . .	63
Accessing User Data with PDUserData . . . . .	64
Installing an Install Conduit with PDInstallConduit. . . . .	65
Registering an Install Conduit . . . . .	66
Unregistering an Install Conduit . . . . .	66
Adding or Modifying Install Conduit Configuration Entries . . . . .	66
Installing a Notifier with PDCondMgr. . . . .	67
Uninstalling Your Conduit. . . . .	67
Testing Your Installer . . . . .	68
Installation Troubleshooting Tips . . . . .	69
 <b>5 Using Expansion Technology</b>	 <b>71</b>
Expansion Support . . . . .	72
Primary vs. Secondary Storage . . . . .	72
Expansion Slot . . . . .	73
Universal Connector . . . . .	74

---

Architectural Overview . . . . .	75
Slot Drivers . . . . .	77
File Systems . . . . .	77
VFS Manager . . . . .	78
Expansion Manager . . . . .	80
Standard Directories . . . . .	81
Card Insertion and Removal . . . . .	82
Checking for Expansion Cards . . . . .	83
Is an Expansion Slot Present? . . . . .	84
How Many Slots are Present? . . . . .	87
Is a Card in the Slot? . . . . .	87
Is a Mounted Volume on the Card? . . . . .	88
Example of Checking for an Expansion Slot, Card, and Volume	89
Determining a Card's Capabilities . . . . .	90
Volume Operations . . . . .	91
Hidden Volumes . . . . .	93
Matching Volumes to Slots . . . . .	94
Naming Volumes. . . . .	95
File Operations . . . . .	95
Common File Operations . . . . .	96
Naming Files . . . . .	98
Working with Palm OS Databases . . . . .	98
Directory Operations . . . . .	99
Directory Paths . . . . .	100
Common Directory Operations. . . . .	100
Enumerating the Files and Subdirectories in a Directory . . .	102
Determining the Default Directory for a Particular File Type .	102
Default Directories Registered at Initialization . . . . .	103
Example of Getting a File . . . . .	105
Custom Calls. . . . .	107
Debugging. . . . .	108
Summary of Expansion and VFS Managers . . . . .	109



# About This Document

---

The COM Sync Suite is a component of the Palm OS® Conduit Development Kit (CDK) for Windows from PalmSource, Inc. It provides COM objects/methods/properties, the IPDClientNotify interface, COM Sync software layer, documents, and utilities to help developers create COM-based conduits that run on Windows 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 *COM Sync Suite Companion* introduces the basic concepts required to create a COM-based conduit using the COM Sync Suite and guides you through the development process.

The sections in this introduction are:

<a href="#">Related Documentation</a>	viii
<a href="#">What this Document Contains</a>	ix
<a href="#">Changes to This Document</a>	ix
<a href="#">Conventions Used in this Document</a>	xii
<a href="#">Additional Resources</a>	xii

## About This Document

### *Related Documentation*

---

## Related Documentation

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

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

The following documents are part of the CDK:

Document	Description
<a href="#"><i><u>Introduction to Conduit Development</u></i></a>	An introduction to conduits on the Windows platform. It describes how they relate to other aspects of the Palm OS platform, how they communicate with the HotSync <sup>®</sup> Manager, and how to choose an approach to conduit development. Recommended reading for developers new to conduits.
<a href="#"><i><u>C/C++ Sync Suite Companion</u></i></a>	An overview of how C API-based conduits operate and how to develop them with the C/C++ Sync Suite.
<a href="#"><i><u>C/C++ Sync Suite Reference</u></i></a>	A C API reference that contains descriptions of all conduit function calls and important data structures used to develop conduits with the C/C++ Sync Suite.
<a href="#"><i><u>COM Sync Suite Companion</u></i></a>	An overview of how COM-based conduits operate and how to develop them with the COM Sync Suite.
<a href="#"><i><u>COM Sync Suite Reference</u></i></a>	A reference for the COM Sync Suite object hierarchy, detailing each object, method, and property.
<a href="#"><i><u>Conduit Development Utilities Guide</u></i></a>	A guide to the CDK utilities that help developers create and debug conduits for Windows.



## What this Document Contains

This section provides an overview of the major parts of this document and the chapters in each.

[Chapter 1, “Introduction.”](#) Describes the COM Sync suite fundamentals, features, and uninstalling the COM Sync Suite.

[Chapter 2, “Development Basics.”](#) Describes basic information about developing COM-based conduits.

[Chapter 3, “Building a Conduit.”](#) Describes how to build a conduit using Visual Basic.

[Chapter 4, “Writing an Installer.”](#) Describes general tips for writing an installer and uninstaller for your COM-based conduit and provides troubleshooting suggestions.

[Chapter 5, “Using Expansion Technology.”](#) Describes how to work with handheld expansion cards from the desktop using the Expansion and Virtual File System (VFS) Managers.

## Changes to This Document

This section describes significant changes made in each version of this document. For a description of what's new in each version of the CDK, see [Chapter 1, “What's New in the Palm OS CDK,”](#) on page 1 in *Introduction to Conduit Development*.

- [Document 3021-006 for CDK 6.0.1](#)
- [Document 3021-005 for CDK 6.0](#)

## About This Document

### *Changes to This Document*

---

## Document 3021-006 for CDK 6.0.1

The significant corrections and additions in this document version are listed by chapter below:

- [Chapter 1, "Introduction."](#)
  - Added [PDSystemCondMgr](#) and [PSDDatabaseUtilities](#).
- [Chapter 3, "Building a Conduit."](#)
  - Noted that COM-based conduits can opt out of the conduit/application requirement when run by HotSync Manager 6.0 or later.
  - Noted several requirements for developers using Visual Basic .NET.
- [Chapter 4, "Writing an Installer."](#)
  - Revised "[Files to Install](#)" on page 53 to describe what CDK binaries you are permitted to redistribute with your conduit to enable it to work on systems running versions of HotSync Manager earlier than 6.0.
  - Noted that you can use [PDSystemCondMgr](#) to register a conduit for all users (the system) in exactly the same way you can use [PDCondMgr](#) to register a conduit for the current Windows user.

## Document 3021-005 for CDK 6.0

The significant corrections and additions in this document version are listed by chapter below:

- [Chapter 1, "Introduction."](#)
  - Added new schema and extended database types.
  - Updated "[COM Sync Suite Files](#)" on page 6.
- [Chapter 2, "Development Basics."](#)
  - Added new COM Sync library files that provide support for schema and extended databases. [Table 2.1](#) on page 11 lists these libraries.
  - Added object models for schema and extended databases to "[COM Sync Object Model](#)" on page 14.

- Updated “[Database Support and Extensibility](#)” on page 28 to include schema and extended databases.
- [Chapter 3, “Building a Conduit.”](#)
  - Noted that HotSync Manager versions 6.0 and later does not require a COM-based conduit to have an application with the same creator ID on the handheld.
  - Modified this chapter to refer to the SimpleDb tutorial project for schema databases, in particular “[Create the Conduit](#)” on page 39.
- [Chapter 4, “Writing an Installer.”](#)
  - Noted that this chapter describes how to install a conduit on a desktop running HotSync Manager versions 6.0 and later. Refer to the documentation for CDK 4.03 for the same information relevant to earlier HotSync Manager versions.
  - Noted that HotSync Manager versions 6.0 and later do not require that you refresh/restart HotSync Manager after registering a conduit.
  - Noted that PDCondMgr can register conduits only as user conduits, not as system conduits.
  - Updated “[Installing Files on the Handheld with PDInstall](#)” on page 58 for HotSync Manager 6.0 and later.
  - Noted that the new [PSDDatabaseQuery.InstallDatabase\(\)](#) method can install files when called, instead of using PDInstall to queue them for later installation.
  - Removed information about PalmCOMInstaller.exe, which is no longer required, because HotSync Manager versions 6.0 and later include COM Sync libraries, so that you do not have to deploy them with your conduit.
- [Chapter 5, “Using Expansion Technology.”](#) Referenced the Card Explorer sample and warned of the problem with the import and export methods not working with schema databases.

## About This Document

### *Additional Resources*

---

## Additional Resources

- Documentation  
PalmSource, Inc. 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/>
- CDK Feedback  
Use this email address to provide feedback on the CDK: features you would like to see, bug reports, errors in documentation, and requests for Knowledge Base articles.  
[cdk-feedback@palmsource.com](mailto:cdk-feedback@palmsource.com)

## Conventions Used in this Document

This guide uses the following typographical conventions:

<b>This style...</b>	<b>Is used for...</b>
<code>sample</code>	Literal text such as filenames, commands, code elements such as functions, structures, and so on.
<i>sample</i>	Emphasis or to indicate a variable.
<b>sample</b>	Definition or first usage of a term, menu and menu item names, user-supplied text, window names in UI descriptions.
<a href="#">sample</a>	Hypertext links.

**This style...**

Sub, If, Case Else,  
Print, Long

**setup**

*event-driven*

*variable*

[*expressionlist*]

{While | Until}

```
Sub  
HelloButton_Click()  
Readout.Text = _  
"Hello, world!"  
End Sub
```

```
C:\Program  
Files\COMConduit.dll
```

->

<-

<->

**Is used for...**

Words with initial letter capitalized indicate Visual Basic language-specific keywords.

Words you're instructed to type appear in bold.

In text, italic letters can indicate defined terms, usually the first time they occur in the book. Italic formatting is also used occasionally for emphasis.

In prototype and text, italic letters can indicate placeholders for information you supply.

In prototype, items inside square brackets are optional.

In prototype, braces and a vertical bar indicate a choice between two or more items. You must choose one of the items unless all of the items are optional and are enclosed in square brackets.

This font is used for code and prototypes.

Paths and filenames are given in mixed case.

Parameter is passed into a function.

Parameter is passed out of a function.

Parameter passed in and out of a function.

## Programming Style in This Manual

The following guidelines are used in writing programs in this manual.

- Keywords appear with initial letters capitalized:  

```
' Sub, If, ChDir, Print, and True are keywords.  
Print "Hello World"
```

## About This Document

### *Conventions Used in this Document*

---

- An apostrophe (') introduces comments:
  - ' These lines are comments
  - ' Comments are ignored when the program
  - ' is running.
- Control-flow blocks and statements in Sub, Function, and Property procedures are indented from the enclosing code:

---

```
Sub InsertWORD(Record as Variant, Value as Integer)
  Dim Utility As New PDUtility
  Dim NextOffset As Long

  ' Extract the value
  NextOffset = Utility.ByteArrayToWORD(Record, 0, False, Value)
End Sub
```

---

- Lines too long to fit on one line (except comments) may be continued on the next line using a line-continuation character, which is a single leading space followed by an underscore ( \_ ) and the following line is indented as shown in the example below:

---

```
Sub Form_MouseDown (Button As Integer, _
  Shift As Integer, X As Single, Y As _
  Single)
```

---

# Introduction

---

The COM Sync Suite is part of the Palm OS® Conduit Development Kit (CDK) for Windows from PalmSource, Inc. This suite uses Microsoft's Component Object Model (COM) or Distributed COM (DCOM) technology, which is widely used by Windows applications to communicate and interoperate with other applications and external data sources. For more information about COM technology, see the Microsoft web site. To develop a COM-based conduit, background in COM/DCOM is required.

This suite allows you to directly access information on Palm Powered™ handhelds from a desktop computer using any COM-enabled programming environment and language such as Visual Basic (VB), Visual C/C++, Borland C++ Builder, Borland Delphi, and Java.

---

**NOTE:** While the COM Sync Suite can be used in any programming language that supports COM, this document is written with Visual Basic developers in mind.

---

This chapter describes the COM Sync Suite fundamentals including:

<a href="#">COM-based Conduit Fundamentals</a>	2
<a href="#">Features</a>	3
<a href="#">Programming Languages</a>	5
<a href="#">COM Sync Suite Files</a>	6
<a href="#">Uninstalling the COM Sync Suite</a>	6

---

**NOTE:** This document assumes that you have read the [Introduction to Conduit Development](#), which provides background information on the HotSync® process and conduit development.

---

## Introduction

### *COM-based Conduit Fundamentals*

---

For the COM Sync Suite programming reference information, see the [COM Sync Suite Reference](#), which provides the definition and syntax of each object, method, and property.

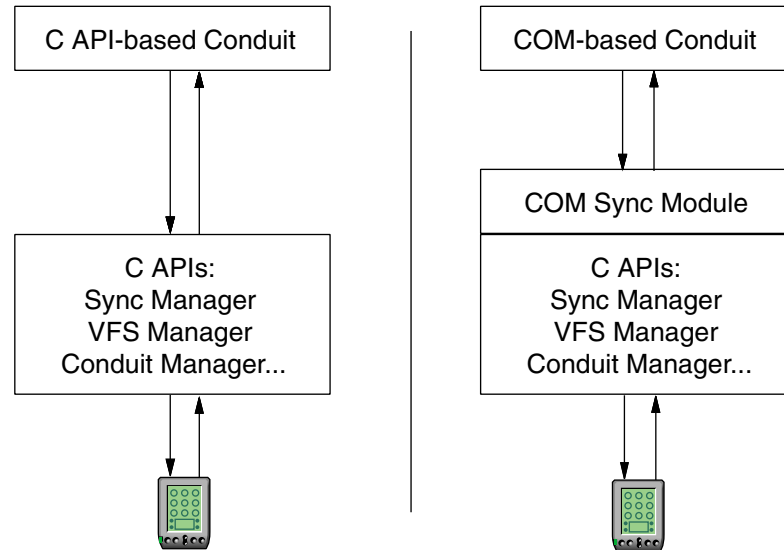
## COM-based Conduit Fundamentals

Conduits exchange and synchronize data between an application running on a desktop computer and a Palm Powered handheld. Conduits are plug-ins to the HotSync® Manager application running on the desktop computer. HotSync Manager runs all registered conduits when the user puts the handheld into the cradle and presses the HotSync button. Conduits that plug in directly to HotSync Manager are called [C API-based conduits](#).

The **COM Sync module** consists of DLLs that act as client and server during the HotSync process. The server is a single DLL and a standard C API-based conduit. The server DLL “wraps” the underlying C APIs (like Sync Manager, VFS Manager, Conduit Manager, and others) with a COM interface, while the client exposes that COM interface using an object model. The client DLLs supply you with an object model that communicates with the server. [Figure 1.1](#) compares how standard C API-based conduits and COM-based conduits communicate with the handheld.



**Figure 1.1 C API-based and COM-based conduits**



The COM Sync module extends the Sync Manager API functionality, with which your conduit can open and access as many databases as you wish—including schema, extended, and even classic databases. Each database object has a categories object permitting easy category management. There is a utility object that permits easy insertion and extraction of data into the record byte streams.

## Features

[Table 1.1](#) describes the COM Sync Suite features and functionality.

**Table 1.1 COM-based conduit features**

COM-based Conduit Feature	Functionality
Debug conduits in real time	You can debug your conduits using your IDE, keeping the handheld on-line. You have real-time access to your handheld data from your IDE without HotSync Manager timing out.
COM Sync controls are written in C++/ATL without MFC	You do not need to use MFC, or download any MFC or VB run-time files.

## Introduction

### Features

---

**Table 1.1 COM-based conduit features (*continued*)**

COM-based Conduit Feature	Functionality
Calls to update the HotSync Manager UI are not necessary	With the COM Sync Suite, your conduit does not have to update the HotSync Manager user interface during synchronization. C API-based conduits must occasionally call <code>SyncYieldCycles()</code> to do this.
Multithreading support	More than one application can talk to the handheld simultaneously.
Open multiple databases concurrently	You can open as many handheld databases as you want. You are not restricted to using only one database during synchronization as C API-based conduits are when working with classic databases.
Multiple abstraction models supported	The COM Sync module presents an abstraction model similar to the Sync Manager API.
Extensible architecture	You can write your own record adapter on top of the COM Sync client, creating your own abstraction model.
DCOM support	Ability to access a Palm Powered handheld across a network.
Use COM Sync Suite with any COM-compliant programming language	You can use Visual Basic or other COM-enabled languages to create a COM-based conduit.
COM Sync permits you to build conduits in multiple forms	You can develop your code as one of three module forms: standard EXE, ActiveX EXE, and ActiveX DLL.

## Programming Languages

The COM Sync Suite is designed for Visual Basic developers but because the COM Sync module exposes standard COM objects, it can also be used with other languages, including:

- Visual Basic (VB)
- Visual Basic for Applications (VBA)
- Borland C/C++ Builder
- Borland Delphi
- Java
- Microsoft Visual C/C++ (MFC is not required)
- Any other COM-compliant language

---

**NOTE:** While the COM Sync Suite is usable by any programming language that uses COM objects, this document is written with Visual Basic users in mind. This document assumes that you have the basic programming skills to take advantage of the provided standard Visual Basic projects and source code.

---

## COM Sync Suite Files

[Table 1.2](#) describes the COM Sync Suite files installed on your computer.

**Table 1.2 COM Sync Suite files**

Files	Description
Samples	<CDK>\COM\Samples contains Visual Basic samples that work with classic, extended, and schema databases, as well as a filesystem on an expansion card.
COMConduit.dll COMDirect.dll COMStandard.dll PSDConduit.dll DmConduit.dll	<CDK>\Common\Bin\ contains these COM Sync libraries. These libraries ship with HotSync Manager 6.0 and later.
Documentation	If Visual Studio .NET is installed, all CDK documentation is integrated in the VS help system. Or open <CDK>\Common\Docs\PalmOSCDK6.0.chm to view all CDK documents in HTML Help format.

## Uninstalling the COM Sync Suite

Do not delete the COM Sync Suite (or other CDK) files; uninstall them. To uninstall the CDK:

1. Open the Control Panel.
2. Select **Add/Remove Programs**.
3. Select **Palm OS Conduit Development Kit**.

You cannot uninstall the COM Sync Suite by itself.

4. Select **Remove**.

The entire Conduit Development Kit is removed.

# Development Basics

---

This chapter describes the following conduit development basics:

<a href="#">HotSync Process</a>	7
<a href="#">COM Sync Architecture</a>	9
<a href="#">COM Sync Client/Server DLLs</a>	10
<a href="#">Client Module Types</a>	12
<a href="#">Multithreading Support</a>	13
<a href="#">COM Sync Object Model</a>	14
<a href="#">COM ProgIDs</a>	25
<a href="#">COM Error Handling</a>	26
<a href="#">Database Support and Extensibility</a>	28

## HotSync Process

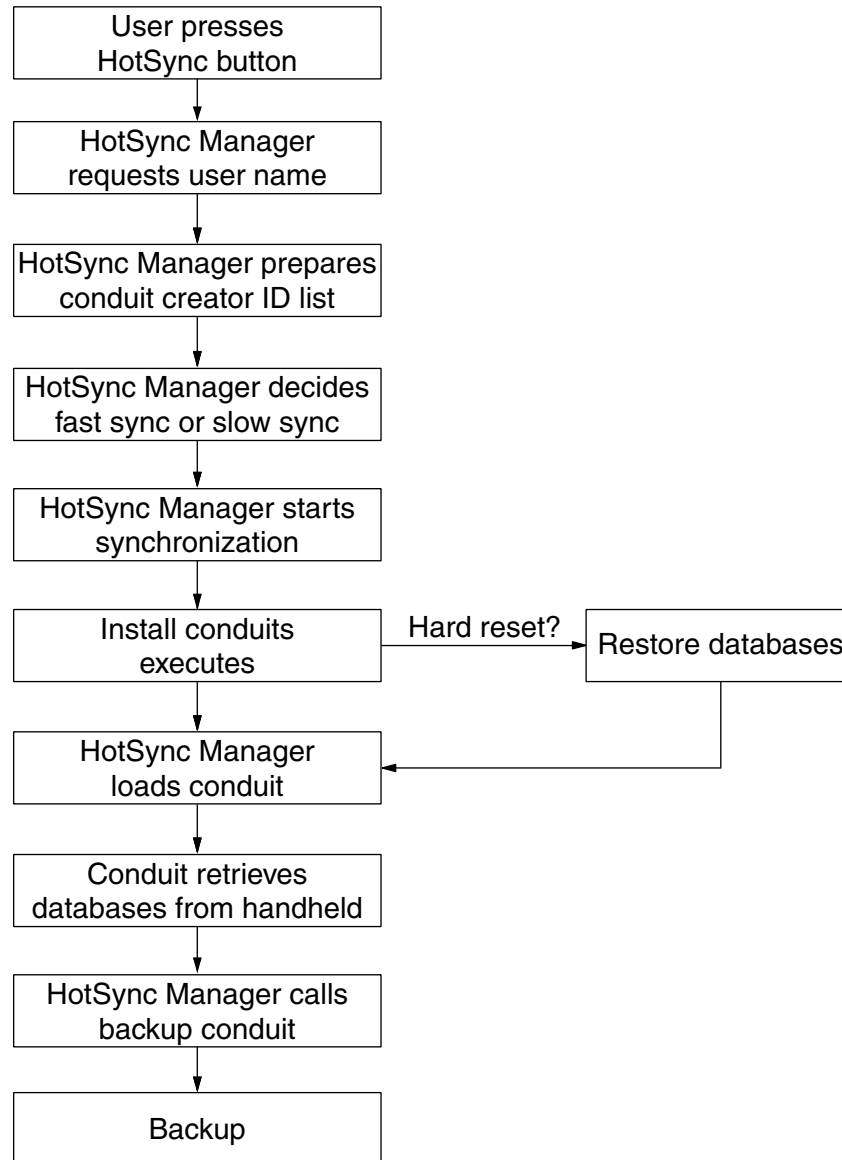
Good conduit development begins with understanding the HotSync® process itself and how the conduit processes information between the handheld and the desktop computer. [Figure 2.1](#) summarizes the HotSync process that begins when the user presses the HotSync button.

---

**NOTE:** For a detailed discussion, see [Chapter 7](#), “[Understanding the HotSync Process](#),” on page 85 in the *Introduction to Conduit Development*.

---

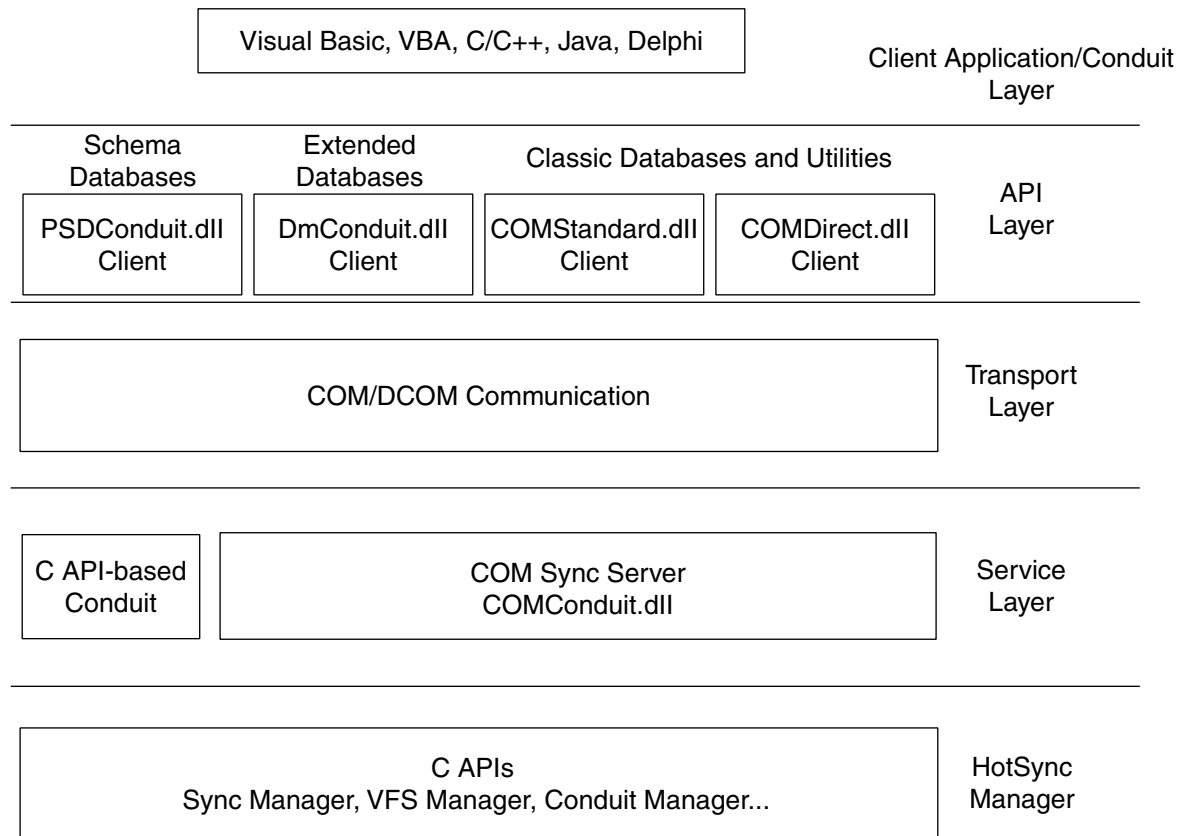
**Figure 2.1 HotSync process summary**



## COM Sync Architecture

The COM Sync architecture facilitates development of COM-based conduits to retrieve information from databases during the HotSync process. [Figure 2.2](#) depicts the COM Sync architecture.

**Figure 2.2 COM Sync architecture**



The following sections describe the components of the COM Sync architecture components.

## COM Sync Client/Server DLLs

The COM Sync module is composed of several DLLs that act as client and server during the HotSync process. The server is a single DLL and is itself a standard C API-based conduit.

The client consists of several DLLs that combine to supply an object model that communicates with the server, providing you another level of abstraction above the Sync Manager and other C APIs. The server DLL “wraps” the C APIs with a COM interface, while the client exposes that COM interface to you through an object model.

---

**IMPORTANT:** Because `COMConduit.dll` is a standard C API-based conduit, it is activated and executed *only* during normal HotSync operations. You will not be able to execute your conduit code outside of a HotSync operation because the COM Sync objects are not available.

---

`COMConduit.dll`, acting as the server, handles all Sync Manager API execution. The client DLLs—`COMDirect.dll`, `COMStandard.dll`, `PSDConduit.dll`, and `DmConduit.dll`—communicate with `COMConduit.dll` while supplying a programming interface for you to use when developing your applications or conduits.

During the HotSync process, HotSync Manager runs `COMConduit.dll` like it does any other C API-based conduit. The COM Sync module activates a client process called the client module. The client module uses the COM Sync object model to communicate with the handheld, through the `COMConduit.dll`.

The server and client DLLs are discussed below.



### Server

The `COMConduit.dll` is an object server. For non-ActiveX clients, the object server remains active as long as there are active objects, or as long as the original client is still alive. This means the original client can spawn a new process and then quit. For ActiveX clients, the object server remains active as long as there are active client objects.

### Client

[Table 2.1](#) lists the client DLLs, which are loaded into the process space of your client module. These combine to expose a set of objects and an object model used to perform database operations on the handheld. These DLLs establish and maintain a connection with `COMConduit.dll` using COM/DCOM.

**Table 2.1 COM Sync client DLLs**

Client DLL	Library Name	Description
<code>COMDirect.dll</code>	<code>PDDirectLib</code>	Palm OS Classic Databases Exposes objects to work with <a href="#">classic databases</a> on the handheld and files on an expansion card.
<code>COMStandard.dll</code>	<code>PDStandardLib</code>	Palm OS Classic Databases and Utilities Exposes additional objects to work with classic databases, to register conduits, to install files on the handheld, and to perform other utility tasks.
<code>PSDConduit.dll</code>	<code>PSDCONDUITLib</code>	Palm OS Schema Databases Exposes objects to work with <a href="#">schema databases</a> on the handheld and to perform other utility tasks.
<code>DmConduit.dll</code>	<code>DMCONDUITLib</code>	Palm OS Extended Databases Exposes objects to work with <a href="#">extended databases</a> on the handheld and to perform other utility tasks.

## Client Module Types

The COM Sync Suite defines a client notification interface, [IPDClientNotify](#), that enables `COMConduit.dll` to notify or activate your client module when HotSync Manager is ready to run it, or when the user selects it via HotSync Manager's **Custom** menu. You are not required to use this interface.

Clients that use the `IPDClientNotify` interface can be either EXEs or DLLs and are referred to as ActiveX clients. Those clients that do not implement the interface must be EXEs (DLLs cannot be supported unless they implement the interface) and are referred to as standard EXE clients.

These two client types are discussed below.

### Standard EXE Client

The standard EXE client allows you to debug your conduit within your preferred IDE. A standard EXE is simpler to code, because there is no implementation of the notification interface.

---

**TIP:** When you develop your conduit as a standard EXE client module, you can debug your conduit with the handheld online. After everything is working properly, you can convert your application to an ActiveX EXE or ActiveX DLL.

---

### ActiveX Client

The ActiveX client allows you to use the [CfgConduit\(\)](#) method to configure your conduit from HotSync Manager's **Custom** menu and provides location independence through DCOM. In other words, you can access the conduit from anywhere on the network through DCOM.

## Multithreading Support

COM Sync multithreading makes it possible to run multiple threads, or client processes, that all connect with the same COM-based conduit. Because HotSync Manager does not support multithreading, the COM Sync module serializes all handheld requests to best leverage the standard HotSync process.

If the client process is an EXE, top-level objects are created in separate threads. DLL clients manage their own threading model, which must be set to "Apartment Threaded." For all clients, the COM Sync thread keeps the HotSync Manager progress indicator updated.

The threading capabilities combined with the use of executable client modules, ActiveX or not, make a powerful development and processing environment. For example:

- During the development stages, the COM Sync module executes your development environment instead of your conduit. This is a great way to debug your conduit from within your IDE.
- Because the COM Sync module takes care of updating the HotSync Manager progress indicator (`SyncYieldCycles()`), you can debug your conduits without HotSync Manager appearing to freeze.
- Conduits provide user interface, or other tasks, at the same time they are synchronizing remote databases using separate threads or processes.

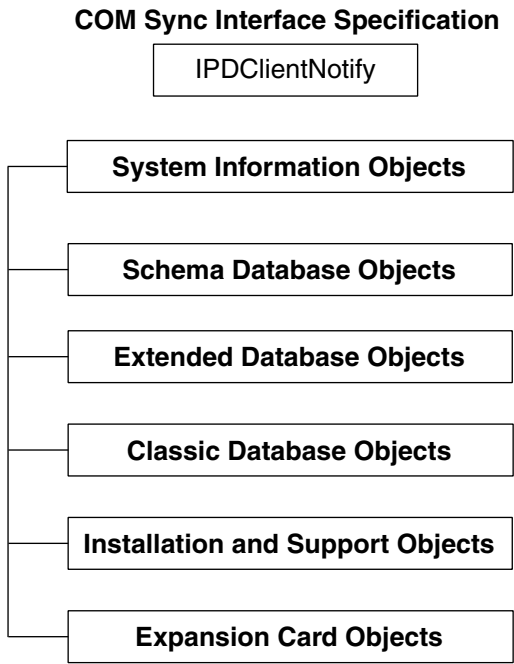
# COM Sync Object Model

The COM Sync object model consists of the following types of objects as shown in [Figure 2.3](#) and in the following subsections. The [IPDClientNotify](#) interface specification defines methods that your COM-based conduit must implement so that HotSync Manager can run your conduit during a HotSync operation.

The following subsections describe each group of related objects:

- [System Information Objects](#) . . . . . 15
- [Schema Database Objects](#) . . . . . 16
- [Extended Database Objects](#) . . . . . 18
- [Classic Database Objects](#) . . . . . 19
- [Installation and Support Objects](#) . . . . . 22
- [Expansion Card Objects](#) . . . . . 24

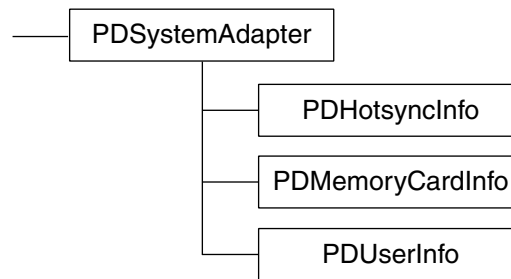
**Figure 2.3    COM Sync interface and object model**



## System Information Objects

System information objects consist of one top-level object, [PDSystemAdapter](#), and its child objects handle system-level data. Methods for these objects in turn use the underlying Sync Manager API.

**Figure 2.4 System information objects**



[Table 2.2](#) summarizes the purpose of these objects.

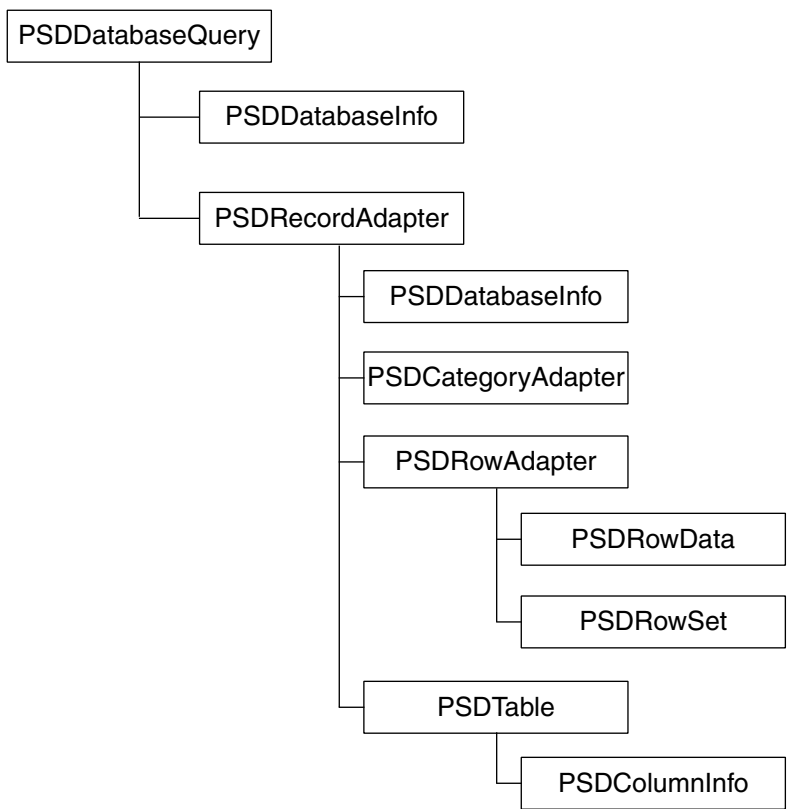
**Table 2.2 System information objects**

Object	Description
<a href="#">PDSystemAdapter</a>	Access to system methods and information.
<a href="#">PDHotsyncInfo</a>	Information supplied by the HotSync® Manager to the conduit.
<a href="#">PDMemoryCardInfo</a>	Memory card information.
<a href="#">PDUserInfo</a>	HotSync user information.

## Schema Database Objects

Schema database objects consist of the top-level [PSDDatabaseQuery](#) object that handles various [schema database](#) operations. Methods for these objects in turn call the underlying schema Sync Manager API.

**Figure 2.5** Schema database objects



[Table 2.3](#) summarizes the purpose of these objects.

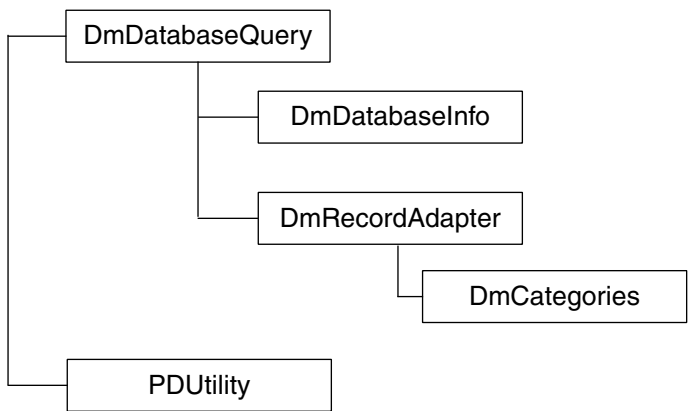
**Table 2.3 Schema database objects**

<b>Object</b>	<b>Description</b>
<a href="#"><u>PSDDatabaseQuery</u></a>	Schema database management, access, and information. Some of this objects's methods are specific to schema databases
<a href="#"><u>PSDDatabaseAdapter</u></a>	Methods to manage rows, groups of rows with the same schema (tables), columns, and each row's category membership.
<a href="#"><u>PSDDatabaseInfo</u></a>	Information about a single schema database.
<a href="#"><u>PSDRowAdapter</u></a>	Methods and properties for access to the rows in a single open schema database.
<a href="#"><u>PSDCategoryAdapter</u></a>	Methods and properties to access categories in a single schema database.
<a href="#"><u>PSDTable</u></a>	Methods and properties to access a single table in a schema database.
<a href="#"><u>PSDColumnInfo</u></a>	Information about a single column in a schema database table.
<a href="#"><u>PSDRowData</u></a>	Methods and properties to access a single row in a schema database table.
<a href="#"><u>PSDRowSet</u></a>	Methods and properties to access a set of rows returned by some <a href="#"><u>PSDRowAdapter</u></a> methods.

## Extended Database Objects

Extended database objects consist of the top-level [DmDatabaseQuery](#) object that handles various [extended database](#) operations. Methods for these objects in turn call the underlying extended Sync Manager API.

**Figure 2.6 Extended database objects**



[Table 2.4](#) summarizes the purpose of these objects.

**Table 2.4 Extended database objects**

Object	Description
<a href="#">DmDatabaseQuery</a>	Extended database management, access, and information.
<a href="#">DmDatabaseInfo</a>	Information about a single extended database.
<a href="#">DmRecordAdapter</a>	Methods and properties for access to the records in a single open extended database.
<a href="#">DmCategories</a>	Wrapper object for the application info block permitting easy access to the database categories.
<a href="#">PDUtility</a>	Byte array access, print formatting, and other useful utility methods.



## Classic Database Objects

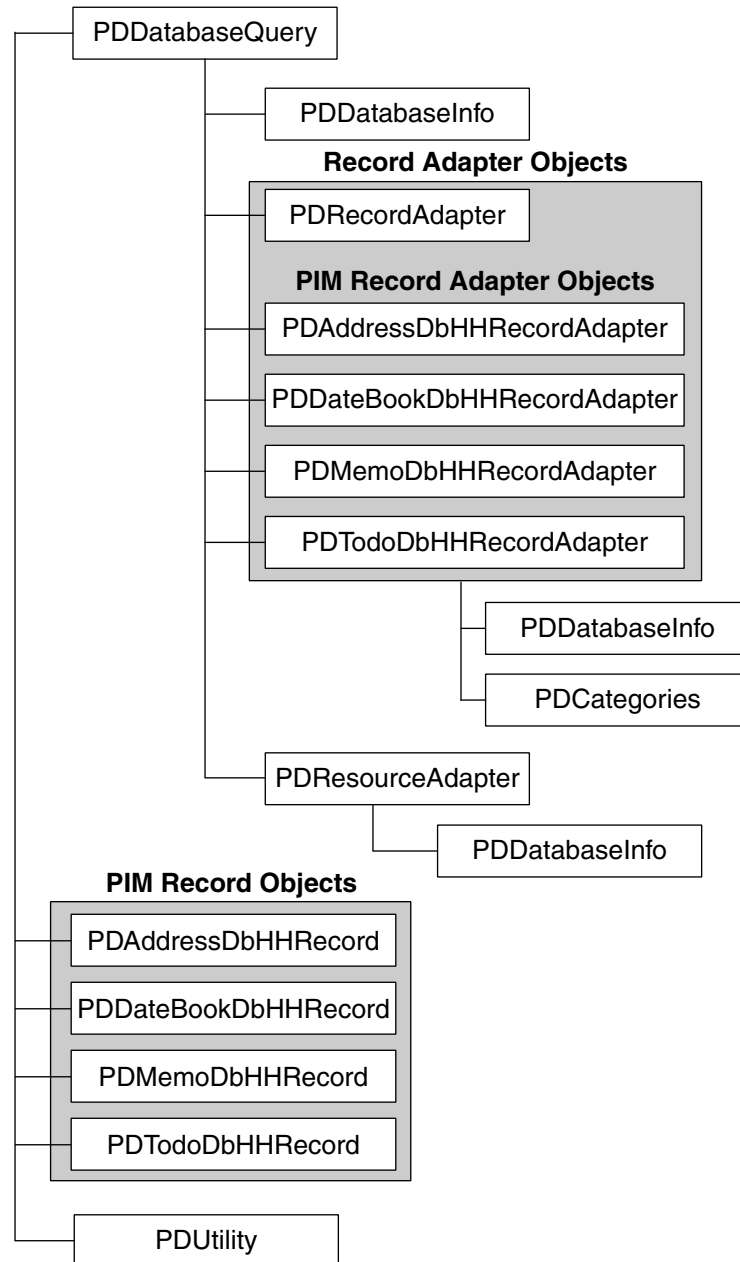
Classic database objects consist of the top-level [PDDatabaseQuery](#) object that handles various database operations along with record objects that handle the databases of the four classic Palm OS® PIM applications (Address Book, Date Book, Memo Pad, and To Do List). Methods for these objects in turn call the underlying Sync Manager API. Another top-level object, [PDUtility](#), provides various utility methods for things like converting a byte array to a string.

---

**NOTE:** The classic PIM applications are available only on some handhelds running versions of Palm OS earlier than Palm OS Cobalt. These PIM record objects do not work with the PIM applications that ship with Palm OS Cobalt.

---

**Figure 2.7 Classic database objects**



[Table 2.5](#) summarizes the purpose of these objects.

**Table 2.5 Classic database objects**

Object	Description
<a href="#">PDDatabaseQuery</a>	Classic database management, access, and information.
<a href="#">PDDatabaseInfo</a>	Information about a single classic database.
<a href="#">PDRecordAdapter</a>	Methods and properties for access to the records in a single open classic database.
<a href="#">PDCategories</a>	Wrapper object for the application info block permitting easy access to the database categories.
<a href="#">PDResourceAdapter</a>	Methods and properties for access to a single open resource database.
<a href="#">PDUtility</a>	Byte array access, print formatting, and other useful utility methods.
<a href="#">PDAddressDbHHRecordAdapter</a>	Methods and properties for access to a single open Address Book database. Use <a href="#">PDDatabaseQuery</a> to obtain this object.
<a href="#">PDDateBookDbHHRecordAdapter</a>	Methods and properties for access to a single open Date Book database. Use <a href="#">PDDatabaseQuery</a> to obtain this object.
<a href="#">PDMemoDbHHRecordAdapter</a>	Methods and properties for access to a single open Memo Pad database. Use <a href="#">PDDatabaseQuery</a> to obtain this object.
<a href="#">PDToDoDbHHRecordAdapter</a>	Methods and properties for access to a single open To Do List database. Use <a href="#">PDDatabaseQuery</a> to obtain this object.
<a href="#">PDAddressDbHHRecord</a>	Represents the structure of a record in the Address Book database on the handheld.
<a href="#">PDDateBookDbHHRecord</a>	Represents the structure of a record in the Date Book database on the handheld.

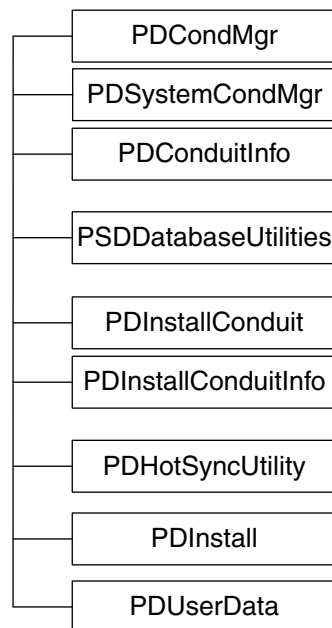
**Table 2.5 Classic database objects (*continued*)**

Object	Description
<a href="#">PDMemoDbHHRRecord</a>	Represents the structure of a record in the Memo Pad database on the handheld.
<a href="#">PDToDoDbHHRRecord</a>	Represents the structure of a record in the To Do List database on the handheld.

## Installation and Support Objects

Installation and support objects consist of top-level objects that provide methods for installing conduits on the desktop, databases to the handheld's primary storage, and files to the handheld's expansion cards. They also provide access to information about HotSync Manager users on the desktop. Methods for these objects in turn call the underlying Conduit Manager, Install Conduit Manager, Notifier Install Manager, Install Aide, User Data, HotSync Manager, and some Sync Manager APIs.

**Figure 2.8 Installation and support objects**



[Table 2.6](#) summarizes the purpose of these objects.

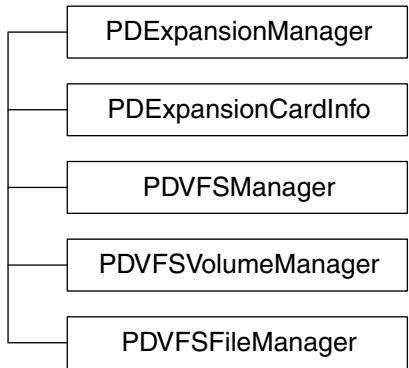
**Table 2.6 Installation and support objects**

<b>Object</b>	<b>Description</b>
<a href="#"><u>PDCondMgr</u></a>	Methods to register a conduit for the current Windows user during installation.
<a href="#"><u>PDSystemCondMgr</u></a>	Methods to register a conduit for the system during installation.
<a href="#"><u>PDConduitInfo</u></a>	Properties of a conduit required to register it with HotSync Manager.
<a href="#"><u>PSDDatabaseUtilities</u></a>	Methods for backing up and installing databases during a HotSync operation, determining desktop trust status, and managing security data.
<a href="#"><u>PDInstallConduit</u></a>	Methods to register an install conduit during installation.
<a href="#"><u>PDInstallConduitInfo</u></a>	Properties of an install conduit required to register it with HotSync Manager.
<a href="#"><u>PDHotSyncUtility</u></a>	Methods for controlling the HotSync Manager application.
<a href="#"><u>PDInstall</u></a>	Methods for queuing databases (including applications) to be installed on the handheld (or files to be copied to an expansion card) during the next HotSync operation.
<a href="#"><u>PDUserData</u></a>	Methods for accessing the users data store on the desktop computer.

## Expansion Card Objects

Expansion card objects consist of top-level objects that represent handheld expansion cards and the volumes and files stored on them. Methods for these objects in turn call the underlying Expansion Manager and Virtual File System (VFS) Manager APIs.

**Figure 2.9    Expansion card objects**



[Table 2.7](#) summarizes the purpose of these objects.

**Table 2.7    Expansion card objects**

Object	Description
<a href="#">PDExpansionManager</a>	Detects the presence of a slot and gets information about a given slot. Gets the slot reference numbers that are used subsequently to gather card information.
<a href="#">PDExpansionCardInfo</a>	Provides information about an expansion card: media type, product name, manufacturer name, and so on.
<a href="#">PDVFSManager</a>	Represents a file system on an expansion card.
<a href="#">PDVFSVolumeManager</a>	Represents a volume in a file system on an expansion card. Use it to create/update files, query for the card size, usage, and so on, and create files and directories on the volume.
<a href="#">PDVFSFileManager</a>	Manages files and directories in a given volume. Each instance carries a volume reference number. Use it to open existing files and directories and to read and write them.

## COM ProgIDs

COM uses GUIDs to uniquely name objects, interfaces, type libraries, and just about anything else that needs a unique name. With automation technology, Microsoft made the decision to add user-friendly names.

These GUIDs are much easier to handle in Visual Basic and scripting languages. There are two registry entries added to the suite of entries used for component objects. These are the `ProgId` and the `VersionIndependentProgId`. They take the form `Application.Object.Version`. The `VersionIndependentProgId` loses the `Version` and becomes `Application.Object`.

Visual Basic automatically adds these names and references to the registry. It uses the project name for the `Application` part and concatenates the object name for the `Object` part. In C++, the ATL Object Wizard gives the developer the chance to input the `ProgID`. These are added to the registry entries in the object's `.rgs` file. ATL also takes care of registry entries.

Scripting languages generally use these names to bind to objects at run-time. The run-time scripting engine handles binding. Visual Basic can also bind to objects at run-time by using the `CreateObject` function. C++ programmers bind at run-time by calling `CLSIDFromProgID` to convert the `ProgId` to a GUID and then calling `CoCreateInstance` to get an instance of the object.

## COM Error Handling

This section discusses error handling in Visual Basic and C/C++.

### COM Error Handling in Visual Basic

COM methods and properties are required to return HRESULT error types. Optionally, on an object basis, they can use the rich error handling facilities, `ErrorInfo` objects. COM defines a protocol for returning and retrieving this information.

The Return protocol requires the object to implement the `ISupportErrorInfo` interface. Implementation of this interface indicates the object supports `ErrorInfo` objects. Object users must call the `ISupportErrorInfo::InterfaceSupportsErrorInfo` method for each interface the object implements to see if they can get `ErrorInfo` on that interface. `ErrorInfo` objects are available when HRESULT returns indicate errors.

The Retrieval protocol permits object users to determine if `ErrorInfo` objects are available. Calling the API function `GetErrorInfo` retrieves an `ErrorInfo` object. These `ErrorInfo` objects contain additional information about the error that occurred.

Visual Basic implements the Retrieval protocol and wraps it with the `Err` object. Visual Basic fills the `Err` object any time an error is returned from a call to a COM object interface method or property. Actions after this depend on the type and level of Visual Basic error handling in the function or subroutine that called the COM object.

A detailed description of COM error handling protocol and Visual Basic error handling can be found on the MSDN CDs or Microsoft web site.



## COM Error Handling in C/C++

[Listing 2.1](#) shows a sample of typical C++ code to catch errors and exceptions.

### Listing 2.1 Example of COM error handling in C++

---

```
{
PDDData baseQuery *pquery;
HRESULT hr;
hr = pquery->OpenRecordDatabase (pn, PDDirect.PRecordDatabase);
if (FAILED (hr)) {
    // Check to see if object supports exceptions.
    IsupportErrorInfo *pInfo = NULL;
    HRESULT hres = pQuery->QueryInterface (IID-ISupportErrorInfo, (void **) &
        pInfo);

    if (succeeded (hrRes) {
        hRes = pInfo-> Interface supports ErrorInfo (IID-IPDDatabaseQuery),if
            (succeeded (hRes)) {
                IErrorInfo *pErrorInfo = NULL;
                hRes = GetErrorInfo (0, pErrorInfo);
                if (hRes == S_OK){
                    BSTR bstrdesc;
                    hRes = pErrorInfo-> GetDescription(bstrDesc);
                }
            }
    }
}
```

---

## Database Support and Extensibility

### Database Support

Palm Powered™ handhelds support one or more of the following types of databases:

**[schema database](#)**: Highly structured, organized into tables that consist of rows and columns. Supported only in Palm OS Cobalt.

**[extended database](#)**: An “extended” version of a classic database. Supports records greater than 64 KB in length. Supported only in Palm OS Cobalt.

**[classic database](#)**: Minimally structured, organized into one or more records, each limited to 64 KB in length. Supported in Palm OS 1 and later. This is the type used by applications running on Palm OS Cobalt through PACE. The COM Sync module supports resources only in classic databases.

For more information on Palm OS databases, see [Chapter 8, “Palm OS Databases,”](#) on page 113 in the *Introduction to Conduit Development*.

The COM Sync module implements methods to create and open databases of each of these types. These methods return IUnknown pointers to COM objects created to manage access to the open databases. Rather than returning pointers to fixed objects defined by the COM Sync Suite, these methods return pointers to objects declared by their ProgIDs in the `Create` or `Open` parameters. These are instances of an object that controls the I/O for that database and are called **database adapters**.

The COM Sync module supplies a database adapter for each type of database. These adapters implement functionality covering the Sync Manager API. The adapter model is independent of the application database and is modeled after the structure of Palm OS® databases.

When a handheld database is opened or created, the COM Sync module creates an adapter:

- [PSDRowAdapter](#) for rows in a schema database
- [DmRecordAdapter](#) for records in an extended database
- [PDRecordAdapter](#) for records and [PDResourceAdapter](#) for resources in a classic database.

### **Extensibility**

You can create your own adapters to access a specific handheld database, such as one of the PIM applications' databases or your own handheld application's database. However, in the case of a schema database, such as those of the Palm OS Cobalt PIM applications, you do not need to. If you know the schema database name and creator ID, you can retrieve the schema of each table, which gives the structure of the database and the data type of each column. As an example, see the Address Book sample in <CDK>\COM\Samples\SchemaDB\AddressDB.

But if you need to create your own adapter, you can. For example, instead of using `PDRecordAdapter` to access the classic Address Book database, you can create a custom adapter to handle the data format specific to the Address Book's classic database and hide all the details from the user. In CDK versions 4.03 and later, the COM Sync module includes four such adapters, one for each of the classic databases used by the classic Palm OS PIM applications that are available on most handhelds running versions of Palm OS earlier than Palm OS Cobalt: Address Book, Date Book, To Do List, and Memo Pad. These **PIM database adapters** have names of the form `PD<PIM>DbHHRecordAdapter`, where <PIM> is either Address, DateBook, Memo, or Todo.

---

**NOTE:** The PIM database adapters provided in the COM Sync module access only the classic PIM databases. They do not work with the PIM databases in Palm OS Cobalt, which are schema databases.

---

Because the `Create` and `Open` methods take the `ProgID` of the adapter object as a parameter, you can create and use your own adapters. These adapters extend the underlying capabilities of the

## Development Basics

### *Database Support and Extensibility*

---

COM Sync adapters to application specific methods and properties. For example, a you can create an adapter that uses the `PDRecordAdapter` object for record I/O and expose methods and properties that in turn expose the structure of the database records that your Palm OS application uses.

The COM Sync module defines an interface, `IPDAInitialize`, and protocol that adapter developers must implement. Adapters use the COM Sync adapters as the basis of their implementation. They can expose the COM Sync adapter interface directly using COM aggregation or they can hide the COM Sync implementation and expose its functionality through delegation.

# Building a Conduit

---

In the following sections, this chapter describes how to build a COM-based conduit using the COM Sync Suite and Visual Basic:

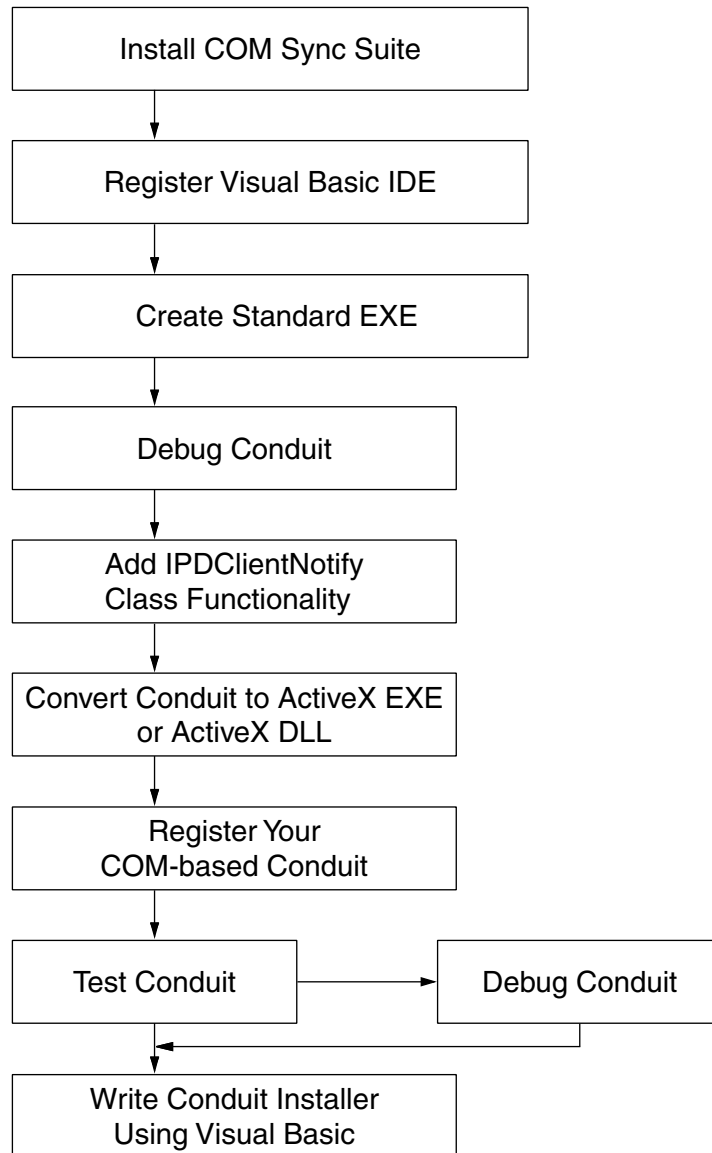
<a href="#">Requirements</a> . . . . .	33
<a href="#">Create the Debug Environment</a> . . . . .	33
<a href="#">Start a Visual Basic Project</a> . . . . .	36
<a href="#">Create the Conduit.</a> . . . . .	39
<a href="#">Debug the Conduit</a> . . . . .	43
<a href="#">Convert Your Conduit to an ActiveX Client</a> . . . . .	43
<a href="#">Implement the IPDClientNotify Interface</a> . . . . .	45
<a href="#">Developing a Conduit with Visual Basic .NET</a> . . . . .	48

## Building a Conduit

---

[Figure 3.1](#) shows the steps required to build a COM-based conduit.

**Figure 3.1 Building a COM-based conduit**



## Requirements

- HotSync Manager versions earlier than 6.0 require that every conduit have an application on the handheld with a unique creator ID that matches the creator ID of the conduit; otherwise HotSync® Manager will not run your conduit.

---

**NOTE:** HotSync Manager versions *6.0 and later* allow a conduit to “opt out” of the requirement that an application with the same creator ID exist on the handheld for the conduit to run. Creator IDs are still required for all conduits for unique identification, but depending on how a COM-based conduit responds when HotSync Manager calls its [`IPDClientNotify.GetConduitInfo\(\)`](#) entry point, HotSync Manager runs the conduit even without a corresponding application on the handheld. But if a conduit does not respond to this query, then the default behavior is that HotSync Manager always runs the conduit. Note that this behavior differs from the default behavior in versions earlier than 6.0.

---

- Every conduit must be registered with HotSync Manager. COM Sync objects that access the handheld are active *only* during a HotSync operation.
- The COM Sync module must be installed properly on every machine your conduit is to run on.

## Create the Debug Environment

The COM Sync module can execute a development environment as a client application. For example, you can configure the COM Sync module to launch Visual Basic during a HotSync operation, which allows you to program and debug with your handheld online. This enables you to debug your conduit in real-time without the handheld timing out.

---

**IMPORTANT:** COM Sync objects that access the handheld are active *only* during a HotSync operation. You cannot interactively debug your conduit in Visual Basic unless you follow the steps below and start a HotSync operation.

---

## Building a Conduit

### Create the Debug Environment

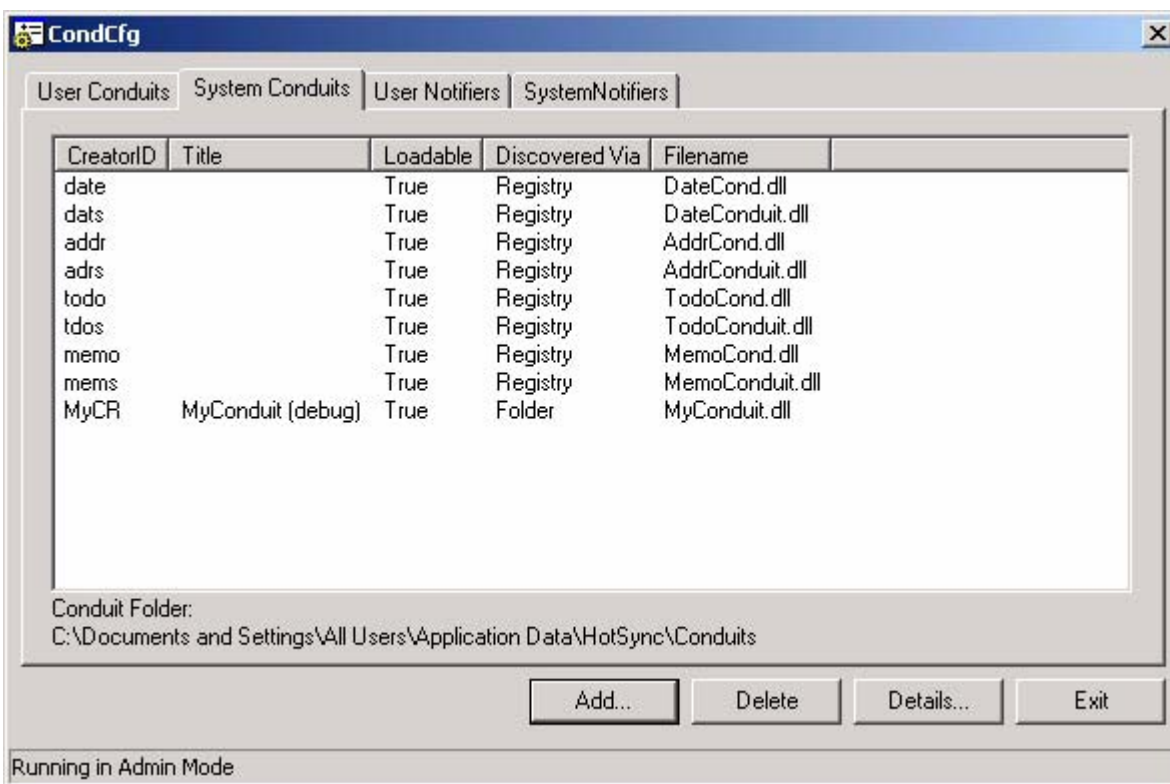
---

The first step to building a conduit is to create a debug environment. The following description shows how to do this with Visual Basic 6. Alternatively, see the Palm OS Knowledge Base article “[How do I debug a COM-based conduit DLL using Visual Basic .Net](#)” (article 359).

1. Start the Conduit Configuration utility (<CDK>\Common\Bin\CondCfg.exe) and click the **System Conduits** tab.

For a complete description of the CondCfg utility, see [Chapter 3, “Conduit Configuration Utility,”](#) on page 11 in the *Conduit Development Utilities Guide*.

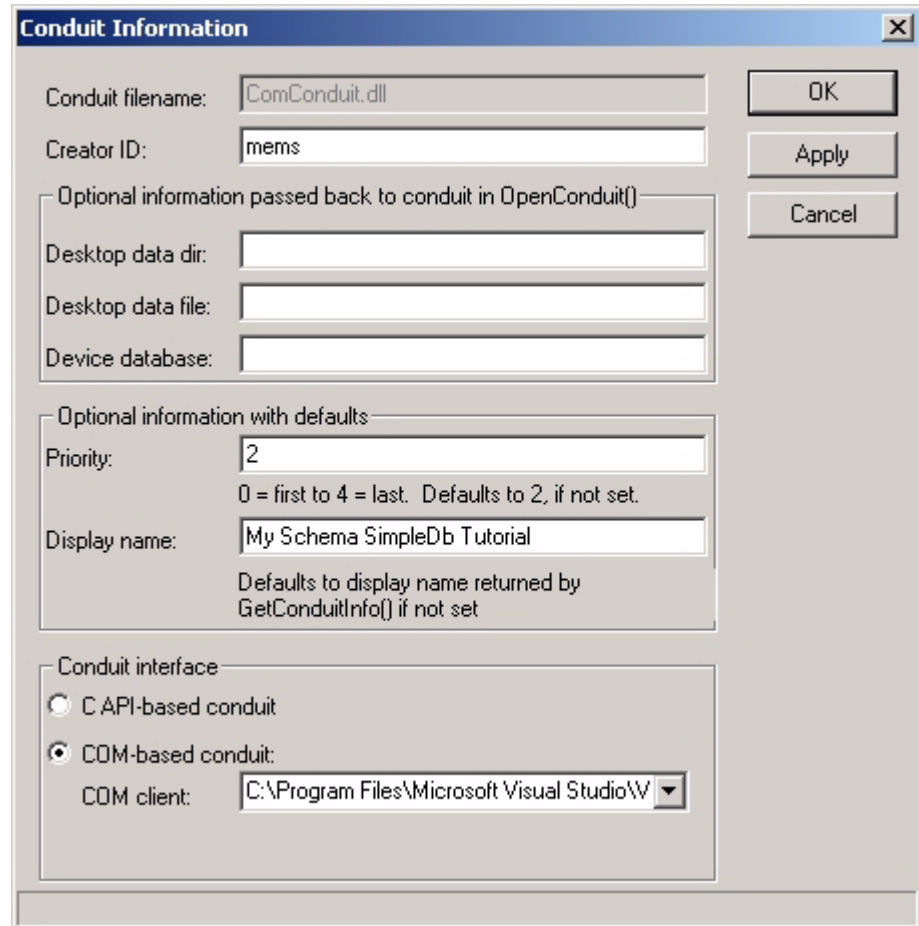
**Figure 3.2** Conduit Configuration utility, System Conduits tab



2. From the **System Conduits** tab, click the **Add** button.  
The **Conduit Information** dialog box displays.



**Figure 3.3** Conduit Information dialog box

The image shows a Windows-style dialog box titled "Conduit Information". It contains several input fields and sections. At the top, "Conduit filename:" is set to "ComConduit.dll" and "Creator ID:" is set to "mems". To the right are "OK", "Apply", and "Cancel" buttons. Below these is a section "Optional information passed back to conduit in OpenConduit()" with fields for "Desktop data dir:", "Desktop data file:", and "Device database:". Another section "Optional information with defaults" contains "Priority:" set to "2" (with a note "0 = first to 4 = last. Defaults to 2, if not set.") and "Display name:" set to "My Schema SimpleDb Tutorial" (with a note "Defaults to display name returned by GetConduitInfo() if not set"). The bottom section "Conduit interface" has two radio buttons: "C API-based conduit" (unselected) and "COM-based conduit:" (selected). Below the selected radio button is a "COM client:" dropdown menu showing "C:\Program Files\Microsoft Visual Studio\W".

3. Select the **COM-based conduit** option at the bottom of the dialog box shown in [Figure 3.3](#).
4. Enter the full path for the Visual Basic IDE in the **COM client** box.

For example, the full path when the COM Client is the VB 6 IDE could be: C:\Program Files\Microsoft Visual Studio\VB98\VB6.exe.

5. In **Creator ID**, type the creator ID of the application on the handheld whose data this conduit synchronizes. For example, to register the conduit in the <CDK>\COM\Samples\SchemaDB\Tutorial\SimpleDb sample, type mems. Note that if the standard Memo Pad conduit is still installed, CondCfg will not allow you to

## Building a Conduit

### Start a Visual Basic Project

---

register a second conduit with the same creator ID, so you must delete the existing conduit first.

6. Fill in any other relevant text boxes in CondCfg. [Chapter 3](#), “[Conduit Configuration Utility](#),” on page 11 in the *Conduit Development Utilities Guide* contains a complete description of all of these fields.

---

**IMPORTANT:** You must register your conduit with a unique creator ID on the desktop.

---

7. Click **OK** to add the new conduit information to the configuration entries.

Your debug environment is registered with HotSync Manager as if your IDE were a conduit. The next step is to use Visual Basic to build a simple client application and construct the client conduit as a standard EXE.

## Synchronizing with Palm OS Simulator

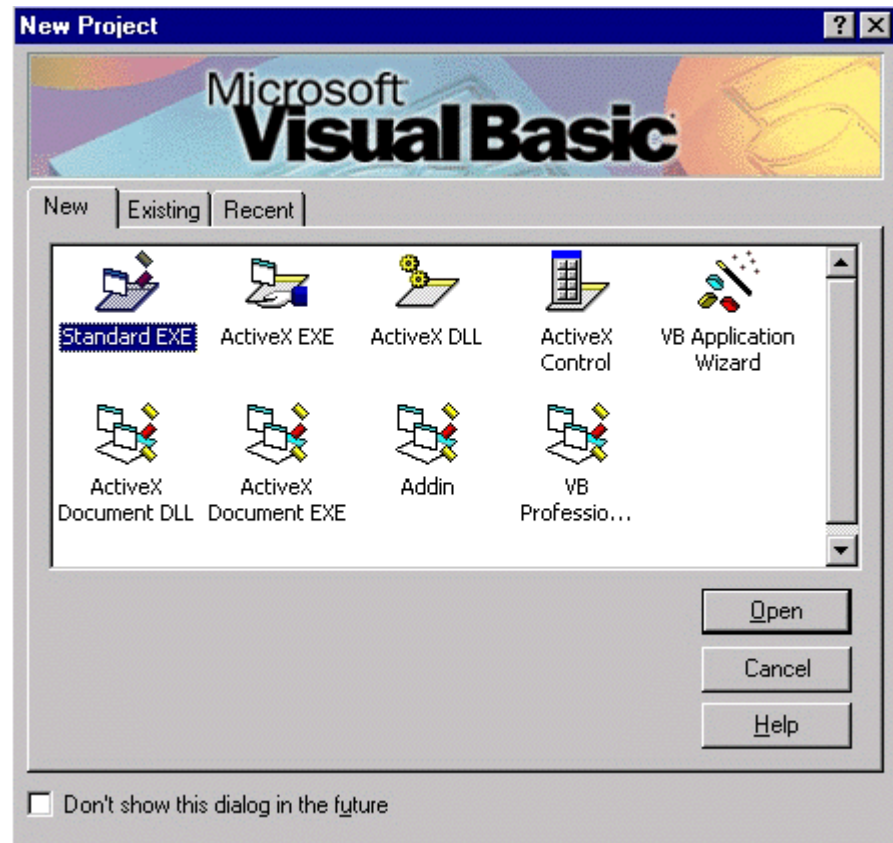
Though you will want to test your conduit with real handhelds, it is very useful during development to synchronize with Palm OS<sup>®</sup> Simulator instead of a real handheld. For details on how to get and set up Simulator, see [Chapter 8](#), “[Synchronizing with Palm OS Simulator](#),” on page 49 in the *Conduit Development Utilities Guide*.

## Start a Visual Basic Project

To start a new Visual Basic 6 project, follow these steps:

1. Start a new Visual Basic Project
2. To begin a new project, launch Visual Basic. The **New Project** dialog box in [Figure 3.4](#) appears.

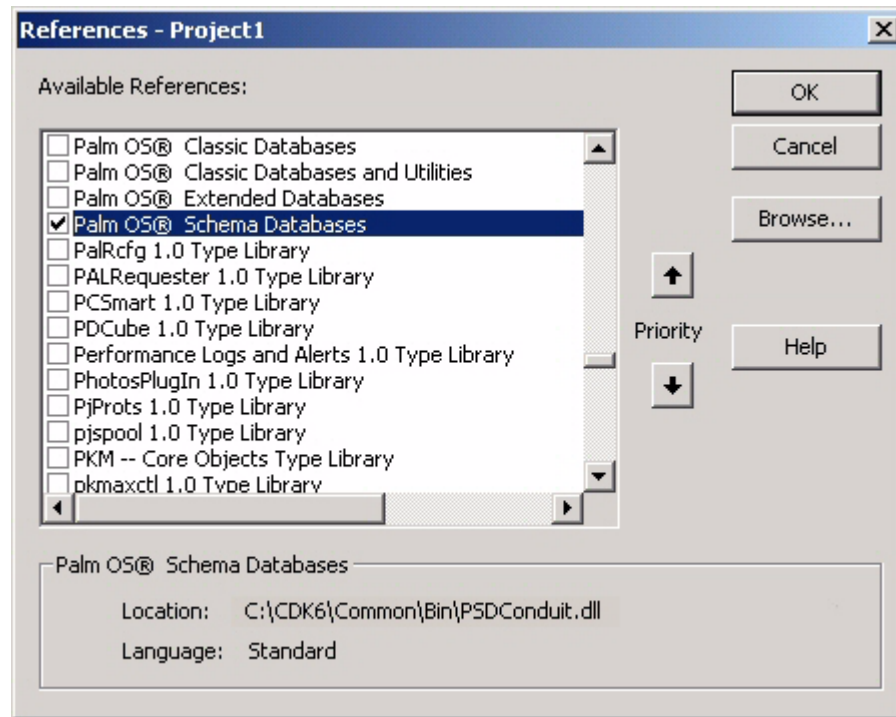
**Figure 3.4** Visual Basic New Project screen



3. Select **Project > References** from the main menu.
4. When the reference list appears, select one or more of the following COM Sync libraries that your project requires.
  - Palm OS® Classic Databases
  - Palm OS® Classic Databases and Utilities
  - Palm OS® Extended Databases
  - Palm OS® Schema Databases

[Figure 3.5](#) shows only the last selected, because the following section shows a sample that works only with schema databases.

**Figure 3.5** Select type library



A standard EXE client allows you to debug your conduit within your IDE. The COM Sync module activates standard EXE clients and waits for them to terminate.

These client modules may access the COM Sync objects using standard COM mechanisms. These clients are not required to access the COM Sync objects themselves and may spawn other processes/threads that access the COM Sync module.

Any EXE that includes a development environment, like Developer Studio or Visual Studio .NET, may be a COM Sync client. However, these client modules have no access to the HotSync Manager **Custom** menu.

For example, the COM Sync module can be configured to launch Microsoft Word. You can write a VBA macro that accesses the handheld through the COM Sync module, and run it within your Word document.

---

**IMPORTANT:** If you are using Visual Basic .NET, then you must put your conduit in the HotSync Manager executable directory. Otherwise, COM Sync will not be able to find the libraries it needs and will cause an error.

---

## Create the Conduit

These are generally the steps that any synchronization conduit must perform for a schema database:

1. You first create a [PSDDatabaseQuery](#) object, which contains all methods necessary to manage schema databases on the handheld.
2. Then you open a database to get a [PSDDatabaseAdapter](#) object, which represents a single open schema database.
3. Generally at this point, you can choose to perform the following three basic tasks in any order: manipulate rows, modify schemas and get/add/remove tables, and manipulate categories.

The remainder of this section discusses the sample, SimpleDb, which is located here:

<CDK>\COM\Samples\SchemaDB\Tutorial\SimpleDb\

This sample uses a text field to enter a search string and a button to perform the search for this string in the Memo Pad database, which is a schema database in Palm OS Cobalt. The text field object is txtSearch; the button object is btnSearch.

The following steps present the SimpleDb sample:

1. Start with the Click event subroutine.

The button Click event subroutine begins below. Note that the [PSDDatabaseQuery](#) object is created first, so that it can be used in the next step to open the Memo Pad schema database.

## Building a Conduit

### Create the Conduit

---

---

```
Private Sub btnSearch_Click()  
    ' Handle errors  
    On Error GoTo Handler  
  
    ' Declare the PSDDatabaseQuery, PSDRowAdapter, and other objects.  
    Dim pDbQuery As New PSDDatabaseQuery  
    Dim pMemoDbAdapter As PSDDatabaseAdapter  
    Dim pMemoSet As PSDRowSet  
    Dim pData As PSDRowData  
    Dim pRowAdapter As PSDRowAdapter  
    Dim TableNames As Variant  
  
    ' Let the user know we're processing  
    lblCount = "Please wait....."
```

---

2. Open the Memo Pad schema database and get a row adapter. Note that the [PSDDatabaseQuery](#) object requires that you open a schema database before your conduit can access it. The [OpenDatabase\(\)](#) method returns a [PSDDatabaseAdapter](#) object, which represents the open database and with which you can get a [PSDRowAdapter](#) object.

---

```
' Open the Memo Pad database.  
Set pMemoDbAdapter = pDbQuery.OpenDatabase("MemoDBS", "mems", _  
    ePSDReadWrite, EPSDShareNone)  
  
' We want to read all the rows in MemoPad, so first  
' get the row adapter.  
' Set pRowAdapter = pMemoDbAdapter.GetRowAdapter
```

---

3. Get the table and column names. The Memo Pad database has only one table with one column in it. Use the [PSDDatabaseAdapter](#) object to get this information.

---

```
' Now get the table name and column name.
' For Memo Pad, the table name is "MemoDBSchema" and there is only one
' column named "MemoText".
pMemoDbAdapter.GetTableNames TableNames

' Alternatively, we can use the following code to find
' the table name and column name.
Dim pTableInfo As PSDTable
Dim pColInfo As PSDColumnInfo
Set pTableInfo = pMemoDbAdapter.GetTableInfo(TableNames(0))
Dim ColNames As Variant
Dim ColCount As Long
ColCount = pTableInfo.GetColumnNames(ColNames)
```

---

4. Get a row set and iterate over the rows searching for the specified string. From the [PSDRowAdapter](#) object, get a [PSDRowSet](#) object and iterate over each row (a memo), searching for the matching string in individual rows represented by a [PSDRowData](#) object.

---

```
' Declare the Memo string.
Dim sMemo As String

' Declare the count of rows that contain the string.
Dim nCount As Long
Dim nTest As Variant

' Read the row set. ReadRows() returns a PSDRowSet object that can be
' used to iterate over the set of rows looking for data.
Set pMemoSet = pRowAdapter.ReadRows(TableNames(0), Null, eDbMatchAll)

' Start iterating over the row set looking for the specified string.
Set pData = pMemoSet.MoveFirst

' Loop and search each row for the string.
Do While Not pMemoSet.EOF
    sMemo = pData.Value("MemoText")

    ' Check whether the search string is in the Memo.
    nTest = InStr(sMemo, txtSearch.Text)
    If VarType(nTest) <> vbNull And nTest > 0 Then nCount = nCount + 1
```

---

## Building a Conduit

### Create the Conduit

---

```
' Read the next row.
Set pData = pMemoSet.MoveNext
Loop

' Prepare the result.
lblCount.Caption = Str(nCount) & " of " & Str(pMemoSet.GetRowCount) & _
" rows contained the string."
```

---

#### 5. Close the database and clean up.

---

**IMPORTANT:** Note that the [PSDDatabaseQuery](#) object requires that you call [CloseDatabase\(\)](#) to close a schema database before your conduit exits. A mirror-image synchronization conduit—which this sample is not—must also pass in `True` in the *bSeenAllChanges* parameter to reset change tracking for this schema database.

---

---

```
pDbQuery.CloseDatabase pMemoDbAdapter, ePSDNone, False
```

```
' Don't need the objects anymore
Set pDbQuery = Nothing
Set pMemoDbAdapter = Nothing
Set pRowAdapter = Nothing
Set pMemoSet = Nothing
```

```
' Normal exit
Exit Sub
```

#### Handler:

```
MsgBox "There was an error performing the search." & vbNewLine & _
Err.Description & vbNewLine & _
"Number = " & Hex(Err.Number)
```

```
pDbQuery.CloseDatabase pMemoDbAdapter, ePSDNone, False
```

```
' Don't need the objects anymore
Set pDbQuery = Nothing
Set pMemoDbAdapter = Nothing
Set pRowAdapter = Nothing
Set pMemoSet = Nothing
```

```
End Sub
```

---



## Debug the Conduit

After creating your debug environment in the previous section, you are now ready to debug your conduit.

1. Open the standard EXE conduit.
2. Set breakpoints, watches, `debug.print` statements, and so on, in the code.
3. Use the COM Sync module to debug your conduit and verify that it works.

The COM Sync module can launch an EXE or ActiveX DLL during the HotSync process, providing you real-time access to the handheld data during the debug process. As a developer, this gives you ultimate flexibility for your product.

---

**IMPORTANT:** You must have followed the steps in “[Create the Debug Environment](#)” on page 33 before you can debug your conduit in Visual Basic.

---

## Convert Your Conduit to an ActiveX Client

The ActiveX client allows you to use the [CfgConduit\(\)](#) method to configure your conduit from the HotSync Manager’s **Custom** menu and provides location independence through DCOM. In other words, you can access the conduit from anywhere on a network through DCOM.

The COM Sync module notifies you or activates an ActiveX client for one of two reasons:

- when a user selects the conduit from the HotSync Manager’s **Custom** menu
- when HotSync Manager is ready to run the client module

The COM Sync module uses standard COM mechanisms to point to your client object that implements the [IPDClientNotify](#) interface. You must build a public object that exposes the interface.

The COM Sync module calls one of two interface methods based on whether the request is coming from HotSync Manager or the

## Building a Conduit

### *Convert Your Conduit to an ActiveX Client*

---

**Custom** menu. ActiveX clients implement configuration and conduit logic in these two interface methods.

Some enterprise data servers run nonstop. If you want to synchronize handheld data with this type of an application, the conduit must notify the data server when it is ready to go. `IPDClientNotify` can be used for this.

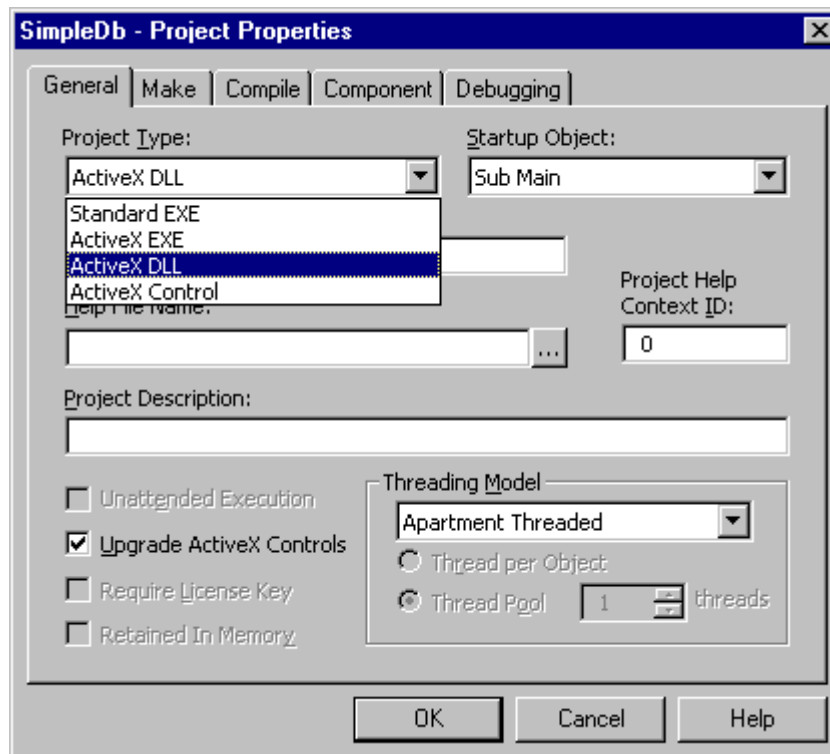
The conduit should be in the ActiveX format to allow HotSync Manager to see the `IPDClientNotify` interface.

There are two examples of the schema version of the SimpleDb sample in the <CDK>\COM\Samples\SchemaDB\Tutorial directory, `ActiveXExe` and `ActiveXDll`. The code is identical with the only difference being the Visual Basic project properties.

To convert SimpleDb to ActiveX, follow these steps:

1. Open the **Project Properties** dialog box and change the **Project Type** to **ActiveX DLL**.

**Figure 3.6 General Project Properties**



2. Set **Threading Model** to **Apartment Threaded**.

If you do not implement this setting, there are no restrictions.

## Implement the IPDClientNotify Interface

1. Add a class module that implements [IPDClientNotify](#).

The COM Sync Suite defines a notification interface, `IPDClientNotify`, which has four methods, including:

- [BeginProcess\(\)](#)
- [CfgConduit\(\)](#)
- [ConfigureConduit\(\)](#)
- [GetConduitInfo\(\)](#)

The first method is used to notify client modules that they can run their conduit logic. The second and third methods permit client conduits to display a dialog box when a user selects their conduit from the HotSync Manager's **Custom** menu. The last method provides HotSync Manager with information about your conduit.

Call your conduit code from the `BeginProcess` method and return `False` when you are done. Returning `False` tells the conduit that your process is done with the server and it should continue with the HotSync Manager process.

2. Set `Cnotify.cls`.

This class contains the [IPDClientNotify](#) interface's [BeginProcess\(\)](#) and other entry points that are not available in a standard EXE for "MultiUse" instancing.

If you want your conduit to be notified when HotSync Manager is ready for it to process, then you must implement this interface and expose it using a single object.

The COM Sync module includes the `IPDClientNotify` interface in its type library. If you choose to implement the interface, Visual Basic developers create a class that uses the "Implements" keyword and make the project either an ActiveX EXE or ActiveX DLL.

The SimpleDb tutorial contains samples for both `ActiveXExe` and `ActiveXDll`. The `IPDClientNotify` interface is not

## Building a Conduit

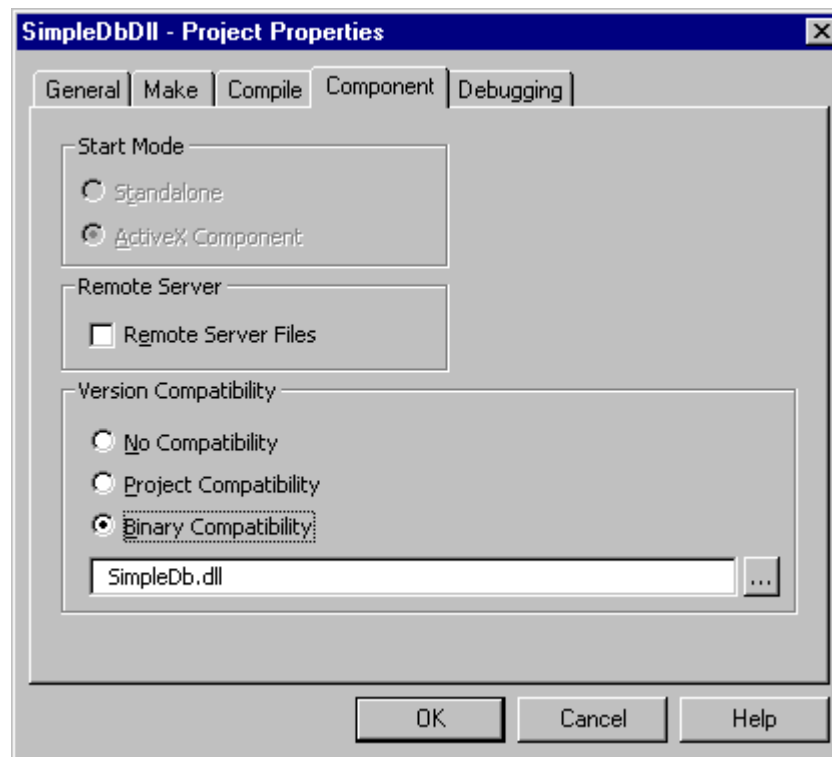
### *Implement the IPDClientNotify Interface*

---

required and is used only in conjunction with ActiveX client module types.

3. Select **File > Make Project** to compile a DLL.  
This compiles a DLL and links your application.
4. In **Project Properties**, set the component's **Version Compatibility** to **Binary Compatibility**.

**Figure 3.7 Component Project Properties**



"Binary Compatibility" ensures the GUID of the `IPDClientNotify` object remains constant. (For help on the difference between "Binary Compatibility" and "Project Compatibility," see the Visual Basic help files.)

5. Launch `CondCfg.exe`.

**Figure 3.8** Registering your COM client using CondCfg

Conduit Information

Conduit filename: ComConduit.dll

Creator ID: mems

Optional information passed back to conduit in OpenConduit()

Desktop data dir:

Desktop data file:

Device database:

Optional information with defaults

Priority: 2  
0 = first to 4 = last. Defaults to 2, if not set.

Display name: My Schema SimpleDb Tutorial  
Defaults to display name returned by GetConduitInfo() if not set

Conduit interface

☐ C API-based conduit

☒ COM-based conduit:

COM client: simpledbdll.CNotifyDll

OK

Apply

Cancel

6. Click the **COM conduit** option button and fill in the COM Client box with the ProgID of your module (see [Figure 3.8](#)).  
For example, the SimpleDb ActiveX DLL sample has a ProgID of simpledbdll.CNotifyDll.  
For standalone EXEs, type the EXE name with its full path in **COM client**.
7. In **Creator ID**, type the creator ID of the application on the handheld whose data this conduit synchronizes. For example, to register the conduit in the <CDK>\COM\Samples\SchemaDB\Tutorial\SimpleDb sample, type mems. Note that if the standard Memo Pad conduit is still installed, CondCfg will not allow you to

## Building a Conduit

*Developing a Conduit with Visual Basic .NET*

---

register a second conduit with the same creator ID, so you must delete the existing conduit first.

8. Fill in any other relevant text boxes in CondCfg. [Chapter 3, “Conduit Configuration Utility,”](#) on page 11 in the *Conduit Development Utilities Guide* contains a complete description of all of these fields.
9. When you write your own installer, use the [PDCondMgr](#) (or [PDSystemCondMgr](#)) and [PDConduitInfo](#) objects, setting the [FileName](#) property to `COMConduit.dll` and the [COMClassID](#) property to your conduit's ProgID.

Once you write, debug, and convert your conduit to the required form, you write a setup/installation routine to deploy the conduit. This is typically done with InstallShield or other installation software packages. For more information, see [Chapter 4, “Writing an Installer,”](#) on page 51.

---

**IMPORTANT:** If you are using Visual Basic .NET to develop your conduit, then you must deploy your conduit in the end user's HotSync Manager executable directory. Otherwise, COM Sync will not be able to find the libraries it needs and will cause an error.

---

## Developing a Conduit with Visual Basic .NET

Conduits written or compiled under Visual Basic .NET produce an **assembly**. HotSync Manager can load these assemblies using COM interoperability. Because COM Sync components still use the underlying COM architecture, you must add references to COM Sync components in your project. To do so in VB .NET, click **Project > Add Resources > COM** and select one or more of the following COM Sync libraries that your project requires.

- Palm OS® Classic Databases
- Palm OS® Classic Databases and Utilities
- Palm OS® Extended Databases
- Palm OS® Schema Databases

Remember that to enable the user to configure your conduit from HotSync Manager's **Custom** dialog box, you must implement the [IPDClientNotify](#) interface in your conduit (see "[Implement the IPDClientNotify Interface](#)" on page 45). In the [PDConduitInfo.COMClassID](#) property for conduit registration, place the classID of the class that implements this interface. If your conduit is a standard EXE and does not implement this interface, set the [COMClassID](#) property to the full path and filename of the EXE.

---

**NOTE:** If you use VB .NET to upgrade a VB 6 ActiveX conduit (like one of the samples in the CDK) to VB .NET 2003, then the VB .NET upgrade wizard inserts a line of code at the beginning of the notification class that looks like this:

```
<System.Runtime.InteropServices.ProgId  
("CNotify_NET.CNotify")> Public Class CNotify
```

The upgrade wizard does this to avoid using the same ProgID as the conduit built with VB 6.

If you retain the code between the <> symbols, then when you register your conduit, you must specify `CNotify_NET.CNotify` in the `ComClient` field. However, if you remove this code, then specifying `SimpleDB.CNotify` in `COMClient` works.

---

## Registration Steps

These steps point out what you need to do differently to get HotSync Manager to run a conduit you build with VB .NET:

1. For HotSync Manager to recognize VB .NET assemblies, they must be in the same directory as HotSync Manager. To do this, you can either copy your conduit to the HotSync Manager directory or change your project's output directory to the HotSync Manager directory (in VB .NET, do this in the project's **Property Pages**).
2. Run `Regasm.exe` on your conduit so that the component details are placed in the registry. (`Regasm.exe` is under `<Windows>\Microsoft.Net\Framework` if you have installed the .NET framework on your computer.)

## **Building a Conduit**

*Developing a Conduit with Visual Basic .NET*

---

Then as with VB, register your conduit with HotSync Manager as described in “[Create the Debug Environment](#)” on page 33 (or in “[Writing an Installer](#)” on page 51 if you’re writing an installer).



# Writing an Installer

---

Creating an easy-to-use installer is an important part of product development. You need to make it easy for end users to install your conduit and your desktop and handheld applications. This chapter describes what your installer needs to do and the components that the COM Sync Suite provides to help you create an installer.

---

**NOTE:** This chapter describes conduit installation on desktops running HotSync Manager versions 6.0 or later, which include the COM Sync module; earlier versions do not. If your conduit must work with earlier versions of HotSync Manager, use CDK version 4.03 and refer to its documentation.

---

The sections in this chapter are:

<a href="#">Installer Tasks</a>	. . . . . 52
<a href="#">Files to Install</a>	. . . . . 53
<a href="#">Sample for Registering a COM-based Conduit</a>	. . . . . 55
<a href="#">Installing Files on the Handheld with PDInstall</a>	. . . . . 58
<a href="#">Controlling HotSync Manager with PDHotSyncUtility</a>	. . . . . 63
<a href="#">Accessing User Data with PDUserData</a>	. . . . . 64
<a href="#">Installing an Install Conduit with PDInstallConduit</a>	. . . . . 65
<a href="#">Installing a Notifier with PDCondMgr</a>	. . . . . 67
<a href="#">Uninstalling Your Conduit</a>	. . . . . 67
<a href="#">Testing Your Installer</a>	. . . . . 68
<a href="#">Installation Troubleshooting Tips</a>	. . . . . 69

## Installer Tasks

Your installer may need to install a handheld application and databases, a desktop application, as well as a conduit. So the best user experience is to write a single installer to install all of these at once. For handheld databases, your installer should display a user list if there is more than one HotSync® user and prepare the databases to be installed on the next HotSync operation. Your installer must register your conduit, otherwise HotSync Manager cannot run it. And finally, it must request that the user perform a HotSync operation to actually install your application and databases on the handheld.

To summarize, your installer must perform at least some of the following tasks to successfully deploy your product on an end user's desktop computer and handheld:

- If your conduit must work with HotSync Manager versions earlier than 6.0, then install the COM Sync libraries as described in "[Files to Install](#)" on page 53. If it must work only with HotSync Manager 6.0 or later, do not install these libraries.
- Copy all of your install files to the end user's desktop computer.
- Install your desktop application, if your product includes one.
- If necessary, ensure your required files are installed with your desktop application or conduit and, if any are ActiveX controls, that they are registered with Windows using `regsvr32.exe`.
- Copy your conduit files (and notifier DLL, if you provide one) to their final destination directory.
- Register your conduit with HotSync Manager by using the Conduit Manager, [PDCondMgr.RegisterConduit\(\)](#) (required for any user conduit), or [PDSystemCondMgr.RegisterConduit\(\)](#) (required for a system conduit).
- Register your notifier with HotSync Manager by calling [PDCondMgr.RegisterNotifier\(\)](#) (required only if you are providing a notifier).

- Restart or refresh HotSync Manager (after you register your conduit) by using [PDHotSyncUtility.RestartHotSyncMgr\(\)](#) or [RefreshConduitInfo\(\)](#). Required only for a conduit registered with HotSync Manager versions earlier than 6.0. Versions 6.0 and later do not require this.
- Queue databases or applications to be installed on the handheld (or files to be installed on handheld expansion cards) during the next HotSync operation by using [PDInstall.InstallFileToHH\(\)](#) or [InstallFileToSlot\(\)](#) (required for any databases or files to be put on the handheld).
- Uninstall your desktop application, conduit, and notifier (if you provide one), which includes unregistering your conduit (and notifier) using [PDCondMgr.UnregisterConduit\(\)](#) and [UnregisterNotifier\(\)](#) for user conduits, and [PDSysCondMgr.UnregisterConduit\(\)](#) for system conduits.

## Files to Install

Typically, the files you need to install on end users' machines include:

- `MyApp.prc` and supporting databases, if any (your handheld application)
- `MyConduit.dll` (your conduit)
- `MyApp.exe` and supporting files (your desktop application, if your product includes one)

---

**IMPORTANT:** If you are using Visual Basic .NET to develop your conduit, then you must deploy your conduit in the end user's HotSync Manager executable directory. Otherwise, COM Sync will not be able to find the libraries it needs and will cause an error.

---

Whether you must install additional files depends on the version of HotSync Manager on your end users' machines, as described below.

#### **HotSync Manager Versions 6.0 and Later**

HotSync Manager versions 6.0 and later include the COM Sync libraries, so you need only to install your product's files. Everything else is already present.

#### **HotSync Manager Versions Earlier than 6.0**

Versions of HotSync Manager earlier than 6.0 do not include the COM Sync libraries. Therefore, if your conduit must work with these earlier versions, then you must install and register the COM Sync libraries on the end user's system.

The following procedure describes what your installer must do:

1. Find the path of the HotSync Manager binaries as described in "[Finding the HotSync Manager Binaries](#)" on page 101 in *C/C++ Sync Suite Companion*. Note that this step requires that you install a temporary copy of `CondMgr.dll` and make a C API call to find the HotSync Manager executable folder.
2. If the version of HotSync Manager is 6.0 or later, then stop here because the COM Sync libraries are already present; otherwise, continue to the next step.
3. Copy the following COM Sync DLLs to the HotSync Manager executable folder that you discovered in step 1:
  - `ComConduit.dll`
  - `ComDirect.dll`
  - `ComStandard.dll`

PalmSource, Inc. permits you to redistribute and install these three files. You can get them from the following CDK folder:

`<CDK>\Common\Bin`

Unless otherwise specified, PalmSource does not permit you to redistribute any other binary files from the CDK.

4. Run `regsvr32.exe` in silent mode (`runsvr32.exe /s`) to register the COM objects contained in `ComConduit.dll`, `ComDirect.dll` and `ComStandard.dll`. Silent mode does not display any messages to the user during installation.
5. Continue installing your own conduit, application, or other files. The COM Sync libraries are now available for you to use.

## Sample for Registering a COM-based Conduit

The COM Sync Suite includes the [PDCondMgr](#) (and [PDSystemCondMgr](#)) and [PDConduitInfo](#) objects that you can use to register and unregister a COM-based conduit. The sample in [Listing 4.1](#) registers and unregisters the COM-based conduit you specify by passing in a creator ID string.

---

**NOTE:** [PDCondMgr](#) can register conduits only for the current Windows user, not for the system (all users); you must use [PDSystemCondMgr](#) to register conduits for the system. For details, see “[User- and System-registered Conduits and Notifiers](#)” on page 78 in the *Introduction to Conduit Development*.

---

The following steps outline the behavior of the sample in [Listing 4.1](#). To try this sample, create a new Visual Basic project and add the following COM library as a reference.

Palm OS® Classic Databases and Utilities

1. To get the long value of the creator ID, use [PDCondMgr.StringToCreatorID\(\)](#) (or [PDSystemCondMgr.StringToCreatorID\(\)](#)) and check whether a conduit with that creator ID already exists. If it does, this sample prompts whether you want to unregister it.
2. To specify the conduit configuration entries, create a [PDConduitInfo](#) object and set its properties. Note that in this sample the [COMClassID](#) property is set to the Windows Calculator program just so that this sample will run. You should replace this with the path of your IDE executable while you’re debugging your conduit, any EXE you want to run, or the class ID of your COM-based conduit if it implements the [IPDClientNotify](#) interface (see “[Convert Your Conduit to an ActiveX Client](#)” on page 43).
3. To register the conduit, pass the [PDConduitInfo](#) object to [PDCondMgr.RegisterConduit\(\)](#) or [PDSystemCondMgr.RegisterConduit\(\)](#).
4. To confirm the conduit is registered, pass [PDCondMgr.GetConduitInfo\(\)](#) (or [PDSystemCondMgr.GetConduitInfo\(\)](#)) the same creator ID and view the retrieved information.

## Writing an Installer

### *Sample for Registering a COM-based Conduit*

---

#### **Listing 4.1 Register a conduit using COM Sync objects**

---

```
Private Function RegisterConduit(strConduitCreatorID As String) As Boolean

    Dim CreatorID As Long
    Dim PDCondMgr As New PDCondMgr
    Dim PConduitInfo As New PDConduitInfo
    Dim RetrievePDConduitInfo As New PDConduitInfo
    Dim CreatorIDExists As Boolean

    On Error GoTo ErrorHandler

    CreatorIDExists = True
    CreatorID = PDCondMgr.StringToCreatorID(strConduitCreatorID)
    ' Make sure a valid CreatorID could be retrieved from the string.
    If CreatorID = 0 Then
        MsgBox "CreatorID '" & strConduitCreatorID & "' was invalid.", _
            vbCritical, "Invalid CreatorID"
        Exit Function
    Else
        Set RetrievePDConduitInfo = PDCondMgr.GetConduitInfo(CreatorID)
    End If

    ' Check whether a conduit with the specified CreatorID currently exists.
    If CreatorIDExists Then
        If MsgBox("A conduit with CreatorID '" & strConduitCreatorID & _
            "' already exists." & " Do you want to remove it ?", vbYesNo + _
            vbQuestion, "Remove Conduit") = vbYes Then
            Call PDCondMgr.UnregisterConduit(CreatorID)
        Else
            Exit Function
        End If
    End If

    Set RetrievePDConduitInfo = Nothing

    ' Set the conduit entries
    With PConduitInfo
        .COMClassID = "c:\winnt\system32\calc.exe" ' Change to your IDE or
                                                ' class ID
        .CreatorID = CreatorID
        .DeskTopDataDirectory = "DeskTopDataDirectory"
        .HandHeldDB = "HandHeldDB"
        .DeskTopDataFile = "DeskTopDataFile"
        .DisplayName = "DisplayName"
        .Priority = 2
    End With
```

```
Call PDCondMgr.RegisterConduit(PConduitInfo)

' Sample to retrieve the conduit info and display one of the entries.
Set RetrievePDConduitInfo = PDCondMgr.GetConduitInfo(CreatorID)
MsgBox "COM Conduit '" & TitleRetrievePDConduitInfo.DisplayName & "' was _
    successfully registered.", vbInformation, "Information"

RegisterConduit = True
Exit Function

ErrorHandler:
' The specified CreatorId is not valid or not found
If Err.Number - vbObjectError = 8223 Then
    Err.Clear
    CreatorIDExists = False
    Resume Next
Else
    MsgBox "Conduit registration failed." & vbCr & "Error Detail : " _
        & Err.Description, vbCritical, "Conduit Registration Error : " _
        & Err.Number
End If
Exit Function
End Function
```

---

For background information on conduit registration, see [Chapter 6, “Registering Conduits and Notifiers with HotSync Manager,”](#) on page 73 in *Introduction to Conduit Development*.

---

**NOTE:** HotSync Manager does not support folder-based conduit registration for COM-based conduits—that is, you cannot register a COM-based conduit by copying it to the `Conduits` folder. You must use `PDCondMgr` or `PDSysCondMgr` to register a COM-based conduit.

---

## Installing Files on the Handheld with PDInstall

The [PDInstall](#) object does not actually move data between the desktop and handheld; it simply moves files into a queue directory on the desktop and tells HotSync Manager that something needs to be installed. A registered [install conduit](#) called by HotSync Manager during a HotSync operation actually transfers queued files to primary storage or to an expansion card in a handheld. HotSync Manager ships with several install conduits, so you likely do not need to implement your own install conduit. “[Running Install Conduits](#)” on page 105 in the *Introduction to Conduit Development* lists these install conduits and the file types they can install.

---

**NOTE:** An alternative to using PDInstall is the [PSDDatabaseQuery.InstallDatabase\(\)](#) method. You can call this method only during a HotSync operation; instead of queueing a file for installation later, this method actually transfers the file to the handheld’s primary storage when you call it.

---

The rest of this section covers the following topics:

- [How PDInstall Works](#)
- [Install Directory Terminology](#)
- [Expansion Slot Support in PDInstall](#)

For details on the PDInstall object, see “[PDInstall](#)” on page 56 in the *COM Sync Suite Reference*.

### How PDInstall Works

The [PDInstall](#) object provides two methods that your installer, desktop application, or conduit can call to queue files for installation: [InstallFileToHH\(\)](#) for files destined for primary storage and [InstallFileToSlot\(\)](#) for files destined for an expansion card. When you call one of these methods, you specify the HotSync user and the path of the file you want to install. At the time you call one of these methods, PDInstall performs the following tasks:

1. PDInstall selects a default install conduit to run based on the *extension* of the file you specify and the *install method* you



call, as shown in [Table 4.1](#). Each install conduit is registered to handle certain filename extensions.

**Table 4.1 PDInstall's choice of install conduit and destination based on filename extension**

Filename Extension	Method Called	Install Conduit used by PDInstall	Destination on Handheld
PRC, PDB, SDB, SSD, PQA	<a href="#">InstallFileToHH()</a>	Install	Primary storage
PNC, SCP	<a href="#">InstallFileToHH()</a>	Install Service Templates <sup>1</sup>	Primary storage
*.*	<a href="#">InstallFileToHH()</a>	HotSync Exchange	Primary storage
*.*	<a href="#">InstallFileToSlot()</a>	Install to Card	Card in specified expansion slot

1. Works only with handhelds running a version of Palm OS earlier than Palm OS Cobalt.

PDInstall chooses the HotSync Exchange conduit if the filename extension does not match one of the preceding Palm OS filename extensions but does match an extension that a handheld application has registered with the Exchange Manager on the handheld. See [Chapter 3, "Using HotSync Exchange,"](#) on page 27 in the *Introduction to Conduit Development*.

- PDInstall copies the file to the install directory that is associated with the install conduit that it selected in step 1.
- PDInstall creates an install flag for the specified HotSync user in the configuration entries.
- PDInstall sets the value of the install flag to the value of the bit mask associated with the install conduit that is registered to handle files of the specified type. HotSync Manager reads this install flag the next time it runs install conduits for the specified user. This flag tells HotSync Manager that it must run the install conduit with the matching mask value.

## Writing an Installer

### *Installing Files on the Handheld with PDInstall*

---

If you call one of the install methods again before HotSync Manager runs the install conduits again, these steps repeat—except that the `PDInstall` ORs the existing value of the install flag with that of each subsequent call. For example, if the first call selects an install conduit with a bit mask value of 1 and the second call selects one with a value of 4, the resulting value of this install flag is 5 the next time the install conduits are run. Therefore HotSync Manager will run two install conduits, one registered with a bit mask of 1 and another with a bit mask of 4.

Each install conduit decides whether to remove its mask value from the HotSync user's install flag after a successful or unsuccessful installation. The Install install conduit leaves the mask value if unsuccessful; it removes it if successful. The HotSync Exchange install conduit runs independent of the mask, because there could be files that need to be transferred from the handheld to the desktop.

HotSync Manager calls install conduits twice: before it runs all synchronization conduits and again after. If one of the install methods is called outside of a HotSync operation, then an install conduit installs the queued file at the *beginning* of the *next* HotSync operation. If a conduit calls one of the install methods during a HotSync operation, then an install conduit installs the queued file at the *end* of the *current* HotSync operation.

The `PDInstall` object provides other methods to help manage files queued to be installed. See "[PDInstall](#)" on page 56 in the *COM Sync Suite Reference* for details.

## Install Directory Terminology

[Table 4.2](#) defines the various desktop directories to which the `PDInstall` object copies files queued to be installed by the default install conduits during the next HotSync operation. [Figure 4.1](#) shows the relative locations of these directories for the current Windows user.

**Table 4.2 Desktop directories associated with PDInstall**

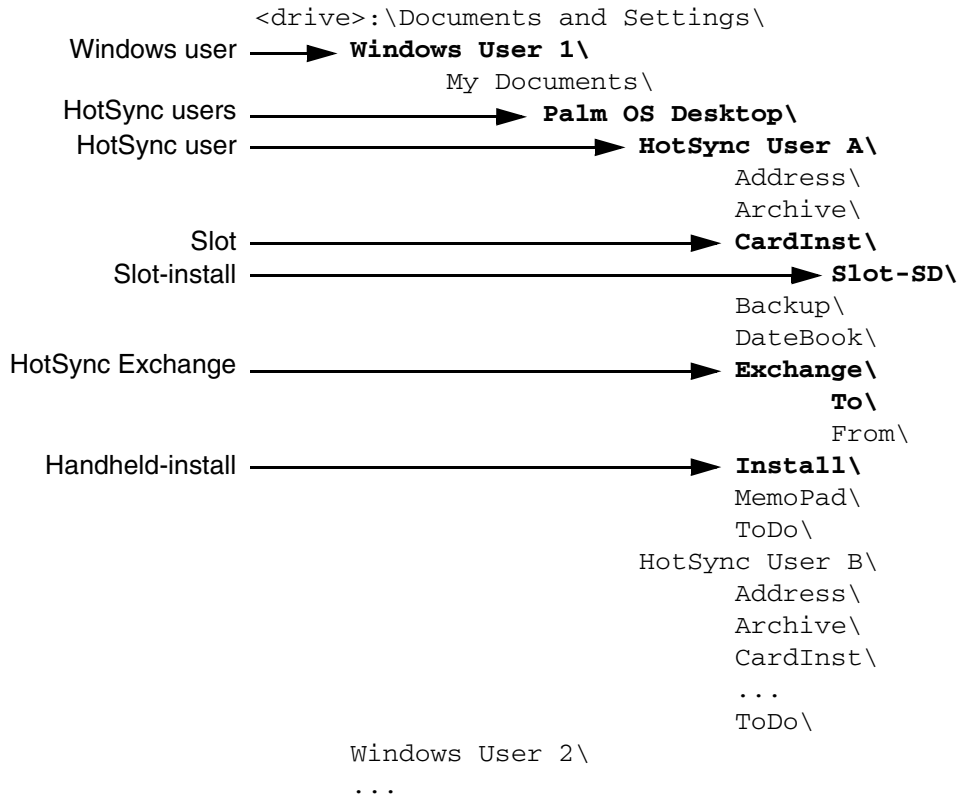
Directory Name	Description
Windows user's directory	For the current Windows user, this directory contains all HotSync Manager files and user directories (by default) for all HotSync users. To get this path, call <a href="#"><code>PDUserData.GetRootDirectory()</code></a> .
HotSync user's directory	For each HotSync user, this directory contains the handheld-install and slot directories, as well as data directories for each default conduit.
Handheld-install directory	For each HotSync user, this directory contains Palm OS applications and databases queued to be installed by an associated install conduit into the handheld's main memory the next time install conduits run. There can be such a directory for each of multiple install conduits.
Slot directory	For each HotSync user, this directory contains subdirectories (slot-install directories) for each slot on the user's handheld. This directory and its subdirectories are associated with an install conduit that installs files to expansion cards.
Slot-install directory	For each slot on each HotSync user's handheld, this directory contains files queued to be installed to a card in the associated slot the next time install conduits run.
HotSync Exchange directory	For each HotSync user, the Exchange directory contains To and From subdirectories. The To directory contains files to be installed to the handheld's main memory via HotSync Exchange. The From directory contains files that have been sent to the desktop from the handheld.

## Writing an Installer

*Installing Files on the Handheld with PDInstall*

---

**Figure 4.1** Directories that default install conduits queue files in



## Expansion Slot Support in PDInstall

Several [PDInstall](#) methods support copying files from the desktop to cards in handheld expansion slots. The most commonly used PDInstall methods are:

- [InstallFileToSlot\(\)](#)—queues files to copy to an expansion card during the next HotSync operation.
- [ChangeFileDestinationSlotToHH\(\)](#), [ChangeFileDestinationHHToSlot\(\)](#), and [ChangeFileSlotDestination\(\)](#)—move files on the desktop between slot-install directories and between slot-install directories and handheld-install directories

- [GetSlotFileCount\(\)](#), [GetSlotFileSize\(\)](#)—get information about files already queued to be installed on expansion cards in a specific user's handheld

Your desktop application or installer calls `PDInstall` methods to queue files for installation. During the next HotSync operation, HotSync Manager calls the install conduit registered to handle the type of files you queued for installation. The install conduit copies the files during the HotSync operation.

HotSync Manager retrieves information about a handheld's expansion slots (or lack thereof) at the beginning of each HotSync operation and saves it for the corresponding user in the user information store on the desktop. However, between HotSync operations, the user can change or update the handheld—for example, the user could upgrade to a handheld with more slots or switch to one with no slots at all. Upon the next HotSync operation, HotSync Manager asks the user either to create a new user name or to choose one from the list of users who have previously performed a HotSync operation on this desktop computer. If the user chooses a user name from the list, the slot information saved for that user at the beginning of the last HotSync operation is now inaccurate.

Because your application or installer calls `PDInstall` methods outside of a HotSync operation, be aware that your files might not be installed during the next HotSync operation if the slot or card does not exist at that time.

## Controlling HotSync Manager with PDHotSyncUtility

Use the [PDHotSyncUtility](#) object to enable your application to control HotSync Manager in the following ways:

- [TerminateHotSyncMgr\(\)](#), [StartHotSyncMgr\(\)](#), [RestartHotSyncMgr\(\)](#), and [RefreshConduitInfo\(\)](#)—stop, start, restart, or refresh HotSync Manager (useful after registering a conduit).
- [LaunchCustomDlg\(\)](#), [LaunchSetupDlg\(\)](#), and [DisplayLog\(\)](#)—call HotSync Manager dialogs to customize conduits or change connection settings; or to display the HotSync Log.

## Writing an Installer

### *Accessing User Data with PDUserData*

---

- [GetCommStatus\(\)](#) and [SetCommStatus\(\)](#)—check or set the connection status type for serial, modem, and network connections.
- [IsSyncInProgress\(\)](#)—checks whether HotSync Manager is idle or synchronizing.

For details on the PDHotSyncUtility methods, see “[PDHotSyncUtility](#)” on page 54 in the *COM Sync Suite Reference*.

## Accessing User Data with PDUserData

The [PDUserData](#) object is the preferred way to access information in the users data store on the desktop. The **users data store** holds name, synchronization preferences, directory, and password information about users who have synchronized Palm Powered™ handhelds with the desktop computer.

For details on the PDUserData methods, see “[PDUserData](#)” on page 82 in the *COM Sync Suite Reference*.

## Installing an Install Conduit with PDInstallConduit

The [PDInstallConduit](#) object registers or unregisters install conduit with HotSync Manager. It provides most of the same functionality for registering install conduits as the [PDCondMgr](#) (or [PDSystemCondMgr](#)) object provides for registering standard synchronization conduits. An **install conduit** is a special type of conduit that installs databases or applications on a handheld (or copies files from the desktop to handheld expansion cards). For more information on how the [PDInstall](#) object uses install conduits, see “[How PDInstall Works](#)” on page 58.

Most conduits are synchronization conduits, not install conduits, and should be registered with the [PDCondMgr](#) (or [PDSystemCondMgr](#)) object instead (see “[Sample for Registering a COM-based Conduit](#)” on page 55).

---

**NOTE:** The `PDInstallCondMgr` can register install conduits only for the current Windows user, not for the system (all users). For details, see “[User- and System-registered Conduits and Notifiers](#)” on page 78 in the *Introduction to Conduit Development*.”

---

This section discusses the following topics:

- [Registering an Install Conduit](#)
- [Unregistering an Install Conduit](#)
- [Adding or Modifying Install Conduit Configuration Entries](#)

## Writing an Installer

### *Installing an Install Conduit with PDInstallConduit*

---

## Registering an Install Conduit

When you register an install conduit with HotSync Manager, you must create and set the properties of a [PDInstallConduitInfo](#) object. These properties specify a unique bit mask value ([Mask](#)) and a unique ID ([UniqueId](#)) associated with your install conduit. HotSync Manager uses only the mask value to identify your install conduit, but the unique ID is also required to register it. You use the [PDInstallConduit](#) object to add these and other values associated with your install conduit in the conduit configuration entries. PalmSource, Inc. recommends that you check the [Mask](#) and [UniqueId](#) values of install conduits that are already registered before registering your install conduit.

To register your install conduit, call the [RegisterIC\(\)](#) method. This method adds all of the properties you defined in the associated [PDInstallConduitInfo](#) object to the install conduit configuration entries on the desktop.

For details on the `PDInstallConduit` methods, see “[PDInstallConduit](#)” on page 58 in the *COM Sync Suite Reference*.

## Unregistering an Install Conduit

The `PDInstallConduit` object also provides a method for unregistering an install conduit. Simply call the [UnregisterIC\(\)](#) method, specifying its [UniqueId](#) value.

## Adding or Modifying Install Conduit Configuration Entries

With the `PDInstallConduit` object, you can also create your own install conduit configuration entries or modify the predefined entries. To add your own, call the [SetDWORDData\(\)](#) or [SetStringData\(\)](#) methods, passing in the name of the entry you want to create/modify and the value you want to assign it.



## Installing a Notifier with PDCondMgr

In addition to handling conduit information, the [PDCondMgr](#) object also registers and unregisters notifier DLLs with HotSync Manager. It uses only the filename of your notifier to register it. To register your notifier, call the [RegisterNotifier\(\)](#) method and pass it the filename. To unregister it, call the [UnregisterNotifier\(\)](#) method. You can also specify a fully qualified path, but note that these methods only read/write configuration entries. Therefore if you specify a full path when you register a notifier, you must specify the same full path when you unregister it.

The PDCondMgr object provides several other methods for accessing information about registered notifiers. See “[PDCondMgr](#)” on page 26 in the *COM Sync Suite Reference* for details on these additional methods.

## Uninstalling Your Conduit

It is customary that applications provide end users a facility to easily remove all the relevant files and other information that were put on the user’s computer during installation. Your uninstall application must:

- Unregister your conduit—that is, it must delete the configuration entries it created. The Conduit Manager API provides functions for removing configuration entries. You can do this easily in COM Sync via [PDCondMgr.UnregisterConduit\(\)](#) or [PDSystemCondMgr.UnregisterConduit\(\)](#).
- Unregister your notifier DLLs, if you installed them.
- Remove all installed files.
- Perform any other cleanup.

And ideally, if your conduit replaced a conduit that was already installed on the end user’s machine (like the conduits for any of the built-in applications), you should at least offer the user the option to reregister it with HotSync Manager, leaving the user’s machine exactly as you found it.

---

**NOTE:** If you installed the COM Sync libraries on a machine running a version of HotSync Manager earlier than 6.0 (as described in [“Files to Install”](#) on page 53), then you must *not* uninstall them. The reason is that other third-party COM-based conduits might be installed as well as yours on the end user’s computer. If you uninstall the COM Sync libraries, then you make it impossible for any other remaining COM-based conduits to run.

---

## Testing Your Installer

Because one of the most important goals of the HotSync process is simplicity for users, ensure that your conduit installs and uninstalls correctly and easily. PalmSource, Inc. strongly recommends that, at a minimum, you test the following conduit installation and removal conditions:

- If the COM Sync module is not present, your conduit is being installed on a desktop running a version of HotSync Manager earlier than 6.0 and should fail gracefully.
- 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.
- After removal of your conduit, the conduit configuration entries are restored to what they were prior to its installation.

Unlike earlier versions, HotSync Manager versions 6.0 and later automatically recognize newly registered conduits (earlier versions required that you call a method to refresh HotSync Manager after changing a conduit’s configuration entries). However, if you change any other configuration entries—for example, the backup conduit, communications settings, and so on—you still must restart HotSync Manager regardless of the version.

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.

## Installation Troubleshooting Tips

This section suggests a few things to look for when you have installation problems:

- Check the HotSync log after you run your conduit. The log contains any error messages that may result. Ensure that you are looking at the log entry for the correct user.
- For additional (verbose) logging information, exit HotSync Manager, click the **Start** button, then **Run**, and type  
`c:\<HotSync Manager directory>\hotsync.exe -v`.
- Conduits are run by HotSync Manager versions earlier than 6.0 only when there is a application with a matching creator ID on the handheld. HotSync Manager versions 6.0 and later run all COM-based conduits regardless of whether a matching application is present.
- Use CondCfg.exe to view the conduit configuration settings.

See the [Conduit Development Utilities Guide](#) for details on using the Conduit Configuration utility and other helpful tools.



# 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:

<a href="#">Expansion Support</a>	. . . . .	72
<a href="#">Architectural Overview</a>	. . . . .	75
<a href="#">Standard Directories</a>	. . . . .	81
<a href="#">Card Insertion and Removal</a>	. . . . .	82
<a href="#">Checking for Expansion Cards</a>	. . . . .	83
<a href="#">Volume Operations</a>	. . . . .	91
<a href="#">File Operations</a>	. . . . .	95
<a href="#">Directory Operations</a>	. . . . .	99
<a href="#">Example of Getting a File</a>	. . . . .	105
<a href="#">Custom Calls</a>	. . . . .	107
<a href="#">Debugging</a>	. . . . .	108
<a href="#">Summary of Expansion and VFS Managers</a>	. . . . .	109

---

**NOTE:** The Card Explorer sample in <CDK>\COM\Samples\ExpansionCard\CardExplorer illustrates many of the concepts explained in this chapter.

---

## 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 using COM Sync database adapters (see “[Database Support and Extensibility](#)” on page 28).

**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

COM-based conduits access primary storage during a HotSync operation with the Sync Manager via COM Sync objects called **database adapters**. To access secondary storage, however, they use the Expansion and VFS Managers, represented as several COM Sync objects shown in [Figure 5.1](#) on page 76. These have been designed to support as broad a range of serial expansion architectures as possible.

Desktop applications or installers have always used the Install Aide to queue applications and databases for an install conduit to install in primary storage on the handheld during the next HotSync operation. With version 4.0 of the Palm OS platform, they can also use the Install Aide to queue files for an install conduit to copy from the desktop to secondary storage on expansion cards. Similarly, the User Data API enables desktop applications and installers to access information about handheld users on the desktop; now they can also access information about expansion slots on users' handhelds. The User Data and Install Aide APIs are exposed in the COM Sync module through the [PDUserData](#) and [PDInstall](#) objects.

For more information about the [PDUserData](#) and [PDInstall](#) objects, see "[Accessing User Data with PDUserData](#)" on page 64 and "[Installing Files on the Handheld with PDInstall](#)" on page 58.

## 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.

## Using Expansion Technology

### Expansion Support

---

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 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 where to learn more about custom C API 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

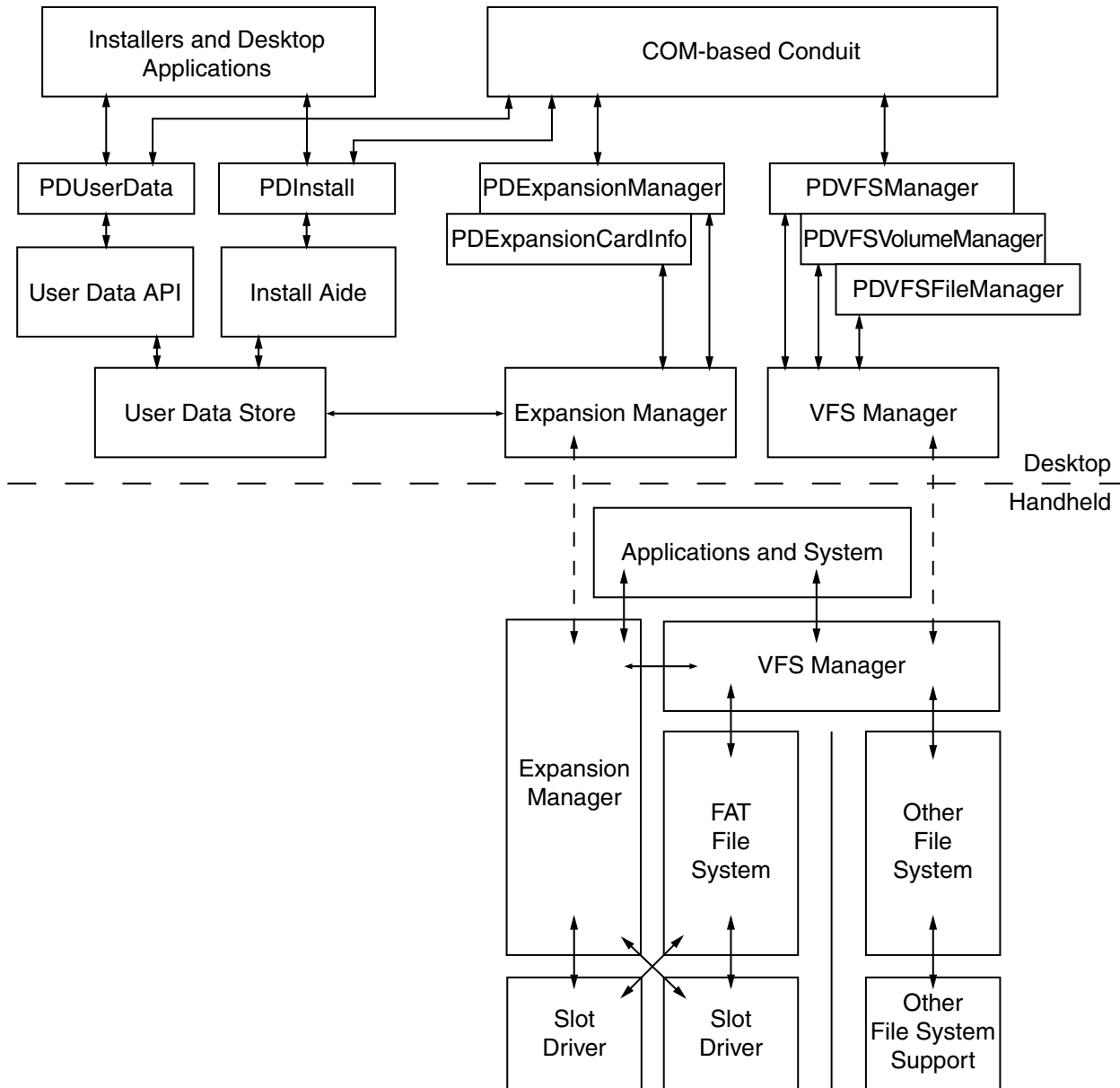
[Figure 5.1](#) illustrates the Palm OS expansion architecture. It is designed to be flexible enough to support multiple file systems and diverse physical expansion mechanisms while still presenting a consistent set of APIs to applications and to other parts of Palm OS. The following sections describe 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](#)

## Using Expansion Technology

### Architectural Overview

**Figure 5.1 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 5.1](#), 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.

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, which is represented by the COM Sync module as a [PDExpansionManager](#) object. 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 Data and Install Aide APIs, which are not used during a HotSync operation, use slot IDs to identify slots instead of slot reference numbers. These APIs—represented by the COM Sync module as [PDUUserData](#) and [PDInstall](#) objects, respectively—simply return 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 **file systems**, including

## Using Expansion Technology

### *Architectural Overview*

---

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.

### **VFS Manager and the Sync Manager APIs**

With the addition of the VFS Manager to the COM Sync Suite, there are now two distinct ways that conduits can store and retrieve Palm OS users' data:

- COM Sync database adapter objects access resource and record databases in the handheld's primary storage RAM via the Sync Manager API on the desktop. The Sync Manager 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 database adapter objects to store and retrieve a Palm OS user's 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 COM Sync module's VFS Manager objects can support the widest variety of file systems. Use these objects when your conduit needs to read and write data stored on external media.

Conduits should use the appropriate objects for each given situation. Database adapters—via 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 objects 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 database adapters, see "[Database Support and Extensibility](#)" on page 28.

#### COM Sync VFS Manager Objects

As shown in [Figure 5.1](#) on page 76, COM-based conduits access the VFS Manager via these COM Sync objects:

**[PDVFSManager](#)**: Represents the VFS Manager on the desktop. Its methods can determine whether and how many volumes are available on an expansion card, get a volume reference number for each mounted volume on the card, and create a [PDVFSVolumeManager](#) object.

**[PDVFSVolumeManager](#)**: Represents a volume on an expansion card, created by a [PDVFSManager](#) object. Its methods can create and open files and directories, copy files to and from the desktop, and import/export files and Palm OS databases between primary storage memory and an expansion card.

**[PDVFSFileManager](#)**: Represents a file or directory created or opened by a [PDVFSVolumeManager](#) object. Its methods can read and write open files or directories.

The following sections in this chapter provide examples using the VFS Manager objects. For details on these objects, see [Chapter 3](#), “[Objects](#),” on page 7 in the *COM Sync Suite Reference*.

#### 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 on the desktop provides an interface to the Expansion Manager on the handheld *during* a HotSync operation. Conduits that use the COM Sync module’s Expansion

Manager objects can determine whether an expansion card is present in a slot and get information about those cards.

### COM Sync Expansion Manager Objects

As shown in [Figure 5.1](#) on page 76, COM-based conduits access the Expansion Manager via these COM Sync objects:

**[PDExpansionManager](#):** Represents the Expansion Manager on the desktop. Its methods can detect the presence of and get information about expansion slots on a handheld and the cards in them. It can get slot reference numbers, which you subsequently use to get a [PDExpansionCardInfo](#) object.

**[PDExpansionCardInfo](#):** Represents expansion card information given a slot reference number, which is provided by a [PDExpansionManager](#) object. This object's properties include the card manufacturer's name, the type of media, and whether the card has storage and is read-only.

The following sections in this chapter provide examples using the Expansion Manager objects. For details on these objects, see [Chapter 3, "Objects,"](#) on page 7 in the *COM Sync Suite Reference*.

## 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 5.1](#) 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 5.1 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 102 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 method fails. PalmSource, Inc. recommends that your conduit always check errors returned by all methods and add helpful messages to the user in the HotSync log.

## Checking for Expansion Cards

Before accessing an expansion card, your conduit should check the following conditions in this order:

1. [Is an Expansion Slot Present?](#)

Your conduit should first make sure that the handheld supports expansion by verifying the presence of the Expansion and VFS Managers, which are present only if the handheld has an expansion slot.

2. [How Many Slots are Present?](#)

Once you know that at least one slot is present, you should enumerate them and determine which your conduit should use.

3. [Is a Card in the Slot?](#)

If there is, you can now ascertain the capabilities of the card: whether it has memory, whether it does I/O, and so on.

4. [Is a Mounted Volume on the Card?](#)

If there is, you can access files and directories on it.

This section discusses each of the above steps. The last two subsections are:

## Using Expansion Technology

### *Checking for Expansion Cards*

---

- [Example of Checking for an Expansion Slot, Card, and Volume](#)
- [Determining a Card's Capabilities](#)

### Is an Expansion Slot Present?

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 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 methods. This section presents two ways of verifying the presence of the VFS Manager on the handheld:

- [Using the IsExpansionSlotPresent Method](#)
- [Using the ReadFeature Method](#)

### Using the IsExpansionSlotPresent Method

The desktop Expansion Manager object provides the [IsExpansionSlotPresent\(\)](#) method to verify whether the handheld has an expansion slot. Actually, at the beginning of each HotSync operation, the desktop VFS Manager checks for the presence of the Expansion Manager and VFS Manager system extensions on the handheld before HotSync Manager calls your conduit to run.

When you call [IsExpansionSlotPresent\(\)](#), it returns `True` if these extensions are present and `False` otherwise. If they are present, you can infer that a slot is present, because no handheld ships with these extensions unless it has a slot.

---

**NOTE:** The [IsExpansionSlotPresent\(\)](#) method 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 this method—which makes it more efficient than using [ReadFeature\(\)](#).

---

“[Example of Checking for an Expansion Slot, Card, and Volume](#)” on page 89 shows the use of `IsExpansionSlotPresent`.

## Using Expansion Technology

### Checking for Expansion Cards

---

#### Using the ReadFeature Method

The [PDSystemAdapter](#) object provides the [ReadFeature\(\)](#) method to verify the presence of defined feature sets on the handheld. To check for the VFS Manager's system feature, do the following:

- call [ReadFeature\(\)](#)
- supply sysFileCVFSMgr for the feature creator
- supply vfsFtrIDVersion for the feature number

These constants are defined in the C header file `VFSMgr.h`.

[Listing 5.1](#) shows how to use [ReadFeature\(\)](#) to check for the presence and proper version of the VFS Manager.

#### Listing 5.1 ReadFeature to verify the presence of the VFS Manager

---

```
Dim pSystem as New PDSystemAdapter
Dim Feature as Long

Feature = pSystem.ReadFeature("v fsm", 0)

' On error, VFS Manager is not present, so fail gracefully.

' On success, call VFS Manager methods as required.
' If you check the return value of Feature and it is the
' expected version number, everything is OK.
```

---

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 [ReadFeature\(\)](#)
- supply sysFileCExpansionMgr for the feature creator
- supply expFtrIDVersion for the feature number

These constants are defined in the C header file `ExpansionMgr.h`.

[Listing 5.2](#) shows how to use [ReadFeature\(\)](#) to check for the presence and proper version of the Expansion Manager.

**Listing 5.2 ReadFeature to verify the presence of the Expansion Manager**

---

```
Dim pSystem as New PDSystemAdapter
Dim Feature as Long

Feature = pSystem.ReadFeature("expn", 0)

' On error, Expansion Manager is not present, so fail
' gracefully.

' On success, call Expansion Manager methods as required.
' If you check the return value of Feature and it is the
' expected version number, everything is OK.
```

---

## How Many Slots are Present?

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 [GetSlotReferenceNumbers\(\)](#) method makes it simple to enumerate these slots, getting a slot reference number for each.

The array of slot reference numbers passed back by [GetSlotReferenceNumbers\(\)](#) uniquely identify all slots. A slot reference number can be supplied to various [PDExpansionManager](#) methods to obtain information about the slot, such as whether there is a card or other expansion module present in the slot. "[Example of Checking for an Expansion Slot, Card, and Volume](#)" on page 89 illustrates this.

## Is a Card in the Slot?

With the [PDExpansionManager](#) object for a specific expansion slot, you can easily determine whether a card is in that slot. Call its [GetSlotInfo\(\)](#) method and check the `IsCardPresent` parameter that it passes back.

## Is a Mounted Volume on the Card?

Conduits rely on the fact that Palm OS automatically mounts any recognized volumes inserted into or snapped onto the handheld.

---

**NOTE:** The slot driver provided by PalmSource currently mounts only one volume per expansion slot.

---

Depending on your needs, you can use either of the following methods to determine whether any volumes are mounted:

- [PDVFSManager.IsVolumeAvailable\(\)](#) determines whether any volumes are currently mounted on any card in any slot. Notice that this does not tell you anything about which slot or which card the volume is on.
- [PDExpansionManager.GetSlotInfo\(\)](#) passes back a value in its `IsVolumeMounted` parameter, which indicates whether a volume is mounted on this card in this slot.

After you have confirmed that a volume is present, you need a volume reference number to specify it. Again, depending on your needs, you can use either of the following methods to get a volume reference number:

- [PDExpansionManager.GetSlotInfo\(\)](#) passes back a volume reference number in its `VolRefNum` parameter for the first volume on the card in this slot.
- [PDVFSManager.GetVolumeReferenceList\(\)](#) returns a list of the volume reference numbers of all mounted volumes on cards in all slots.

You can then pass that volume reference number to the [GetVolumeManager\(\)](#) method to create a [PDVFSVolumeManager](#) object, with which you can manipulate files and directories on that volume.

See “[Example of Checking for an Expansion Slot, Card, and Volume](#)” for an example that illustrates the use of the `GetSlotInfo` and `GetVolumeReferenceList` methods to perform these tasks as well as the checks described previously in this section.

## Example of Checking for an Expansion Slot, Card, and Volume

[Listing 5.3](#) illustrates how to perform the tasks described in the preceding subsections. The example VB subroutine `GetVolume` performs all the checks previously described and then displays information about the first volume it finds.

### Listing 5.3 `GetVolume` example

---

```
Public Sub GetVolume()  
  
    Dim ExpansionSlot As New PDExpansionManager  
    Dim SlotNumbers As Variant  
    Dim IsCardPresent As Boolean  
    Dim IsVolumeMounted As Boolean  
    Dim VolumeRef As Long  
    Dim VolumeCount As Long  
  
    Dim VFSManager As New PDVFSManager  
    Dim VFSVolume As PDVFSVolumeManager  
    Dim VolumeList() As Integer  
    Dim i As Integer  
  
    On Error GoTo ErrorHandler  
  
    If ExpansionSlot.IsExpansionSlotPresent Then  
        ' Get a list of all available slots.  
        SlotNumbers = ExpansionSlot.GetSlotReferenceNumbers  
        ' Retrieve slot information from the first available slot.  
        For i = 0 To UBound(SlotNumbers)  
            Call ExpansionSlot.GetSlotInfo(SlotNumbers(i), IsCardPresent, _  
                IsVolumeMounted, VolumeRef)  
            If IsCardPresent Then Exit For  
        Next i  
  
        If IsCardPresent Then  
            If IsVolumeMounted Then  
                VolumeList = VFSManager.GetVolumeReferenceList(VolumeCount)  
                ' Use the first available volume  
                Set VFSVolume = VFSManager.GetVolumeManager(VolumeList(0))  
            End If  
        Else  
            MsgBox("No card present in Slot ", vbInformation, "Information")  
            Exit Sub  
        End If  
    End If
```

## Using Expansion Technology

### Checking for Expansion Cards

---

```
Else
    MsgBox("No Expansion slot was found.", vbInformation, "Information")
    Exit Sub
End If

' Display the amount of used space on the current volume.
MsgBox(VFSVolume.UsedSpace & " Bytes found on Volume '" & VFSVolume.Label _
    & "'", vbInformation, "Information")

Exit Sub

ErrorHandler:
    MsgBox("Error " & Err.Number & " " & Err.Description, vbExclamation, _
        "GetVolume")
    Exit Sub
End Sub
```

---

## 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, it's 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 COM Sync module does not support access to custom control, but "[Custom Calls](#)" on page 107 tells you where to learn more about custom C API calls to access custom file systems or I/O devices.



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 [GetCardInfo\(\)](#) method to get a [PDExpansionCardInfo](#) object whose properties specify the following:

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

Note that the existence of the [GetCardInfo\(\)](#) method 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 [GetVolumeReferenceList\(\)](#). This method, the use of which is illustrated in "[Is a Mounted Volume on the Card?](#)" on page 88, returns a list of volume reference numbers, which you select from and then supply to the [GetVolumeManager\(\)](#) method, which creates a [PDVFSVolumeManager](#) object. This object provides the methods you need to manipulate files and directories on a volume.

---

**NOTE:** Volume reference numbers can change each time the handheld mounts a given volume. To keep track of a particular volume, save the [PDVFSVolumeManager](#) object's [Label](#) property rather than its reference number.

---

## Using Expansion Technology

### *Volume Operations*

---

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.

You can use the [Format\(\)](#) method to reformat an already formatted volume; it cannot format an unformatted card. Once the card has been formatted, the VFS Manager automatically mounts it and can assign it a new volume reference number. Despite the change in its volume reference number, you can continue to use the same [PDVFSVolumeManager](#) object from which you formatted the volume. But the list of volume reference numbers you obtained with [GetVolumeReferenceList\(\)](#) is no longer valid. This is another reason not to use volume reference numbers to track a particular volume.

The [PDVFSVolumeManager.Label](#) property allows you to get and set the volume label. 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 95 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 set the [Label](#) property. Setting this property may create or delete a file in the root directory, which would invalidate any current calls to [GetFileList\(\)](#).

---

Additional information about the volume can be obtained through getting the [TotalCapacity](#), [UsedSpace](#), and other [PDVFSVolumeManager](#) properties. These properties return 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. Other properties provide various pieces of information about the volume, including:

- whether the volume is hidden ([Attributes](#))
- whether the volume is read-only ([Attributes](#))
- whether the volume is supported by a slot driver, or is being simulated by Palm OS Emulator ([mountClass](#))

- the slot with which the volume is associated ([SlotReferenceNumber](#)), and the reference number of the slot driver controlling the slot ([SlotLibRefNumber](#))
- the type of media on which this volume is located, such as SD, CompactFlash, or Memory Stick ([MediaType](#))

All of the above information is provided by the properties of a [PDVFSVolumeManager](#) object. Whether the volume is hidden or read-only is further encoded in the [Attributes](#) property; see “[VFS Volume Attributes](#)” on page 576 in the *COM Sync Suite Reference* for the bits that make up this property.

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. A [PDVFSVolumeManager](#) object looks for this file and, if found, sets the `vfsVolumeAttrHidden` bit along with the volume’s other attributes in the [Attributes](#) property.

---

**NOTE:** The bit values that the volume [Attributes](#) property can be set to and described in “[VFS Volume Attributes](#)” on page 576 in the *COM Sync Suite Reference* are defined in the C header file (`VFSMgr.h`) of the *desktop* VFS Manager API. However, the `vfsVolumeAttrHidden` bit is not defined in that header file but can be passed back in an [Attributes](#) property. 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 [GetCardInfo\(\)](#) method. Often, the information provided by the properties of a [PDVFSVolumeManager](#) object 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 properties of a [PDVFSVolumeManager](#) object include a [SlotReferenceNumber](#) property that can be passed to [GetCardInfo\(\)](#). 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 just as simple: call the [GetSlotInfo\(\)](#) method, passing it a slot reference number, and it returns the volume reference number of the first mounted volume on a card in that slot. (The current slot driver provided by PalmSource supports only one volume per card.)

## 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 98.

This section covers the following topics:

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

## Common File Operations

As shown in [Table 5.2](#), the VFS Manager provides all of the standard file operations that should be familiar from desktop and larger computer systems. Because these methods work largely as you would expect, their use is not detailed here. See the descriptions of each individual method in [Chapter 4](#), “[Methods](#),” on page 111 in the *COM Sync Suite Reference* for the parameters, return values, and side effects of each.

Most file operations are performed on [PDVFSFileManager](#) objects, except for a few such as the [PDVFSVolumeManager](#)’s [Open\(\)](#) method, which creates a [PDVFSFileManager](#) object to represent a file. Then you call its methods to read, write, and close the file. Note that the [PDVFSVolumeManager](#)’s methods operate only on *closed* files, whereas [PDVFSFileManager](#)’s methods operate only on *open* files.

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

**Table 5.2 Common file operations**

Function	Description
<a href="#">PDVFSVolumeManager</a> methods	
<a href="#">Open()</a> method	Opens a file, given a file path and open mode. Opening a file creates a <a href="#">PDVFSFileManager</a> object to represent it.
<a href="#">CreateFile()</a> method	Creates a file on this volume given a path.
<a href="#">Delete()</a> method	Deletes a closed file.
<a href="#">Rename()</a> method	Renames a closed file.

**Table 5.2 Common file operations (*continued*)**

Function	Description
<a href="#">PDVFSFileManager</a> methods and properties	
<a href="#">Close()</a> method	Closes an open file.
<a href="#">Read()</a> method	Reads data from an open file into the specified buffer.
<a href="#">Write()</a> method	Writes data to an open file.
<a href="#">Seek()</a> method	Sets the position within an open file from which to read or write.
<a href="#">Tell()</a> method	Gets the current position of the file pointer within an open file.
<a href="#">EOF</a> property	Indicates whether the file pointer has reached the end of the file.
<a href="#">Size</a> property	Size of a file or what to resize a file to.
<a href="#">Attributes</a> property	Attributes of a file, including hidden, read-only, system, and archive bits. See “ <a href="#">VFS File and Directory Attributes</a> ” on page 574 in the <i>COM Sync Suite Reference</i> for the bits that make up this property.
<a href="#">CreationDate</a> , <a href="#">LastModificationDate</a> , <a href="#">LastAccessedDate</a> properties	Dates on which an open file was created, last modified, and last accessed.

To open a file with the [PDVFSVolumeManager](#)’s [Open\(\)](#) method, you specify only its path and an open mode (read-only, exclusive open, and so on). This method returns a [PDVFSFileManager](#) object, which uniquely identifies the file. 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.



---

**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 [ExportDatabaseToFile\(\)](#) method 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 [ImportDatabaseFromFile\(\)](#) method 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 methods when moving Palm OS databases between primary storage memory on a handheld and secondary storage on an expansion card.

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

---

**IMPORTANT:** The import and export methods do not work with schema databases.

---

## 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 100. One common operation—determining the files that are contained within a given directory—is covered in “[Enumerating the Files and Subdirectories 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 102 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 and Subdirectories 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 98 for details on valid characters to use in file and directory names.

## Common Directory Operations

As shown in [Table 5.3](#), the VFS Manager provides all of the standard directory operations that should be familiar from desktop and larger computer systems. Because these methods work largely as you would expect, their use is not detailed here. See the descriptions of each individual method in [Chapter 4, “Methods,”](#) on page 111 in the *COM Sync Suite Reference* for the parameters, return values, and side effects of each.

Most directory operations are performed on [PDVFSFileManager](#) objects, except for a few such as the [PDVFSVolumeManager](#)’s [Open\(\)](#) method, which creates a [PDVFSFileManager](#) object to represent a directory. Then you call its methods to read, write, and close the directory. Note that the [PDVFSVolumeManager](#)’s methods operate only on *closed* directories, whereas [PDVFSFileManager](#)’s methods operate only on *open* directories.

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

**Table 5.3 Common directory operations**

Function	Description
<a href="#">PDVFSVolumeManager</a> methods	
<a href="#">Open()</a> method	Opens a directory, given a file path and open mode. Opening a directory creates a <a href="#">PDVFSFileManager</a> object to represent it.
<a href="#">CreateDirectory()</a> method	Creates a new directory on this volume given a path.
<a href="#">Delete()</a> method	Deletes a closed directory.
<a href="#">Rename()</a> method	Renames a closed directory.
<a href="#">PDVFSFileManager</a> methods and properties	
<a href="#">Attributes</a> property	Attributes of a directory, including read-only and slot-based bits. See “ <a href="#">VFS Volume Attributes</a> ” on page 576 in the <i>COM Sync Suite Reference</i> for the bits that make up this property.
<a href="#">CreationDate</a> , <a href="#">LastModificationDate</a> , <a href="#">LastAccessedDate</a> properties	Dates on which an open directory was created, last modified, and last accessed.

To open a directory with the [PDVFSVolumeManager](#)’s [Open\(\)](#) method, you specify only its path and an open mode (read-only, exclusive open, and so on). This method returns a [PDVFSFileManager](#) object, which uniquely identifies the directory. 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 “/”.

## Enumerating the Files and Subdirectories in a Directory

To enumerate the files and subdirectories within a directory, call the [GetFileList\(\)](#) and [GetSubDirectoryList\(\)](#) methods, respectively.

## Determining the Default Directory for a Particular File Type

As explained in “[Standard Directories](#)” on page 81, 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.

A [PDVFSVolumeManager](#) object’s [GetDefaultDirectory\(\)](#) method takes 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”.

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.

[GetDefaultDirectory\(\)](#) 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 5.4](#) under the `expMediaType_Any` media type, which [GetDefaultDirectory\(\)](#) reverts to when there is no default registered by the slot driver for a given media type. (The supported media type is specified in the [MediaType](#) property in [PDExpansionCardInfo](#) and [PDVFSVolumeManager](#) objects.)

**Table 5.4** 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/
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/

**Table 5.4 Default directory registrations (*continued*)**

File Type	Path
.wav	/AUDIO/
audio/x-wav	/AUDIO/

The SD slot driver provided by PalmSource, Inc. registers the file types in [Table 5.5](#), because it has an appropriate specification for these file types.

**Table 5.5 Directories registered by the SD slot driver**

File Type	Path
.jpg	/DCIM/
.jpeg	/DCIM/
image/jpeg	/DCIM/
.qt	/DCIM/
.mov	/DCIM/
video/quicktime	/DCIM/
.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 card install conduit (`CardInst.dll`) 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.

## Example of Getting a File

[Listing 5.4](#) is an example VB subroutine named `GetFile`, which performs the checks described in “[Checking for Expansion Cards](#)” on page 83 and then gets a list of directories from the root. If a directory exists, it gets a list of files from the first directory in the list. It then opens the first file, if it exists, and displays some file information.

## Using Expansion Technology

### *Example of Getting a File*

---

#### Listing 5.4    GetFile example

---

```
Public Sub GetFile()

    Dim ExpansionSlot As New PDExpansionManager
    Dim SlotNumbers As Variant
    Dim IsCardPresent As Boolean
    Dim IsVolumeMounted As Boolean
    Dim VolumeRef As Long
    Dim VolumeCount As Long

    Dim VFSManager As New PDVFSManager
    Dim VFSVolume As PDVFSVolumeManager
    Dim VFSFile As PDVFSFileManager
    Dim VolumeList() As Integer
    Dim i As Integer

    Dim DirectoryCount As Integer
    Dim FileCount As Integer
    Dim DirectoryList As Variant
    Dim FileList As Variant

    On Error GoTo ErrorHandler

    If ExpansionSlot.IsExpansionSlotPresent Then
        ' Get a list of all available slots.
        SlotNumbers = ExpansionSlot.GetSlotReferenceNumbers
        ' Retrieve slot information from the first available slot.
        For i = 0 To UBound(SlotNumbers)
            Call ExpansionSlot.GetSlotInfo(SlotNumbers(i), IsCardPresent, _
                IsVolumeMounted, VolumeRef)
            If IsCardPresent Then Exit For
        Next i

        If IsCardPresent Then
            If IsVolumeMounted Then
                VolumeList = VFSManager.GetVolumeReferenceList(VolumeCount)
                ' Use the first available volume.
                Set VFSVolume = VFSManager.GetVolumeManager(VolumeList(0))
            End If
        Else
            MsgBox("No card present in Slot ", vbInformation, "Information")
            Exit Sub
        End If
    Else
        MsgBox("No Expansion slot was found.", vbInformation, "Information")
        Exit Sub
    End If
```



```
' Retrieve a list of the topmost directories.
DirectoryCount = VFSVolume.GetSubDirectoryList("/", DirectoryList)

If DirectoryCount > 0 Then
    ' Retrieve a list of files from the first directory.
    FileCount = VFSVolume.GetFileList(DirectoryList(0), FileList)
    If FileCount > 0 Then
        ' Open the first file found in the first directory in read-only mode.
        Set VFSFile = VFSVolume.Open(DirectoryList(0) & "/" & FileList(0), _
            PDDirectLib.EPDVFSFileOpenAttr.eVFSModeRead)

        ' Display file information.
        MsgBox("Opened '" & DirectoryList(0) & "/" & FileList(0) & "'" & _
            vbCr & vbCr & "File Size   : " & VFSFile.Size & " Bytes " & vbCr & _
            "Created on : " & VFSFile.CreationDate, vbInformation, _
            "Information")

        VFSFile.Close
    End If
End If

Exit Sub

ErrorHandler:
    MsgBox("Error " & Err.Number & " " & Err.Description, vbExclamation, _
        "GetFile")
    Exit Sub

End Sub
```

---

## 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\(\)](#), is available only in the C API; it is not exposed in the COM Sync module. For more information on accessing custom file systems and custom I/O from the VFS Manager's C API, see "[Custom Calls](#)" on page 80 in the *C/C++ Sync Suite Companion*.

## Debugging

Palm OS Simulator supports the expansion capabilities of the VFS Manager. It presents a directory on the host file system as a volume to the virtual file system. You can populate this directory on your host system and then simulate a volume mount. Changes made to the simulated expansion card's contents can be verified simply by examining the directory on the host.

For more information on configuring and operating Palm OS Simulator, see *Palm OS Cobalt Simulator Guide* or *Testing with Palm OS Garnet Simulator*. For information on performing HotSync operations with Simulator, see [Chapter 8, "Synchronizing with Palm OS Simulator,"](#) on page 49 of the *Conduit Development Utilities Guide*.

## Summary of Expansion and VFS Managers

---

### Expansion Manager Methods ([PDExpansionManager](#))

<a href="#">GetCardInfo()</a>	<a href="#">GetSlotReferenceNumbers()</a>
<a href="#">GetSlotInfo()</a>	<a href="#">IsExpansionSlotPresent()</a>

---

---

### Expansion Card Information Properties ([PDExpansionCardInfo](#))

<a href="#">CapabilityFlags</a>	<a href="#">ManufacturerName</a>
<a href="#">DeviceClass</a>	<a href="#">MediaType</a>
<a href="#">DeviceUniqueId</a>	<a href="#">ProductName</a>

---

---

### VFS Manager Methods ([PDVFSManager](#))

<a href="#">GetVolumeCount()</a>	<a href="#">GetVolumeReferenceList()</a>
<a href="#">GetVolumeManager()</a>	<a href="#">IsVolumeAvailable()</a>

---

---

### VFS Volume Manager ([PDVFSVolumeManager](#))

#### Methods

<a href="#">CopyFileFromDeskTop()</a>	<a href="#">GetDefaultDirectory()</a>
<a href="#">CopyFileToDeskTop()</a>	<a href="#">GetFileList()</a>
<a href="#">CreateDirectory()</a>	<a href="#">GetSubDirectoryList()</a>
<a href="#">CreateFile()</a>	<a href="#">ImportDatabaseFromFile()</a>
<a href="#">Delete()</a>	<a href="#">Open()</a>
<a href="#">ExportDatabaseToFile()</a>	<a href="#">Rename()</a>
<a href="#">Format()</a>	

#### Properties

<a href="#">Attributes</a>	<a href="#">SlotLibRefNumber</a>
<a href="#">FileSystemType</a>	<a href="#">SlotReferenceNumber</a>
<a href="#">Label</a>	<a href="#">TotalCapacity</a>
<a href="#">MediaType</a>	<a href="#">UsedSpace</a>
<a href="#">mountClass</a>	

---

## Using Expansion Technology

*Summary of Expansion and VFS Managers*

---

---

### VFS File Manager ([PDVFSFileManager](#))

---

#### Methods

[Close\(\)](#)

[Read\(\)](#)

[Seek\(\)](#)

[Tell\(\)](#)

[Write\(\)](#)

#### Properties

[Attributes](#)

[CreationDate](#)

[EOF](#)

[LastAccessedDate](#)

[LastModificationDate](#)

[Size](#)

---

# Index

---

## A

- ActiveX client 12
- adapters 28
- Address Book 19, 29
- application name
  - on expansion cards 99
- architecture
  - COM Sync Suite 9
  - expansion 75
  - extensible database adapters 28
- Attributes property 93, 97, 101

## B

- Borland C/C++ Builder 5
- Borland Delphi 5

## C

- CardInfo application 93
- cards, expansion
  - about 74
  - insertion and removal 82
- CDK
  - documentation viii
- ChangeFileDestinationHHToSlot() 62
- ChangeFileDestinationSlotToHH() 62
- ChangeFileSlotDestination() 62
- classic databases
  - objects, COM Sync Suite 19
- client module types 12
  - ActiveX client 12
  - standard EXE client 12
- Close() 97
- COM error handling
  - C/C++ 27
  - Visual Basic 26
- COM ProgIds 25
- COM Sync
  - architecture 9
  - client/server DLLs 10
  - features 3
  - module 2
  - objects 14
    - classic databases 19
    - database adapters 28

- expansion 24, 81
  - extended databases 18
  - installation and support 22, 51
  - PIM database adapters 29
  - schema databases 16
  - system information 15
  - VFS Manager 80
- COMConduit.dll 10
- COMDirect.dll 10
- CompactFlash 73
- COMStandard.dll 10
- conduits
  - architecture
    - COM Sync 9
  - building, COM-based 32
  - development requirements, COM Sync 33
  - entry points
    - COM interface 12
  - installing 51
  - samples 6
- CreateDirectory() 101
- CreateFile() 96
- CreationDate 97, 101

## D

- databases
  - adapters for COM Sync 28
  - COM support for 28
  - on expansion cards 98
  - PIM database adapters 29
- Date Book 19, 29
- debugging conduits
  - expansion cards 108
- default directories 82
  - determining by file type 102
  - registered upon initialization 103
  - SD slot driver 104
- Delete() 96, 101
- deployment
  - installer 51
- directories
  - basic operations 99
  - default for file type 102
  - enumerating files within 102
  - referencing 101

---

directory names, install 60  
DisplayLog() 63  
DmConduit.dll 10  
documentation viii

## E

EOF property 97  
error handling, COM 26  
expansion 71–110  
    architecture 75  
    auto-start PRC 82  
    custom calls 107  
    databases, working with 98  
    debugging 108  
    default directories 82  
    enumerating slots 87  
    file system operations 95  
    file systems 77  
    installing files from desktop 62  
    mounted volumes 88  
    naming applications on expansion cards 99  
    PDInstall object 62  
    ROM 74  
    slot driver 77  
    slot reference number 77  
    standard directories 81  
    standard directory layout 81  
    volume operations 91  
expansion cards 74  
    capabilities 90  
    checking for 83  
    insertion and removal 82  
    objects, COM Sync Suite 24  
    verifying presence in slots 87  
Expansion Manager 80  
    checking card capabilities 90  
    COM Sync objects 81  
    enumerating slots 87  
    PDExpansionCardInfo object 81  
    PDExpansionManager object 81  
    purpose 80  
    slot reference number 77  
    verifying presence of 84  
expansion slots 73  
    verifying presence of 84

expMediaType\_Any constant 103  
ExportDatabaseToFile method 99  
extended databases  
    objects, COM Sync Suite 18  
extensible object model 28

## F

FAT 78  
features, COM Sync Suite 3  
file systems 77  
    and filenames 98  
    and volume names 95  
    basic operations 95  
    custom calls to 107  
    FAT 78  
    implementation 78  
    long filenames 78  
    multiple 78  
    nonstandard functionality 107  
    VFAT 78, 98  
filenames  
    long 78  
files  
    enumerating 102  
    installing on expansion cards 62  
    naming 78, 95, 98  
    paths to 97  
    referencing 97  
Format() 92  
formatting volumes 92

## G

GetCardInfo() 91  
GetCommStatus method 64  
GetDefaultDirectory() 102  
GetFileList() 102  
GetSlotFileCount method 63  
GetSlotFileSize method 63  
GetSlotInfo() 87, 88  
GetSlotReferenceNumbers() 87  
GetSubDirectoryList() 102  
GetVolumeReferenceList() 88

---

## H

handheld-install folder, defined 61  
HotSync Exchange  
    To/From directories, defined 61  
HotSync Manager  
    COM API  
        using 63  
    progress display 13  
HotSync user folder, defined 61

## I

ImportDatabaseFromFile() 99  
install conduits  
    Install Aide 58  
    installing 65  
    mask 59  
install directory names 60  
installation and support, COM Sync Suite  
    objects 22  
installer 51  
    files 53  
    tasks 52  
    testing 68  
    troubleshooting 69  
InstallFileToSlot method 62  
installing  
    conduits 55  
    files on expansion cards 62  
    files on the handheld 58  
    notifiers 67  
    PDCondMgr, using 55  
IPDAInitialize interface 30  
IPDClientNotify  
    about 12  
    implementing 45  
IsExpansionSlotPresent() 85  
IsSyncInProgress() 64  
IsVolumeAvailable() 88

## J

Java 5

## L

Label 92

LastAccessedDate 97, 101  
LastModificationDate 97, 101  
LaunchCustomDlg() 63  
LaunchSetupDlg() 63  
license to redistribute files, limits of 54

## M

mapping file types to directories 102  
mask, install conduit 59  
Memo Pad 19, 29  
Memory Stick 73, 102  
MIME types 102  
MultiMediaCard (MMC) 73  
multithreading support 13

## N

names, install directory 60  
naming conventions  
    files 98  
    volumes 95  
notifiers  
    installing 67

## O

object model, COM 14  
Open() 96, 101

## P

PD<PIM>DbHHRecordAdapter objects. *See* PIM  
    database adapters  
PDB files  
    on expansion cards 98  
PDCondMgr 55, 67  
PDConduitInfo 55  
PDExpansionCardInfo 81  
PDExpansionManager 81  
PDHotSyncUtility  
    using 63  
PDInstall 58  
PDInstallConduit  
    using 65  
PDSystemCondMgr 55

---

- PDUserData
  - using 64
- PDVFSFileManager 80
- PDVFSManager 80
- PDVFSVolumeManager 80
- PIM database adapters 29
- plug and play slot driver 74
- PRC files
  - on expansion cards 98
- primary storage 72
- programming languages 5
- programming style xiii
- progress display, HotSync Manager 13
- PSDConduit.dll 10

## Q

queue, install file. *See* install directory names

## R

- RAM
  - expansion 72
- Read() 97
- ReadFeature() 86
- redistribute, permission to 54
- RefreshConduitInfo() 63
- registering conduits
  - COM-based 55
- Rename() 96, 101
- RestartHotSyncMgr() 63
- ROM
  - expansion 74

## S

- samples
  - COM Sync Suite 6
- schema databases
  - objects, COM Sync Suite 16
- SD slot driver 104
- secondary storage 73
- Secure Digital (SD) 73
- Seek() 97
- SetCommStatus() 64

- Size 97
- slot driver 77
  - default directory 104
  - plug and play 74
- slot folder, defined 61
- slot ID 77
- slot reference number 87
- slot-install folder, defined 61
- slots 73
  - and volumes 94
  - checking for a card 87
  - enumerating 87
  - referring to 77
  - volumes 93
- standard directories on expansion media 81
- standard EXE client 12
- start.prc 82
- StartHotSyncMgr() 63
- storage
  - primary 72
  - secondary 73
- Sync Manager
  - COM API 28
  - VFS Manager 78
- system information objects, COM Sync Suite 15

## T

- Tell method 97
- TerminateHotSyncMgr() 63
- testing your installer 68
- To Do List 19, 29
- TotalCapacity property 92
- troubleshooting
  - installer 69

## U

- uninstalling your conduit 67
- universal connector 74
- updating the HotSync Manager progress display 13
- UsedSpace property 92
- user data store
  - accessing 64



---

## V

VFAT 78

VFS Manager 79

- COM Sync objects 80

- custom calls 107

- debugging applications 108

- directory operations 99

- directory paths 101

- enumerating files 102

- file paths 97

- file system operations 95

- filenames 98

- PDVFSFileManager object 80

- PDVFSManager object 80

- PDVFSVolumeManager object 80

- Sync Manager 78

- verifying presence of 84

- volume operations 91

Virtual File System. *See* VFS Manager

Visual Basic 5

Visual Basic .NET

- developing a conduit with 48

Visual Basic for Applications 5

Visual C/C++ 5

volumes

- automatically mounted 92

- basic operations 91

- enumerating 88

- formatting 92

- hidden 92, 93

- matching to slots 94

- mounted 88

- mounting 92

- naming 95

- read-only 92

- size 92

- slots 93

- space used 92

- unmounting 92

## W

Write() 97

