palmsource™

Introduction to Conduit Development

Palm OS® Conduit Development Kit for Windows, Version 6.0.1

Written by Brent Gossett.
Technical assistance from Cole Goeppinger, Gerard Pallipuram, Atul Gupta, Thomas Butler, Jeffrey Shulman, and Robert Rhode.

Introduction to Conduit Development
Document Number 3023-005
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

Table of Contents

About This Document

This document introduces conduit development for the Windows platform with the Palm OS® Conduit Development Kit (CDK) from PalmSource, Inc.; it describes how conduits relate to the Palm OS platform, how they communicate with the HotSync® Manager, and how to choose an approach to conduit development. This document is recommended reading for developers new to conduits.

The sections in this introduction are:

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
Introduction to Conduit Development	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® Manager, and how to choose an approach to conduit development. Recommended reading for developers new to conduits.
C/C++ Sync Suite Companion	An overview of how C API-based conduits operate and how to develop them with the C/C++ Sync Suite.
C/C++ Sync Suite Reference	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.
COM Sync Suite Companion	An overview of how COM-based conduits operate and how to develop them with the COM Sync Suite.
COM Sync Suite Reference	A reference for the COM Sync Suite object hierarchy, detailing each object, method, and property.
Conduit Development Utilities Guide	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 chapters of this document.

- Chapter 1, "What's New in the Palm OS CDK." Summarizes the enhancements and new features available in the last few releases of the Palm OS® CDK.
- Chapter 2, "Conduits and the Palm OS Platform." Describes how different software and hardware components fit into the world of software development for the Palm OS platform, and how conduits fit with those components.
- Chapter 3, "Using HotSync Exchange." Introduces HotSync Exchange, which is an alternative to writing a conduit if all you need to do is to transfer a desktop file to and from the handheld as a Palm OS database.
- Chapter 4, "Introducing Conduits." Introduces basic conduit concepts, including the types of conduits, what conduits do, example synchronization logic they implement, and the principles developers need to adhere to when developing conduits.
- Chapter 5, "Introducing HotSync Manager." Describes HotSync Manager's basic settings and functions.
- Chapter 6, "Registering Conduits and Notifiers with HotSync Manager." Describes how conduits must be registered with HotSync Manager for Windows.
- Chapter 7, "Understanding the HotSync Process." Presents the HotSync process, step-by-step from the moment the user presses the HotSync button to backing up databases and logging.
- Chapter 8, "Palm OS Databases." Describes the types and layout of Palm OS databases.
- Chapter 9, "Understanding Trusted Desktops." Explains how the HotSync client on the handheld establishes "trust" with a desktop to synchronize and back up secure databases.
- Chapter 10, "Conduit Development." Describes the different development approaches and provides the background for deciding which development approach best meets your conduit's requirements.

- Appendix A, "Configuration Entries." Describes the PalmSource-defined conduit configuration entries that you can access with the Conduit Manager API.

Changes to This Document

This section describes the changes made in each version of this document, starting with 3023-003 for CDK 6.0.

- Document 3023-005 for CDK 6.0.1
- Document 3023-004 for CDK 6.0
- Document 3023-003 for CDK 6.0

Document 3023-005 for CDK 6.0.1

The significant changes are listed below:

- Added Chapter 1, "What's New in the Palm OS CDK," on page 1.
- Removed information about file link, because the file link feature has been removed from HotSync Manager 6.0.1.
- Updated "Adding Messages to the HotSync Log" on page 46 to describe HTML formatting added to the HotSync log in HotSync Manager 6.0.1.
- Added "Creating Secure Databases" on page 134.
- Added "Guidelines for Sharing Access to a Schema Database" on page 137.

Document 3023-004 for CDK 6.0

The significant changes are listed by chapter below:

- Chapter 8, "Palm OS Databases."
  - Added "Mutually Exclusive Database Characteristics" on page 116.
  - Added to "Uniquely Identifying Databases" on page 117 that schema, extended, and classic databases exist in disjoint namespaces.

  - Clarified that schema database name, table names, and column names should be valid SQL identifiers.

Document 3023-003 for CDK 6.0

The significant changes are listed by chapter below:

- Chapter 3, "Using HotSync Exchange." Added this chapter.
- Chapter 5, "Introducing HotSync Manager." Renamed this chapter.
  - Added "Support for Multiple Users" on page 64.
  - Added "Interfacing with Windows Desktop Applications" on page 70.
- Chapter 6, "Registering Conduits and Notifiers with HotSync Manager." Added this chapter.
- Chapter 7, "Understanding the HotSync Process."
  - Renamed and reorganized this chapter so that it describes the HotSync process in sequence. This chapter was previously titled "HotSync Manager and Conduits."
  - In "Running the Backup Conduit" on page 109, corrected an error that said the Backup conduit for Windows does not back up a database if its type is `'data'` and a conduit registered with the same creator ID has run. The type is `'DATA'`, not `'data'`.
  - In "Determining the Sync Mode" on page 93, added a description of how the sync mode is determined for schema databases.
- Chapter 8, "Palm OS Databases." Added this chapter to describe both schema and non-schema databases.
- Chapter 9, "Understanding Trusted Desktops." Added this chapter.
- Appendix A, "Configuration Entries." Moved this information from the *C/C++ Sync Suite Reference* and the *Conduit Development Utilities Guide*.
  - In Table A.1 on page 177, changed the `Directory`, `File`, `Remote` conduit configuration entries to optional. They had previously been documented as required by HotSync Manager versions earlier than 2.0 and optional for later

versions. No version of HotSync Manager has ever failed to run a conduit that did not specify these values.

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

Conventions Used in this Document

This guide uses the following typographical conventions:

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

1

What's New in the Palm OS CDK

This section summarizes the enhancements and new features available in the last few releases of the Palm OS® CDK.

In This Section

- What's New in CDK 6.0.1

  Learn about the new features and enhancements available since the release of CDK 6.0/6.0a.
- What's New in CDK 6.0/6.0a

  Learn about the new features and enhancements available since the release of CDK 4.03.
- What's New in CDK 4.03

  Learn about the new features and enhancements available since the release of CDK 4.02/4.02a.

What's New in CDK 6.0.1

The Palm OS Conduit Development Kit (CDK) version 6.0.1 allows developers to create conduits using Microsoft Visual C++ .NET 2003 and Microsoft Visual Basic .NET 2003.

CDK 6.0.1 contains:

- C/C++ Sync Suite for C API-based conduits
- COM Sync Suite for COM-based conduits

Supported Versions of HotSync Manager and Sync Manager

CDK version 6.0.1 fully supports conduit development for HotSync Manager version 6.0.1. Conduits created with CDK 6.0.1 work with earlier versions of HotSync Manager, but only if they don't make use of newer features—for example, accessing a classic database works but accessing a schema database does not work with HotSync Manager versions earlier than 6.0.

IMPORTANT: PalmSource strongly recommends that you test your conduit with all the versions of HotSync Manager that it must work with. Do not assume that if your conduit works with version 6.0.1, it will work with versions 4.x or 3.x.

Table 1.1 shows which versions of HotSync Manager correspond to which versions of the Sync Manager API. It also states which types of databases each Sync Manager version can access.

Table 1.1 Comparison of HotSync Manager and Sync Manager versions

HotSync Manager Version	Sync Manager API Version	Database Types Supported
3.x	2.2	Classic
4.x	2.3	Classic
6.0	2.4	Schema, extended, and classic
6.0.1	2.5	Schema, extended, and classic

To determine in what version other features are available, refer to the "Compatibility" section of each function description in the reference documentation or to "Revision History" on page 1105 in *C/C++ Sync Suite Reference* or to "Revision History" on page 591 in *COM Sync Suite Reference*.

Development System Requirements for CDK 6.0.1

To develop conduits with this CDK, your development system must be equipped with at least:

- Microsoft Visual Studio .NET (Visual C++ or Visual Basic) 2003 on Windows 2000 or XP.

  Visual Studio .NET 2002 and Visual Basic 6 have been verified to work, but they are not officially supported. Other COM-enabled development environments may be used with the COM Sync Suite, but they are not specifically supported by PalmSource, Inc.
- 55 MB of free disk space and the minimum amount of RAM required by your development environment.
- A Palm Powered™ handheld with cradle/cable and software for testing, or Palm OS® Simulator (see Chapter 8, "Synchronizing with Palm OS Simulator," on page 49 of the *Conduit Development Utilities Guide*).

Deployment System Requirements for Conduits Developed with CDK 6.0.1

PalmSource, Inc. supports the deployment of conduits built with this CDK on end users' systems that meet these requirements:

- Windows 98SE, ME, 2000, or XP
- HotSync Manager versions 6.0.1 and later, or earlier versions if no new features are used (see "Supported Versions of HotSync Manager and Sync Manager" on page 2 in *Introduction to Conduit Development*)

Redistributable COM Sync Libraries

Though HotSync Manager versions 6.0 and later include the COM Sync libraries required to run COM-based conduits, versions earlier than 6.0 do not. Therefore, you may redistribute the COM Sync libraries, but only to install them on systems running a version of HotSync Manager earlier than 6.0.

See "Files to Install" on page 53 in the *COM Sync Suite Companion*.

Registering COM-based Conduits for All Windows Users

CDK 6.0.1 adds support for registering COM-based conduits for all Windows users. The new `PDSystemCondMgr` object mirrors the existing `PDCondMgr` object, except that it registers a conduit for all Windows users rather than for only the current user.

NOTE: COM Sync does not provide methods to register and manage system-registered notifiers or install conduits.

See "User- and System-registered Conduits and Notifiers" on page 78 for more on how HotSync Manager treats user- vs system-registered conduits.

Opting Out of the Conduit/Application Requirement for COM-based Conduits

CDK 6.0.1 adds support for COM-based conduits to choose whether to opt out of the conduit/application requirement. 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. Make sure you are aware of the implications of using this feature, because it is easy to create an "orphaned" database on the handheld.

NOTE: In HotSync Manager versions 6.0 and later, the COM Sync module opts out of the conduit/application requirement for all COM-based conduits by default. That is, if a COM-based conduit does not handle the case where `GetConduitInfo()`'s *`infoType`* parameter is `ePDRunAlways`, then HotSync Manager runs the conduit regardless of whether a matching application is on the handheld. Note that this behavior differs from the default behavior in versions earlier than 6.0.

Preventing Display of a COM-based Conduit in the Custom and Progress Dialog Boxes

CDK 6.0.1 adds support for COM-based conduits to prevent HotSync Manager from displaying a conduit's name in the **Custom** and **HotSync Progress** dialog boxes. To do so, a COM-based conduit responds to two new enum values passed to it via the `IPDClientNotify`.`GetConduitInfo()` method. Only HotSync Manager versions 6.0 and later pass in these new values.

Backing Up a Database with a Single Call

Sync Manager API version 2.5 adds a call that allows you to back up a database:

- `SyncBackupDatabase()` in the C/C++ Sync Suite
- `PSDDatabaseUtilities.BackupDatabase()` in the COM Sync Suite

Calling an Application on a Palm OS Cobalt Handheld

Sync Manager API version 2.5 adds the ability to call an application on a Palm OS Cobalt handheld:

- `SyncCallDeviceApplication()` in the C/C++ Sync Suite
- `PSDDatabaseUtilities.CallDeviceApplication()` in the COM Sync Suite

These calls work only with a Palm OS Cobalt handheld. For Palm OS Garnet or earlier handhelds, you must use continue to use `SyncCallRemoteModule()` or `CallRemoteModule()`.

Exporting/Importing Databases to/from an Expansion Card on Palm OS Cobalt Handhelds

VFS Manager API, version 1.1 adds two calls to enable conduits to export/import a database image to/from an expansion card on handhelds running Palm OS Cobalt, versions 6.0.1 or later. These functions can handle all database types (schema, extended, and classic).

- `VFSImportDatabaseFromFileEx()`
- `VFSExportDatabaseToFileEx()`

The existing versions of these calls (`VFSExportDatabaseToFile()` and `VFSImportDatabaseFromFile()`) are deprecated for use with Palm OS Cobalt handhelds.

In CDK 6.0.1, these new functions are available only in the C/C++ Sync Suite.

New COM Sync Suite Methods

In CDK 6.0.1, the COM Sync Suite adds the `PSDDatabaseUtilities` object. This object includes the following new methods:

- `BackupDatabase()`
- `CallDeviceApplication()`
- `InstallAndBackupDatabase()`
- `IsDatabaseBackupNeeded()`

For convenience, this object includes several other methods already provided in `PSDDatabaseQuery`.

HotSync Log Support for HTML

HotSync Manager 6.0.1 stores the HotSync log on the desktop as an HTML file rather than as an ASCII text file. This allows HotSync Manager to format a log entry depending on the "activity" value that it or a conduit specifies. By specifying a special activity value, a conduit can pass in HTML code for custom formatting, hypertext links, or scripts.

For more information, see "Adding Messages to the HotSync Log" on page 46 in *Introduction to Conduit Development*.

C++ Generic Conduit Framework

CDK 6.0.1 adds the following C++ Generic Conduit Framework

- The Classic C++ Generic Conduit Framework. This is the same as that provided in CDK 4.03, but was missing from CDK 6.0.

  ```
  Palm OS CDK\C++\Win\Samples\GenericConduit
  \Classic
  ```

- A ported version of the classic Memo Pad application that uses an extended database rather than a classic database. To help you experiment with the Extended C++ Generic Conduit Framework, you can use this application to sync with.

  `Palm OS CDK\C++\Win\Samples\GenericConduit\SamplePrc`

New CDK Installation Folder Name

The CDK 6.0.1 installer uses a new default installation path:

```
C:\Program Files\PalmSource\Palm OS CDK
```

Note that you can install the CDK to another location, but you cannot change the CDK directory name from `Palm OS CDK`.

Discontinued File Linking Feature

File linking was deprecated in HotSync Manager 6.0 and is discontinued in HotSync Manager 6.0.1. As a result, the following changes in have been made to HotSync Manager and CDK 6.0.1:

- The file link feature is no longer present. HotSync Manager no longer calls the file linking entry points of any conduit: `ImportData()`,`ConfigureSubscription()`, `SubscriptionSupported()`, and `UpdateTables()`. Also, the **File Link** item on the HotSync Manager menu is no longer present.
- `Subscribe.h` is no longer in CDK 6.0.1. This header declared the File Link API.
- All APIs that call the file link user interface are deprecated. These include `HsDisplayFileLink()` and `PDHotSyncUtility`.`LaunchFileLinkDlg()`.

See Also "Changes in C/C++ Sync Suite 6.0.1" on page 1106 in *C/C++ Sync Suite Reference*, "Changes in COM Sync Suite 6.0.1" on page 591 in *COM Sync Suite Reference*

What's New in CDK 6.0/6.0a

The Palm OS Conduit Development Kit (CDK) version 6.0/6.0a allows developers to create conduits using Microsoft Visual C++ .NET 2002 and 2003 and Microsoft Visual Basic 6.0/.NET.

CDK 6.0/6.0a contains:

- C/C++ Sync Suite for C API-based conduits
- COM Sync Suite for COM-based conduits

Development System Requirements for CDK 6.0/6.0a

Table 1.2 is a complete list of supported development environments and the platforms on which you can use each sync suite provided in CDK 6.0/6.0a.

Table 1.2 CDK 6.0/6.0a supported development environments and platforms

Development Environment and Platforms	C/C++ Sync Suite	COM Sync Suite
Microsoft Visual Studio .NET (Visual C++ and Visual Basic) 2002 and 2003: Windows 2000 and XP	Yes	Yes
Microsoft Visual Basic, version 6: Windows 2000 and XP	N/A	Yes

Deployment System Requirements for Conduits Developed with CDK 6.0/6.0a

CDK 6.0/6.0a supports deployment of conduits on end-user machines running Microsoft Windows 98SE, ME, 2000, XP.

Sync Manager Support for Extended and Schema Databases on Palm OS Cobalt Handhelds

The header files for the Sync Manager API have been reorganized to make it easier for you to include only those functions that access the type of database your conduit synchronizes. Sync Manager version 2.4 consists of three sets of APIs and headers for accessing different types of databases:

- `SyncMgr.h` for accessing **classic database**s. Native on Palm OS <= 4, and running in PACE on **Palm OS Garnet** and **Palm OS Cobalt**.
- `SyncDm.h` for accessing **extended database**s (similar to classic, but record size is not limited to 64 KB). Palm OS Cobalt only.
- `SyncDb.h` for accessing **schema database**s (new self-describing row-based database with many new features). Palm OS Cobalt only.

There is also a new `SyncCommon.h` header, which declares APIs and defines that are either common to all types of databases or are utility functions that do not depend on database type.

For more details on these and other Sync Manager API changes, see "Sync Manager API, Version 2.4 Changes" on page 1110 in the *C/C++ Sync Suite Reference*.

New Sync Mode for Schema Databases

HotSync Manager 6.0 supports a new synchronization mode specifically for schema databases: fast sync after a purge. This is the same as a fast sync, except that the schema Sync Manager can determine that instances of a **sync atoms** that were deleted on the handheld have been purged since the last HotSync operation with the current desktop. All change flags are valid, so a conduit needs only to read the ID lists of the sync atoms that were purged and compare them with those on the desktop to identify the purged instances of each atom.

Because of the Data Manager on the handheld tracks more detailed change information for schema databases than for non-schema databases, a conduit must determine the sync mode differently for schema databases. Don't use the `CSyncProperties::m_SyncType` in `OpenConduit()` (or the

`PDHotsyncInfo`.`SyncType` property in COM Sync) to determine the sync mode. Instead call `SyncDbGetSyncMode()` (or `PSDDatabaseAdapter`.`GetSyncTypeInfo()`) for each schema database that your conduit synchronizes.

When finishing a schema database sync, make sure to call `SyncDbCloseDatabase()` with the `SYNC_CLOSE_DB_OPT_SYNCED_ALL_CHANGES` flag (or call the `PSDDatabaseQuery`.`CloseDatabase()` with the *`bSeenAllChanges`* parameter) when closing the database. If you do not, any modified rows remain modified for the next sync.

Registering C API-based Conduits for All Windows Users

HotSync Manager version 6.0 supports **system conduits**, which run for all Windows users, not just the current Windows user. Conduits that run for only the current user (like all previous conduits) are now called **user conduits**. Creator ID uniqueness is still required for conduits, but only at the same level of registration. If two user or system conduits register for the same creator ID, it is still a conflict; but if a single user and a single system conduit register for the same creator ID, the user conduit runs. This allows different conduit configurations for different Windows users.

COM-based conduits can be registered as system conduits, but only by using the CondCfg utility or the Conduit Manager API provided in the C/C++ Sync Suite; the `PDCondMgr` object in COM Sync can register only user conduits.

Folder-based Conduit Registration

HotSync Manager version 6.0 supports a new additional method of conduit registration called **folder-based conduit registration**, which doesn't require any API to be called. Instead of passing the conduit's registration information (its creator ID, display title, etc.) into an API, HotSync Manager queries the conduit for this information using the `GetConduitInfo()` entry point. HotSync Manager discovers these conduits by looking in special folders, one system folder for all Windows users, and one user folder unique for each Windows user.

Folder-based conduit registration is currently supported only for C API-based conduits; it is not supported for COM-based conduits.

Opting Out of the Conduit/Application Requirement

HotSync Manager version 6.0 allows 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 if a C API-based conduit responds to the `GetConduitInfo()` entry point correctly, HotSync Manager runs the conduit even without a corresponding application on the handheld. Make sure you are aware of the implications of using this feature, because it is easy to create an "orphaned" database on the handheld.

NOTE: The COM Sync module in HotSync Manager 6.0 opts out of the conduit/application requirement for all COM-based conduits; no explicit action is required of a COM-based conduit. This is a change in the behavior compared to previous releases.

Preventing Display of a C API-based Conduit in the Custom and Progress Dialog Boxes

HotSync Manager version 6.0 allows a conduit to prevent its name from being listed in the **Custom** dialog box and in the **HotSync Progress** dialog box. To do so, a conduit responds to two new enum values passed to it via the `GetConduitInfo()` entry point.

This is feature is not available for COM-based conduits, which are always displayed in these dialog boxes.

COM Sync Libraries Installed with HotSync Manager 6.0

The COM Sync libraries required for COM-based conduits to run are now installed as part of HotSync Manager version 6.0 or later. No mini-installer is required for COM-based conduit installers as in previous releases, unless your conduit must be installed on machines running an earlier version of HotSync Manager. You can still get the mini-installer from CDK 4.03.

Handheld's Palm OS Version Stored on the Desktop

HotSync Manager version 6.0 stores the Palm OS version of the user's handheld in the **user data store** on the desktop. This enables a desktop application to retrieve this information *outside* of a HotSync operation using the User Data API.

In C, call `UmGetInteger()` and specify the section/key names defined in "Palm OS Version Section and Key Names" on page 1013 in the *C/C++ Sync Suite Reference*.

In COM, call `PDUserData`.`GetIntegerValue()` and specify the same section/key names.

Previously, only a conduit could get the version number *during* a HotSync operation by using `SyncReadSystemInfo()` or `PDSystemAdapter`.`RomSoftwareVersion`. Now you can retrieve the it both ways.

Extended C++ Generic Conduit Framework

The C/C++ Sync Suite includes the Extended C++ Generic Conduit sample. It is not tested and is not a supported sample. Please use it for experimentation purposes only.

Documentation in Integrated in Visual Studio .Net

CDK documentation is now fully integrated into Visual Studio .NET. Click **Help** > **Contents** to view the "Palm OS Conduit Development Kit 6.0" collection. The documentation also supports Visual Studio .NET's Dynamic Help and F1 help features: type or select a CDK API in the code editor and the relevant topic appears in the Dynamic Help window; just click the topic or press F1.

If you are still using Visual Studio 6, the CDK also includes the same documents in HTML Help format for your convenience. From the Start menu, click **Programs** > **PalmSource** > **Conduit Development Kit** > **Documentation** > **Palm OS CDK 6**.

Discontinued Palm Utility ActiveX Control (PalmCntl.ocx)

`PalmCntl.ocx` was deprecated in CDK 4.03 and is no longer provided in CDK 6.0/6.0a. Starting in CDK 4.03, the same functionality is provided in these COM Sync objects: `PDCondMgr`, `PDConduitInfo`, `PDInstallConduit`, `PDInstallConduitInfo`, `PDUserData`, `PDHotSyncUtility`, and `PDInstall`.

Deprecated File Linking Feature

In HotSync Manager version 6.0, file linking is deprecated. Conduits that implement file linking with any database type (schema, extended, or classic) continue to function normally, but the limitations on categories remain in effect: only 16 categories with names limited to 15 characters. Note that in certain upgrade scenarios, records in a file-link category are duplicated.

See Also "Changes in C/C++ Sync Suite 6.0" on page 1110 in *C/C++ Sync Suite Reference*, "Changes in COM Sync Suite 6.0" on page 593 in *COM Sync Suite Reference*

What's New in CDK 4.03

The Palm OS Conduit Development Kit (CDK) version 4.03 allows developers to create conduits using Microsoft Visual C++ 6.x/.NET, Microsoft Visual Basic 6.0/.NET, and WebGain VisualCafe 4.0/4.0a.

CDK 4.03 contains:

- C/C++ Sync Suite for C API-based conduits
- COM Sync Suite for COM-based conduits
- JSync Suite for Java-based conduits

Development System Requirements for CDK 4.03

CDK 4.03 supports Visual Studio .NET (version 7) and includes a new Conduit Wizard redesigned for Visual Studio .NET. (The Conduit Wizard is no longer available for Visual C++ 6.x.)

Table 1.3 is a complete list of supported development environments and the platforms on which you can use each sync suite provided in CDK 4.03.

Table 1.3 CDK 4.03 supported development environments and platforms

Development Environment and Platforms	C/C++ Sync Suite	COM Sync Suite	JSync Suite
Microsoft Visual Studio .NET (Visual C++ and Visual Basic, version 7): Windows NT 4.0 x86, 2000	Yes	Yes	N/A
Microsoft Visual C++,[1] version 6: Windows 98SE, ME, NT 4.0 x86, 2000	Yes	Yes	No
Microsoft Visual Basic, version 6: Windows 98SE, ME, NT 4.0 x86, 2000	N/A	Yes	N/A
WebGain VisualCafe, version 4.0 and 4.0a: Windows 98SE, ME, NT 4.0 x86, 2000	No	No	Yes

1. The Conduit Wizard is not available for Visual C++ version 6.

Deployment System Requirements for Conduits Developed with CDK 4.03

CDK 4.03 supports deployment of conduits on end-user machines running Microsoft Windows 98SE, ME, NT 4.0 x86, or 2000.

COM Sync Suite Support for Registering Conduits, Installing Applications, and Accessing Expansion Cards

CDK 4.03 adds the following objects to the COM Sync Suite.

Table 1.4 Conduit registration, application installation, and support objects

Object	Description
`PDCondMgr`	Methods to register a conduit during installation.
`PDConduitInfo`	Properties of a conduit required to register it with HotSync Manager.
`PDInstallConduit`	Methods to register an install conduit during installation.

Table 1.4 Conduit registration, application installation, and support objects *(continued)*

Object	Description
`PDInstallConduitInfo`	Properties of an install conduit required to register it with HotSync Manager.
`PDHotSyncUtility`	Methods for controlling the HotSync Manager application.
`PDInstall`	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.
`PDUserData`	Methods for accessing the users data store on the desktop computer.

Table 1.5 Expansion card objects

Object	Description
`PDExpansionManager`	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.
`PDExpansionCardInfo`	Provides information about an expansion card: media type, product name, manufacturer name, and so on.
`PDVFSManager`	Represents a file system on an expansion card.
`PDVFSVolumeManager`	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.
`PDVFSFileManager`	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 Sync Suite Support for PIM Databases

CDK 4.03 adds the following objects to the COM Sync Suite to enable you to access the classic PIM application databases more easily.

Table 1.6 Classic database objects

Object	Description
`PDAddressDbHHRecordAdapter`	Methods and properties for access to a single open Address Book database. Use `PDDatabaseQuery` to obtain this object.
`PDDateBookDbHHRecordAdapter`	Methods and properties for access to a single open Date Book database. Use `PDDatabaseQuery` to obtain this object.
`PDMemoDbHHRecordAdapter`	Methods and properties for access to a single open Memo Pad database. Use `PDDatabaseQuery` to obtain this object.
`PDTodoDbHHRecordAdapter`	Methods and properties for access to a single open To Do List database. Use `PDDatabaseQuery` to obtain this object.
`PDAddressDbHHRecord`	Represents the structure of a record in the Address Book database on the handheld.
`PDDateBookDbHHRecord`	Represents the structure of a record in the Date Book database on the handheld.
`PDMemoDbHHRecord`	Represents the structure of a record in the Memo Pad database on the handheld.
`PDTodoDbHHRecord`	Represents the structure of a record in the To Do List database on the handheld.

"MFC-less" HotSync Manager

This release of the CDK includes only one version of the HotSync Manager binaries; it is an "MFC-less" version also capable of sending messages to Conduit Inspector. Because it does not use MFC, this version of HotSync Manager works with conduits compiled in Release or Debug mode. When you are debugging conduits, you do not need to run a special debug version of the HotSync Manager as in CDKs before version 4.02.

Deprecated Palm Utility ActiveX Control (PalmCntl.ocx)

Though `PalmCntl.ocx` is still provided and supported in CDK 4.03, it may not be in later CDKs. Starting in CDK 4.03, the same functionality is provided in these COM Sync objects: `PDCondMgr`, `PDConduitInfo`, `PDInstallConduit`, `PDInstallConduitInfo`, `PDUserData`, `PDHotSyncUtility`, and `PDInstall`. PalmSource highly recommends that you use these instead.

Discontinued Support for Palm MFC Conduit Framework

Previous releases of the CDK provided the Palm MFC Conduit Framework (`CBaseMon`, `CBaseTable`, and so on) for developing a conduit. Support for building these types of conduits has been discontinued in CDK 4.03. The Generic Conduit Framework provides more flexibility in developing conduits and will continue to be the model sync logic for conduits. The Palm Desktop software will continue to include the necessary library files (`table2x.dll` and `pdcmn2x.dll`) to run existing conduits based on the MFC Conduit Framework. However, PalmSource encourages all developers to migrate to the Generic Conduit Framework.

See Also "Changes in C/C++ Sync Suite 4.03" on page 1118 in *C/C++ Sync Suite Reference*, "Changes in COM Sync Suite 4.03" on page 595 in *COM Sync Suite Reference*

2

Conduits and the Palm OS Platform

This chapter describes software development for the Palm OS® platform and how conduits work with Palm OS.

The sections in this chapter are:

Basic Operations

Using a Palm Powered™ handheld connected to a desktop computer, you can perform the following operations:

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

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

- portable data entry
- portable data viewing
- remote transactions

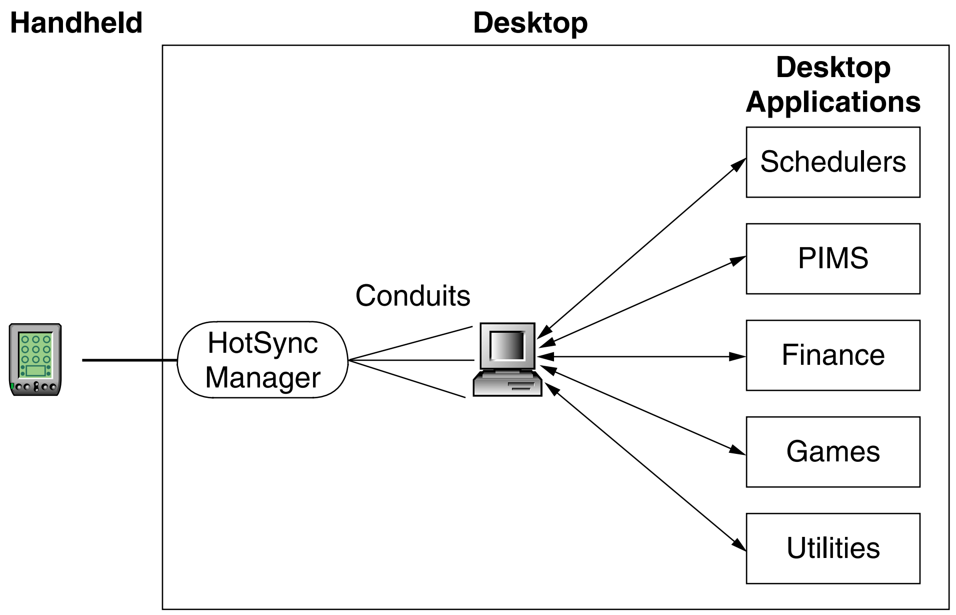
Applications that perform the following operations are considered more suitable for the desktop computer's processing power:

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

Component Relationships

Figure 2.1 provides a simplified view of the relationship among the Palm OS platform components.

Figure 2.1 Palm OS platform simplified view



HotSync® Manager is a desktop application that manages communications with the handheld. This application is described in Chapter 5, "Introducing HotSync Manager," on page 57. HotSync Manager uses a communications API, called the Sync Manager API, to handle the actual sending and receiving of bytes to and from the handheld, which makes HotSync Manager and the Sync Manager API independent of the connection type.

Typical communication connection types include:

- direct serial cable connection
- infrared connection
- modem connection
- network connection

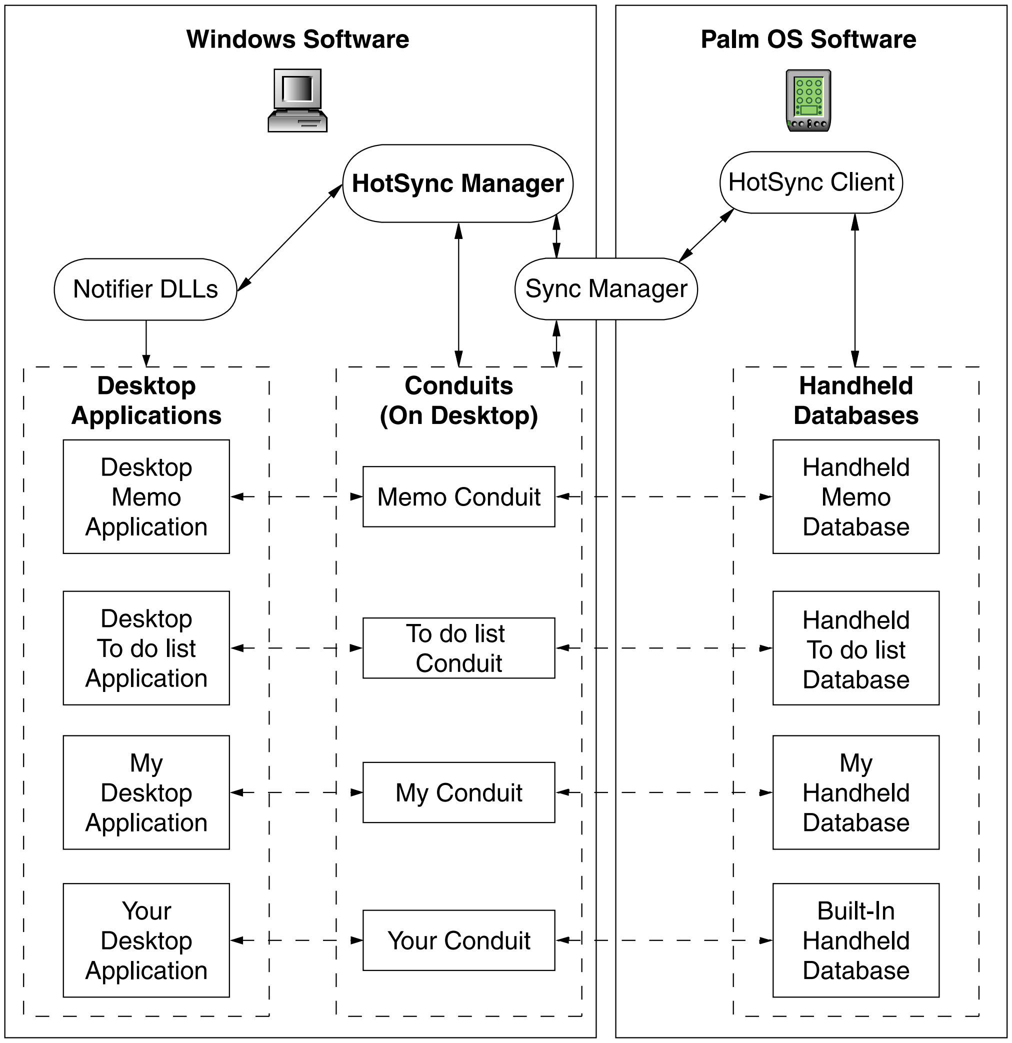
Software Components

The following are the software components of the Palm OS platform that enable you to synchronize desktop and handheld data:

- **Desktop applications**, which are developed by you or another developer, run on a desktop computer and operate on data that is sent to or retrieved from a handheld.
- **Palm OS applications** are developed to run on any Palm Powered handheld. These are also referred to as **handheld applications**.
- The **HotSync Manager** application runs on a desktop computer and communicates with a handheld. When the user presses the HotSync button, HotSync Manager awakens and calls each of the conduits that are properly installed and configured on the user's desktop computer.
- The **HotSync client** launches on the handheld when the user presses the HotSync button on the cradle. This handheld application wakes up HotSync Manager and responds to Sync Manager requests to access databases on the handheld.
- **Conduits** are modules that plug into HotSync Manager, which calls each in turn to read and write data on both the handheld and desktop. You use the Conduit Development Kit to create conduits. Generally, when you develop a handheld application that shares data with a desktop application, you also develop a conduit to synchronize the data between the two.
- The **Sync Manager API** provides a programmatic interface that conduits use for communicating with a handheld. This communications API allows the conduit to remain independent of the connection type between a handheld and a desktop computer.
- **Notifiers** are called by HotSync Manager to tell programs on the desktop computer that HotSync Manager is running. A notifier is a Windows DLL that you create to tell your desktop application that your conduit is modifying its desktop data. This behavior ensures that the application and its associated conduit are not both changing data at the same time. Note that notifiers are called only by the Windows version of HotSync Manager.

Figure 2.2 shows the process flow through these components.

Figure 2.2 Palm OS platform process flow



Desktop Applications

Desktop applications are standard desktop computer applications that either share data with a handheld application or contain a data entry component that runs on the handheld. For example, the Palm OS® Desktop software shares Date Book, Address Book, Memo Pad, and To Do List data with the built-in handheld applications.

Handheld Applications

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

Developers create applications for Palm Powered handhelds using the development environments described in the documentation that ships with the Palm OS Software Development Kit (SDK).

HotSync Manager

HotSync Manager oversees synchronization of a handheld on the desktop computer. HotSync Manager runs in the background on the user's computer and monitors one or more communication ports for the signal to begin synchronization. When HotSync Manager receives that signal, HotSync Manager initiates the synchronization process and calls the conduits installed on the user's desktop computer.

NOTE: HotSync Manager launches only when it receives a signal from a handheld. PalmSource does not support any method of starting a HotSync operation from the desktop.

During a HotSync operation, each conduit performs its own synchronization operations. To ensure that the entire operation does not take too much time, it is fundamental to the philosophy of the the Palm OS platform that each conduit perform its tasks as quickly as possible.

For more information about HotSync Manager, see Chapter 5, "Introducing HotSync Manager," on page 57.

HotSync Client

The HotSync client is a built-in Palm OS application on the handheld that launches when a user presses the HotSync button on the cradle. The client wakes up HotSync Manager on the desktop computer. HotSync Manager communicates with the HotSync client to set up the properties of the synchronization operation. From the HotSync client, the user selects the connection type (local or modem) and its properties (direct serial, infrared, modem type, and so on). During a HotSync operation, the HotSync client responds to Sync Manager requests to access databases on the handheld.

Conduits

A conduit is a plug-in module for HotSync Manager that transfers a specific kind of data between the handheld and the desktop computer. Each conduit should require no user interaction.

IMPORTANT: Conduits should operate without user interaction so that users can press the HotSync button on the cradle and have data synchronized without any required intervention. This is especially important for users who are synchronizing remotely (for example, over a network) and cannot interact with the desktop computer.

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

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

Notifiers

When both an application on the desktop computer and its corresponding conduit can modify user data, HotSync Manager uses notification to ensure that both are not changing data at the same time. HotSync Manager calls a notifier with a message for the desktop application, and the notifier calls the application, passing the message in a format that the application can understand. Conduit developers create their own notifiers to communicate with their desktop applications. For more information about notifiers, see Chapter 8, "Writing a Desktop Notifier," on page 93 in the *C/C++ Sync Suite Companion*.

Conduit Development Kit

To enable developers to create conduits, PalmSource, Inc. offers the Conduit Development Kit (CDK) for Windows, which consists of the following suites:

- **C/C++ Sync Suite**—the components of the CDK for Windows used to create a **C API-based conduit**. Includes APIs, the C++ Generic Conduit Framework, C libraries, samples, documentation, and utilities.
- **COM Sync Suite**—the components of the CDK for Windows used to create a **COM-based conduit**. Includes a COM object model (including a COM interface specification so that HotSync Manager can call your conduit), samples, documents, utilities, and the COM Sync Installer for deploying the required COM Sync components on end users' computers.

You can use these Sync Suites to develop conduits for all handhelds running Palm OS. See Chapter 1, "What's New in the Palm OS CDK," on page 1 in the *Introduction to Conduit Development* for details.

3

Using HotSync Exchange

This chapter introduces the HotSync® Exchange feature available in HotSync Manager versions 6.0 and later and Palm OS® Cobalt. HotSync Exchange enables a handheld and a desktop computer to use Palm OS standard interfaces to exchange files in their native formats.

IMPORTANT: If your conduit design only converts between handheld and desktop data formats, then you may be able to use HotSync Exchange and not need to develop a conduit at all.

The sections in this chapter are:

Overview

HotSync Exchange enables a handheld and a desktop computer to use Palm OS standard interfaces to exchange files in their native formats. For example, HotSync Exchange enables the user to install bitmap (BMP) files to a handheld image viewer application via the standard Palm OS Desktop software's Install Tool application, provided the viewer has registered with the handheld Exchange Manager for the BMP extension. This feature eliminates the need for a custom conduit that simply packs bitmap data into Palm OS database (PDB) format. During the HotSync operation following the bitmap installation, the handheld viewer receives the bitmap data from HotSync Exchange using the Palm OS standard Exchange

Manager interface—the same interface that it would use to receive beamed data. This eliminates the need for the application to discover and unwrap the PDB file containing the bitmap. The handheld viewer can also send files via the Exchange Manager directly to the desktop, where HotSync Exchange stores them in the standard bitmap (BMP) format.

HotSync Exchange consists of three components:

- An install conduit shipped with HotSync Manager versions 6.0 and later. This install conduit is named "HotSync Exchange" in the HotSync Manager **Custom** dialog box.
- A handheld Exchange library shipped in Palm OS Cobalt. This library is called the "HotSync Exchange library" in the rest of this chapter.
- A **File Exchange** > **Edit Queue** menu item in the HotSync client in Palm OS Cobalt. The user can look at the File Exchange Queue on the handheld to see which files on the handheld will be transferred during the next HotSync operation. The user can also direct the file to go to a selected desktop.

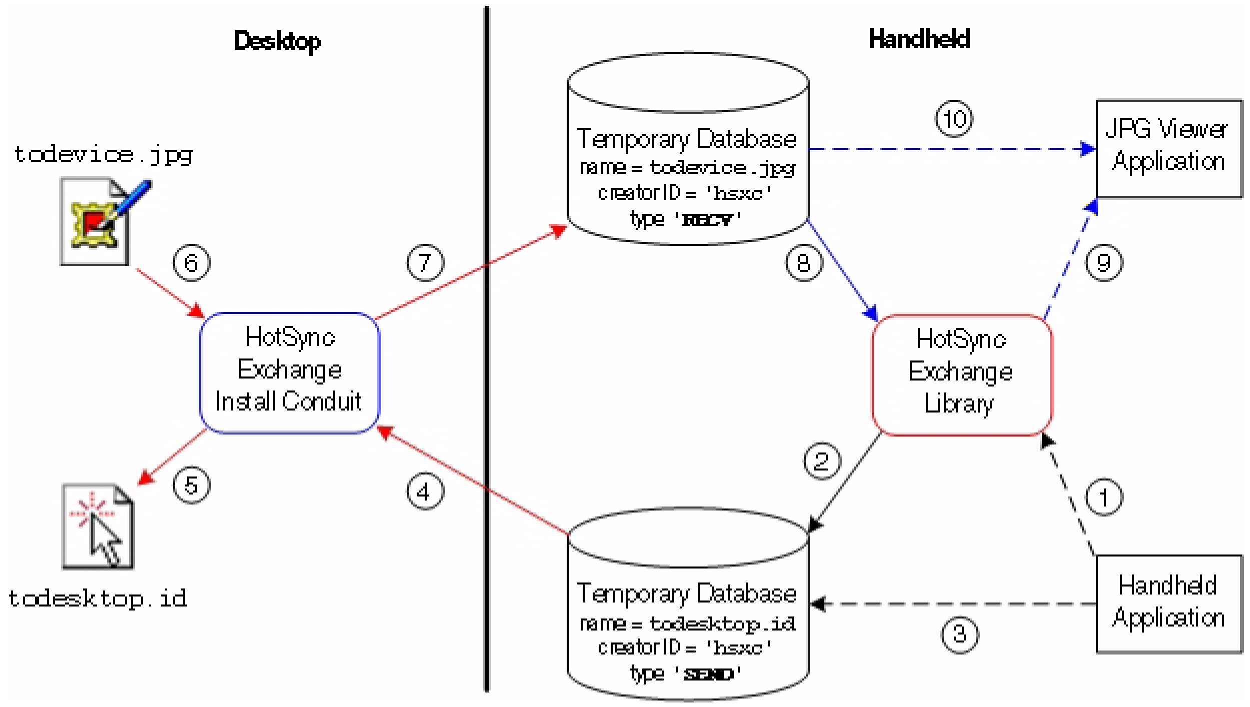
HotSync Exchange offers developers the following advantages over custom conduit development:

- Eliminates the need to convert between Palm OS and desktop file formats.
- Eliminates the need for conduits that simply translate file formats.
- Relies on the existing Exchange Manager interface on the handheld.
- Handles standard data formats:
  - Palm OS application registers for a MIME-type/file extension using `ExgRegisterDataType()`.
  - Desktop file type is determined by the file extension specified in the URL passed to `ExgPut()`—for example, URL: `_desktop://HSuser/images/flowers.jpg` specifies a JPEG file.

Flow of Data during a HotSync Exchange

This section presents an example that explains the flow of data before, during, and after a HotSync operation in which a HotSync Exchange occurs. In this example, a handheld application queues a file named `todesktop.id` to be transferred to the desktop and the user queues a file named `todevice.jpg` to be transferred to the handheld. Assume that an application on the handheld is registered with Exchange Manager for the JPG file type. Figure 3.1 illustrates the flow of data in the example described below.

Figure 3.1 HotSync Exchange data flow


Desktop
Handheld
todevice.jpg
todesktop.id
HotSync Exchange Install Conduit
HotSync Exchange Library
Temporary Database name = todevice.jpg creatorID = 'hsxc' type 'RECV'
Temporary Database name = todesktop.id creatorID = 'hsxc' type 'SEND'
JPG Viewer Application
Handheld Application
1
2
3
4
5
6
7
8
9
10

Before a HotSync Operation

On the desktop, the user uses Install Tool to queue `todevice.jpg` for installation during the next HotSync operation. Install Tool in turn calls the Install Aide function `PltInstallFile()`, which directs the file to the HotSync Exchange install conduit by placing it specific download folder on the desktop.

On the handheld (depicted by the black lines numbered 1, 2, and 3 in Figure 3.1), an application uses the standard Palm OS Exchange Manager API to perform the following steps:

1. Initialize the `ExgSocketType` structure with a filename of `todesktop.id`.
2. Call `ExgPut()`.

   This call causes the HotSync Exchange library to create a new database with creator ID `'hsxc'`, type `'SEND'`, and a unique name. The library stores the database name, file name, and default destination (`"<Any Desktop>"`) in a catalog of pending desktop exchanges.
3. Call `ExgSend()` to fill the database with data.
4. Call `ExgDisconnect()` to close the database. The new database is queued until the data in it can be successfully sent to the desktop during a HotSync operation.

During a HotSync Operation

As indicated by the red lines numbered 4, 5, 6, and 7 in Figure 3.1 on page 29, the HotSync Exchange install conduit first checks whether any handheld-to-desktop transfer requests are pending for the local desktop by examining the handheld exchange catalog mentioned above. Because it finds an entry in the catalog with a destination of `"<Any Desktop>"`, the HotSync Exchange conduit creates a desktop file with the specified filename (`todesktop.id`) in the `Exchange\From` subdirectory of the HotSync user's data directory. The HotSync Exchange conduit transfers the contents of the temporary handheld database, less the header, into this file and then deletes the catalog entry and the temporary database on the handheld.

Finally, the HotSync Exchange install conduit checks whether any files need to be transferred to the handheld. In this example, it finds the file `todevice.jpg` queued on the desktop. The HotSync Exchange conduit creates a database on the handheld with creator ID `'hsxc'`, type `'RECV'`, and name `todevice.jpg`.

After a HotSync Operation

As indicated by blue lines numbered 8, 9, and 10 in Figure 3.1 on page 29, Palm OS sublaunches newly installed applications and applications whose data has been modified by the HotSync operation. At this point, newly installed applications can register the file types they support with the Exchange Manager.

NOTE: To enable users to install a file to either the handheld's primary storage or to an expansion card, your handheld application must register the file type with *both* the Exchange Manager (to install to primary storage) and the VFS Manager (to install to an expansion card). The VFS Manager registers several common desktop file types itself and associates them with specific default directories on a card in which files of these types are installed during a HotSync operation. If your file type is not one of these, your application must register the file type and specify a new default directory on the card. Refer to *Exploring Palm OS: Memory, Databases, and Files* in the Palm OS SDK for details on registering new default directories on an expansion card.

For example, if the JPG viewer application were installed during this HotSync operation, it would need to register for the JPG file type now. When the HotSync Exchange library receives the `sysNotifySyncFinishEvent` notification, it searches for databases with creator ID `'hsxc'` and type `'RECV'`. Upon finding `todevice.jpg`, it opens this database and calls `ExgNotifyReceive()`. This call causes the Exchange Manager to launch the associated application (the JPG viewer) and to transfer `todevice.jpg` to it. At this time, the application can choose whether to the data in a native Palm OS database. When the application calls `ExgDisconnect()`, the HotSync Exchange library deletes the temporary database created by the HotSync Exchange conduit.

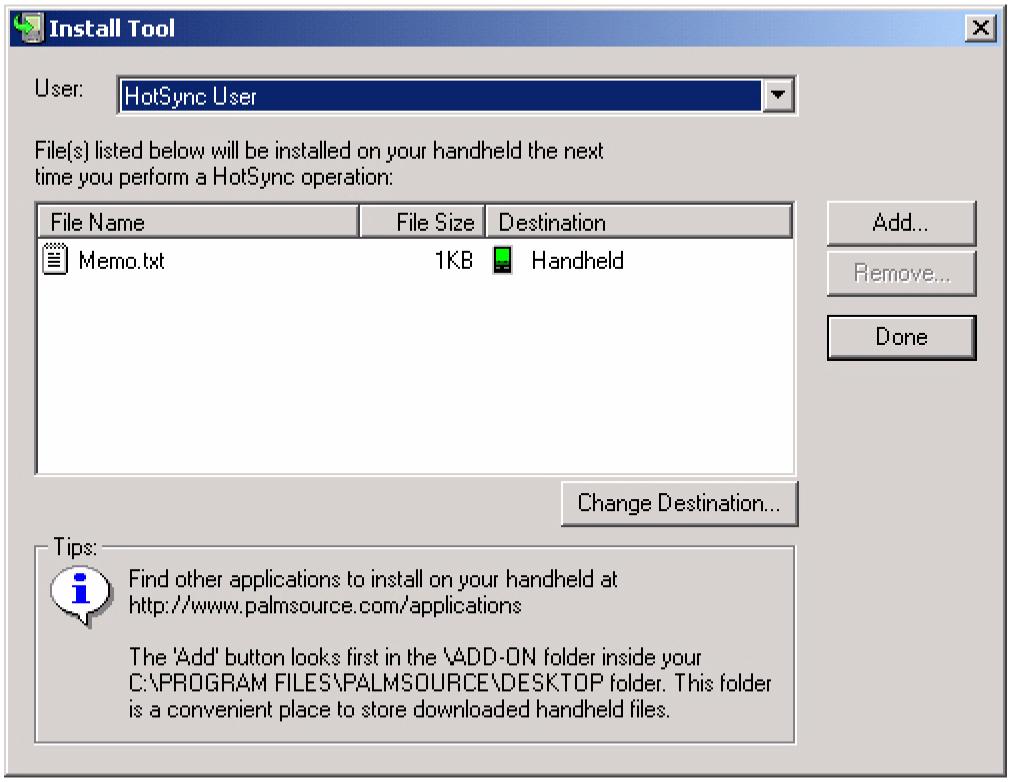
Desktop-to-Handheld HotSync Exchange

Using HotSync Exchange to transfer a file from the desktop to the handheld is similar to installing a Palm OS file type (PRC, PDB, etc.) on the handheld. The following sections describe the user interface and API that enables desktop applications or conduits to transfer files via HotSync Exchange.

Desktop User Interface

Install Tool (Figure 3.2) is the primary user interface for HotSync Exchange on the desktop. Users can launch Install Tool and add files to be installed during the next HotSync operation. If the filename extension is recognized by the HotSync Exchange Library on the handheld, the handheld accepts the file and the information is available to the user the next time an associated handheld application is run. If the extension is not recognized, the file is not installed on the handheld and a message is written to the HotSync log on the handheld, but not on the desktop.

Figure 3.2 Install Tool



User: HotSync User

File(s) listed below will be installed on your handheld the next time you perform a HotSync operation:

File Name	File Size	Destination
Memo.txt	1KB	Handheld

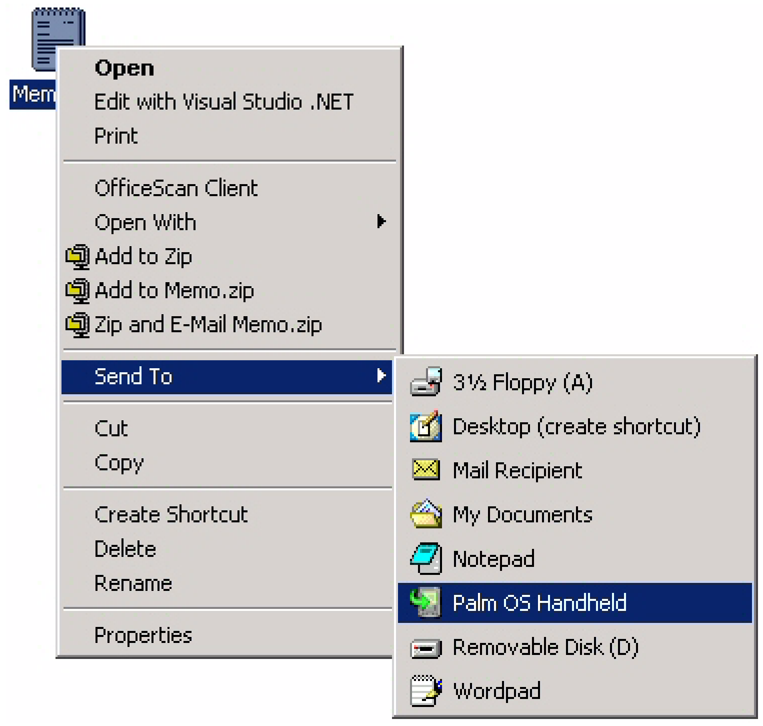
Tips:

Find other applications to install on your handheld at http://www.palmsource.com/applications

The 'Add' button looks first in the \ADD-ON folder inside your C:\PROGRAM FILES\PALMSOURCE\DESKTOP folder. This folder is a convenient place to store downloaded handheld files.

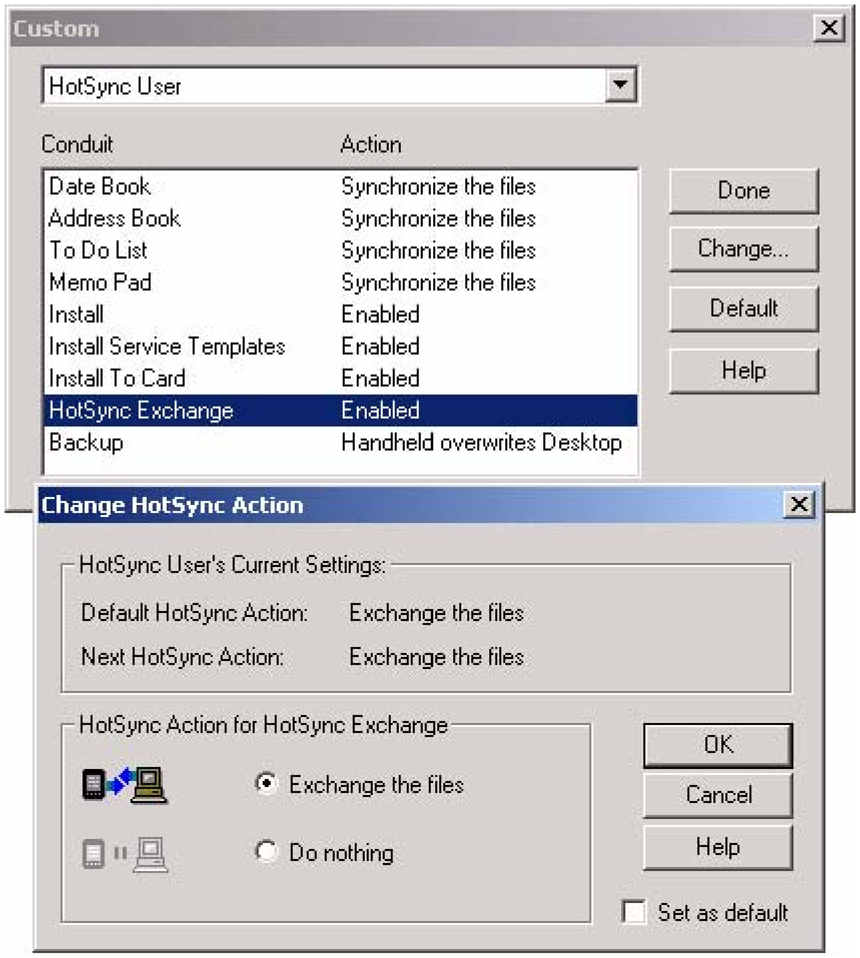
In addition to the Install Tool application, the user interacts with HotSync Exchange via a Send To item in the Windows Explorer context menu. This item is named “Palm OS Handheld” (Figure 3.3). Right-clicking a file and choosing this menu item passes the file path to the Install Tool, which prompts for the HotSync user for whom to install the file.

Figure 3.3 Context menu for HotSync Exchange and Install Tool



As with any install conduit, the user can enable and disable the HotSync Exchange install conduit via the HotSync Manager **Custom** dialog box as shown in Figure 3.4.

Figure 3.4 Configuring the HotSync Exchange install conduit



Conduit	Action
Date Book	Synchronize the files
Address Book	Synchronize the files
To Do List	Synchronize the files
Memo Pad	Synchronize the files
Install	Enabled
Install Service Templates	Enabled
Install To Card	Enabled
HotSync Exchange	Enabled
Backup	Handheld overwrites Desktop

HotSync User's Current Settings:

Default HotSync Action:	Exchange the files
Next HotSync Action:	Exchange the files

HotSync Action for HotSync Exchange

[✓ Exchange the files] Do nothing

☐ Set as default

The HotSync Exchange library disables the confirm receipt dialog that is normally displayed when data is sent via the Exchange Manager. Thus there is no user interface on the handheld during a successful desktop-to-handheld exchange.

Desktop HotSync Exchange API

There is no new API for HotSync Exchange. To transfer to the handheld's *primary storage* a file of a type registered with the Exchange Manager on the handheld, you call the same API as you do to install a file of any Palm OS file type (PRC, PDB, etc.) on the handheld:

- C/C++ Sync Suite: Install Aide, `PltInstallFile()`
- COM Sync Suite: `PDInstall`.`InstallFileToHH()`

If the file to install is not registered by another install conduit, the HotSync Exchange install conduit installs the file, if the handheld is running Palm OS Cobalt. Otherwise, the Install to Card conduit attempts to install the file on an expansion card, if present.

To transfer to the handheld's *expansion card* a file of a type registered with the VFS Manager on the handheld, you call one of these APIs:

- C/C++ Sync Suite: Install Aide, `PlmSlotInstallFile()`
- COM Sync Suite: `PDInstall`.`InstallFileToSlot()`

For more information on these APIs, see one of the following:

- "Using the Install Aide API" on page 128 in *C/C++ Sync Suite Companion*
- "Installing Files on the Handheld with PDInstall" on page 58 in *COM Sync Suite Companion*

Bundled Install of Application and Files

A bundled install transfers both an application and its data to the handheld in a single HotSync operation—for example, an image viewer application and sample JPG images. The following steps outline how a bundled install proceeds:

1. The HotSync operation begins.
2. The Install conduit installs PRC and PDB files—including, for example a JPG viewer application.
3. The HotSync Exchange install conduit transfers other files, like JPG files, to the Exchange Library on the handheld.
4. All applications (including any PRCs installed in step 2) receive the `sysAppLaunchCmdSyncNotify` launch code.

5. During sublaunch, the viewer application installed in step 2 must register for the JPG file type with the Exchange Manager.
6. The HotSync Exchange library receives a `sysNotifySyncFinishEvent` notification.
7. The HotSync Exchange Library launches the viewer application and transfers the JPG files to it.

Handheld-to-Desktop HotSync Exchange

Using HotSync Exchange to transfer a file from the handheld to the desktop is similar to transferring data between handheld applications using the Exchange Manager. The primary difference is that the data is queued and transferred during a HotSync operation with a desktop computer.

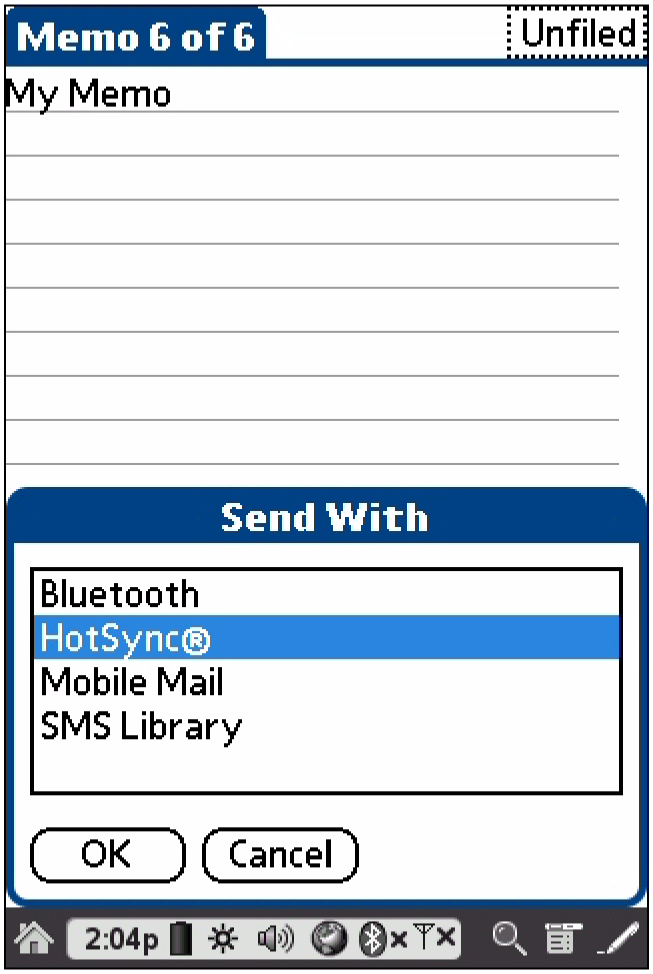
An application that supports HotSync Exchange allows the user to select an item and “send” it to the desktop via HotSync Exchange. The application calls the Exchange Manager, specifying the information to send and the file to create on the desktop in the user’s `Exchange\From` directory. The HotSync Exchange Library on the handheld creates a new database containing this information and queues it until it can be successfully transferred to a desktop computer during a HotSync operation.

The following sections describe the user interface and API that enables handheld applications to transfer files via HotSync Exchange.

Handheld User Interface

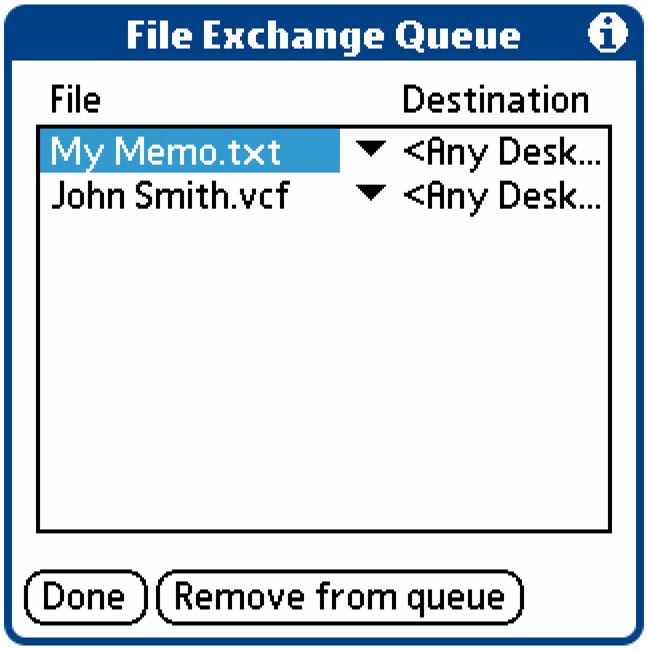
On the handheld, each application that supports HotSync Exchange must provide an interface whereby the user selects the information to send to the desktop. For example, Memo Pad provides a **Send Memo** menu item. Memo Pad calls the Exchange Manager, which presents a list of connection methods (Figure 3.5). Choosing the **HotSync** method queues the memo to be transferred via HotSync Exchange during a HotSync operation.

Figure 3.5 HotSync Exchange from Memo Pad



The HotSync client on the handheld provides a way for the user to remove items that are queued for transfer to the desktop (Figure 3.6). It also allows the user to select a target desktop. The file is then transferred during the next HotSync operation with the selected desktop.

Figure 3.6 HotSync client's File Exchange Queue



Handheld HotSync Exchange API

Applications use the standard Palm OS Exchange Manager API to send files via HotSync Exchange to the desktop. HotSync Exchange supports two Exchange Manager schemes:

- The **desktop scheme** (`_desktop`). This scheme supports direct exchange of a file to a desktop running HotSync Manager version 6.0 or later.
- The **send scheme** (`_send`). This scheme allows a user to send data using any available transport. Palm OS Cobalt includes HotSync technology as one such transport. Other transports may include SMS or Bluetooth, depending on the handheld's capabilities.

Because neither scheme supports the specification of a target desktop in the Exchange Manager URL, the file is, by default, sent during the next HotSync operation to *any* desktop. However, the user may select a specific target desktop for each file pending HotSync Exchange via the HotSync client, if desired (see Figure 3.6 on page 39).

Files transferred to the desktop via HotSync Exchange are placed in the HotSync user's `Exchange\From` subdirectory. If the target URL specifies a directory, the target file is placed in the corresponding subdirectory of `Exchange\From`. This subdirectory is created if it does not already exist. The target filename and extension are taken from the URL. For example, the URL:

```
_desktop:///portraits/Jennifer Connelly/
jcoscar12.bmp
```

specifies the transfer of a bitmap image from the handheld to the file

```
...Exchange\From\portraits\Jennifer
Connelly\jcoscar12.bmp
```

on the next desktop with which the handheld synchronizes.

For more information about the Exchange Manager on the handheld, see *Exploring Palm OS: High-Level Communications* in the Palm OS SDK.

Locating Files Transferred from the Handheld

If you are developing a desktop application that needs to be notified of a HotSync Exchange from the handheld to the desktop, you need to write a custom **notifier**. Your notifier can detect the presence of new files from the handheld in the user's `Exchange\From` directory at the end of a HotSync operation. For more information on notifiers, see "Desktop Notifiers" on page 71.

To get the path a HotSync user's `Exchange\From` directory, call one of these functions:

- C/C++ Sync Suite: User Data API, `UmGetUserDirectory()`
- COM Sync Suite: `PDUserData`.`GetUserDirectory()`

For more information on these APIs, see one of the following:

- "Using the User Data API" on page 145 in *C/C++ Sync Suite Companion*
- "Accessing User Data with PDUserData" on page 64 in *COM Sync Suite Companion*

4

Introducing Conduits

This chapter introduces basic conduit concepts, including the types of conduits, what conduits do, example synchronization logic they implement, and the principles developers need to adhere to when developing conduits.

This chapter includes the following topics:

Types of Conduits

Conduits can perform a variety of tasks, but they all fall into the three major types described in Table 4.1.

Table 4.1 Conduit types

Conduit Type	Description	Notes
Synchronization conduit	Performs whatever tasks a conduit developer designs it to do, typically synchronizing databases on the handheld with their desktop counterparts. See "Types of Synchronization" on page 51.	HotSync Manager for Windows runs registered synchronization conduits only if there is an application with a matching creator ID on the handheld. An exception to this exists for HotSync Manager 6.0 or later: a conduit can indicate that it must always be run regardless of whether matching application is on the handheld.
Install conduit	Installs Palm OS® applications and databases in a handheld's main memory or files on expansion cards. PalmSource ships HotSync Manager with several install conduits.	HotSync® Manager for Windows runs install conduits both before and after it runs synchronization conduits.
Backup conduit	Copies to the desktop entire handheld databases and applications without synchronizing specific records within them. PalmSource ships HotSync Manager with a backup conduit named Backup. The Backup conduit is intended for handheld databases and applications that do not have synchronization conduits associated with them.	HotSync Manager runs the backup conduit after it runs synchronization conduits.

Conduit Design Philosophy

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

Table 4.2 Conduit design goals

Design goal	Description
Fast execution	The overarching goal for the HotSync process is that a complete synchronization happen very quickly. Conduits need to be designed for optimal processing speed and minimal data transfer between the desktop and handheld.
Zero data loss	Conduits must take measures required to prevent data loss under any circumstances, including loss of connection during a HotSync operation.
Good conflict handling	Conduits that perform mirror-image synchronization must be able to gracefully handle conflicting modifications, also called a **double modify**. This occurs when the user modifies a record on both the desktop computer and the handheld. In addition to managing the conflict, the conduit should add an entry to the log to notify the user of this situation.
No user interaction	The user expects to be able to press the HotSync button on the cradle and have synchronization proceed without any required interaction. Conduits need to be constructed to avoid the need for user interaction. This is especially important for users who are synchronizing remotely (for example, over a network) and cannot interact with the desktop computer.

Conduit Tasks

Conduits synchronize data for a specific application on the handheld with the desktop computer. Conduits perform the following tasks:

- open and close databases on the handheld
- add, delete, and modify records on the handheld and desktop computer, converting formats as required

Because each application stores data in its own format, each conduit must implement custom data handling and conversion algorithms. To work with data on the handheld, a conduit calls the Sync Manager API, as described in "Classic Sync Manager API" on page 163.

Conduit Entry Points

To perform a conduit's tasks, HotSync Manager calls a conduit's **entry points**, which implement its synchronization logic, user customization, and responses to HotSync Manager's queries. There are two types of entry points that HotSync Manager calls in your conduit:

- Required entry points that must be implemented in each conduit.
- Optional entry points that provide additional information or capabilities.

This section provides a brief description of these conduit entry points. For more information about these functions, see the *Companion* document for the sync suite you are using to create a conduit ("Related Documentation" on page viii).

Table 4.3 provides a summary of the entry point functions.

Table 4.3 Conduit entry points called by HotSync Manager

Function Type	C/C++ Sync Suite for Windows Function Name	COM Sync Suite Function Name	Description
Required	`N/A`	`N/A`	Returns the conduit's name for HotSync Manager to display.
Required	`GetConduitVersion`	`IPDClientNotify->GetConduitVersion`	Returns the conduit's version number.
Required	`OpenConduit`	`IPDClientNotify->BeginProcess`	The main conduit entry point.
Required	`GetConduitInfo`	`IPDClientNotify->GetConduitInfo`	Returns information about the conduit to HotSync Manager (based on parameters passed in).
Customization (optional)	`ConfigureConduit` (prior to HotSync 3.0.1) or `CfgConduit` (HotSync 3.0.1 or later)	`IPDClientNotify->ConfigureConduit` (prior to HotSync 3.0.1) or `IPDClientNotify->CfgConduit` (HotSync 3.0.1 or later)	Presents a dialog that allows the user to configure the conduit. PalmSource strongly recommends that you provide this function.
Optional	`GetConduitName`	`N/A`	Returns the conduit's name for HotSync Manager to display.

Adding Messages to the HotSync Log

During a HotSync operation, HotSync Manager and any running conduit can add messages to the HotSync log, either on the desktop, on the handheld, or both. Because the handheld log has limited space, keep your entries in it as short as possible. If any unusual messages are added to the desktop log, HotSync Manager notifies the user at the end of the HotSync operation.

Conduits can also add messages to the HotSync log for debugging purposes.

Desktop HotSync Log

Each sync suite provides functions for logging messages. Each message string can optionally be time-stamped and associated with a specified activity type. Perhaps the simplest logging function is the one that adds an entry to the desktop HotSync log, as shown for each sync suite below:

`LogAddEntry()` in the C/C++ Sync Suite

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

`AddLogEntry()` in the COM Sync Suite

```
Sub AddLogEntry (
   ByVal pLogText As String, _
   [ByVal eActivity As ELogActivity = eText], _
   [ByVal bTimeStamp As Boolean = True], _
   [ByVal bPalmLog As Boolean = False])
```

Each of these functions allow your conduit to specify the activity that causes it to add a message: pass in one of the `Activity` (C/C++ Sync Suite) or `ELogActivity` (COM Sync Suite) values. Each time a log entry is added, HotSync Manager increments a counter to track how many warnings have occurred. The only log activity values that are not counted are:

C/C++ Sync Suite	COM Sync Suite
`slSyncStarted`	`eSyncStarted`
`slSyncFinished`	`eSyncFinished`
`slSyncAborted`	`eSyncAborted`
`slText`	`eText`

Of all the log activity values, only two explicitly cause text to be added to the log beyond any you specify in your conduit’s call to `LogAddEntry()/AddLogEntry()`:

C/C++ Sync Suite	COM Sync Suite
`slSyncAborted`	`eSyncAborted`
`slText`	`eText`

Specifying any other log activity value inserts only the text you specify.

When no synchronization errors or warnings have occurred, `slSyncFinished/eSyncFinished` inserts into the log the text “OK “ followed by the message you specify. If, on the other hand, one or more synchronization warnings have been logged, `slSyncFinished/eSyncFinished` inserts the text “OK “, then the message your conduit specified, then the text “with X message(s)”, where “X” is the number of messages logged. This text may be localized depending on what version of HotSync Manager you have installed.

The `slSyncAborted/eSyncAborted` activity code displays the message you passed in followed by “ synchronization failed”.

NOTE: The conduits provided by PalmSource, Inc. pass simply the conduit name into `LogAddEntry()` for the `slSyncFinished` and `slSyncAborted` activities. This results in messages such as “OK Address Book” and “Address Book synchronization failed” to appear. PalmSource suggests that you follow this standard for consistency.

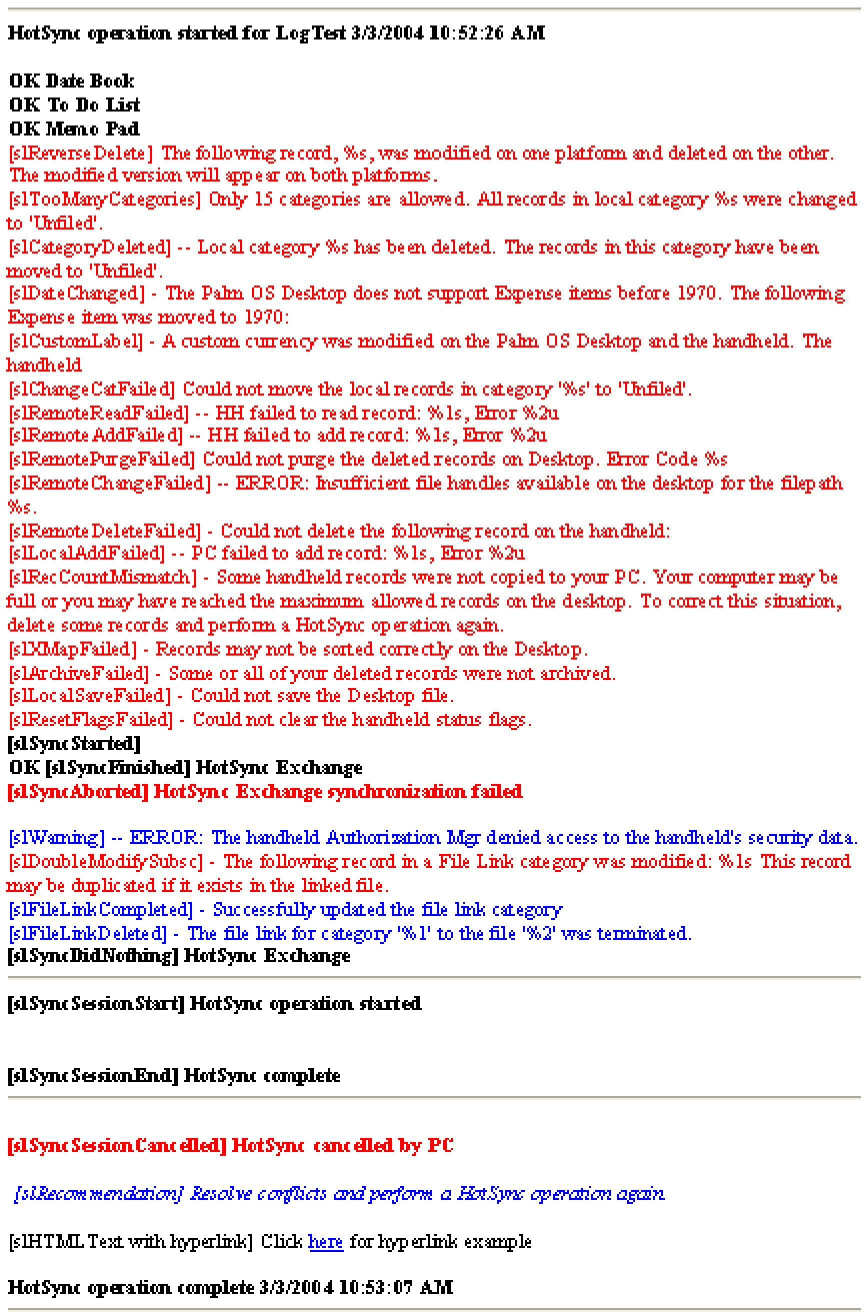
Beginning with HotSync Manager version 6.0.1, the HotSync log on the desktop is stored as an HTML file; in earlier versions, the log is an ASCII text file. When the user views the log, HotSync Manager calls the desktop’s default web browser to display it. Logging in HTML format allows HotSync Manager and conduits to do the following:

- Format entries for easier reading.
- Provide hypertext links to further information on the conduit developer’s web site.
- Make the log “active” by adding scripts.

When you add a log entry and specify a log activity value, the entry is automatically formatted to help indicate the type of activity that caused the entry to be logged—a warning, an error, the conduit started/completed, etc. All activity types filter out special HTML control characters so that they do not cause formatting errors; however, one special log activity value—`slHTMLText/eHTMLText`—allows the caller to pass in any string, including HTML tags and characters, to take advantage of HTML features.

Figure 4.1 shows an example of the HotSync log in HTML. It also shows how each activity type is formatted by HotSync Manager.

Figure 4.1 Example of formats used by HotSync log activity types



```
HotSync operation started for LogTest 3/3/2004 10:52:26 AM

OK Date Book
OK To Do List
OK Memo Pad
[slReverseDelete] The following record, %s, was modified on one platform and deleted on the other.
The modified version will appear on both platforms.
[slTooManyCategories] Only 15 categories are allowed. All records in local category %s were changed
to 'Unfiled'.
[slCategoryDeleted] -- Local category %s has been deleted. The records in this category have been
moved to 'Unfiled'.
[slDateChanged] - The Palm OS Desktop does not support Expense items before 1970. The following
Expense item was moved to 1970:
[slCustomLabel] - A custom currency was modified on the Palm OS Desktop and the handheld. The
handheld
[slChangeCatFailed] Could not move the local records in category '%s' to 'Unfiled'.
[slRemoteReadFailed] -- HH failed to read record: %1s, Error %2u
[slRemoteAddFailed] -- HH failed to add record: %1s, Error %2u
[slRemotePurgeFailed] Could not purge the deleted records on Desktop. Error Code %s
[slRemoteChangeFailed] -- ERROR: Insufficient file handles available on the desktop for the filepath
%s.
[slRemoteDeleteFailed] - Could not delete the following record on the handheld:
[slLocalAddFailed] -- PC failed to add record: %1s, Error %2u
[slRecCountMismatch] - Some handheld records were not copied to your PC. Your computer may be
full or you may have reached the maximum allowed records on the desktop. To correct this situation,
delete some records and perform a HotSync operation again.
[slXMapFailed] - Records may not be sorted correctly on the Desktop.
[slArchiveFailed] - Some or all of your deleted records were not archived.
[slLocalSaveFailed] - Could not save the Desktop file.
[slResetFlagsFailed] - Could not clear the handheld status flags.
[slSyncStarted]
OK [slSyncFinished] HotSync Exchange
[slSyncAborted] HotSync Exchange synchronization failed

[slWarning] -- ERROR: The handheld Authorization Mgr denied access to the handheld's security data.
[slDoubleModifySubsc] - The following record in a File Link category was modified: %1s This record
may be duplicated if it exists in the linked file.
[slFileLinkCompleted] - Successfully updated the file link category
[slFileLinkDeleted] - The file link for category '%1' to the file '%2' was terminated.
[slSyncDidNothing] HotSync Exchange

[slSyncSessionStart] HotSync operation started


[slSyncSessionEnd] HotSync complete


[slSyncSessionCancelled] HotSync cancelled by PC

 [slRecommendation] Resolve conflicts and perform a HotSync operation again

[slHTML Text with hyperlink] Click here for hyperlink example

HotSync operation complete 3/3/2004 10:53:07 AM
```

Handheld HotSync Log

The HotSync log on the handheld is accessible from the HotSync client application: from Launcher, tap **HotSync**, and then tap the **Log** button.

Because the handheld log has limited space, keep your entries in it as short as possible.

Each sync suite provide a function for adding messages to the handheld HotSync log:

`SyncAddLogEntry()` in the C/C++ Sync Suite

```
SInt32 SyncAddLogEntry (const char *pText)
```

`AddLogEntry()` in the COM Sync Suite

```
Sub AddLogEntry (
   ByVal pLogText As String, _
   [ByVal eActivity As ELogActivity = eText], _
   [ByVal bTimeStamp As Boolean = True], _
   [ByVal bPalmLog As Boolean = False])
```

Note that in the COM Sync Suite, the function for writing to the handheld log is the same as that for writing to the desktop log. However, with both of the above functions, you can write only unformatted text to the handheld log.

See Also `LogAddEntry()` and `SyncAddLogEntry()` in *C/C++ Sync Suite Reference*, `AddLogEntry()` in *COM Sync Suite Reference*

Types of Synchronization

The synchronization type depends on the nature of the applications involved and how you want to handle their data. The desktop and handheld synchronization types are summarized in Table 4.4.

Table 4.4 Conduit synchronization types

Synchronization Type	Description	Notes
Mirror-image	Ideal for applications that run on both the desktop and the handheld; allows modifications on both; makes data identical on both. Example applications for this kind of synchronization include the Address Book and Date Book applications.	Your conduit must provide conflict resolution solutions when the same record has been modified on both the desktop and the handheld (known as double modify). This must occur without user intervention.
One-directional	For applications that run on both the desktop and the handheld, but only allow data modification on one; copies the data to the other. Example: Stock quotation program that updates data on the desktop computer and copies the latest data to the handheld.	Requires less time, and should be used when possible.
Transaction-based	For applications that need to perform additional processing on the desktop computer between record synchronizations. Example: E-mail program that uses transaction processing to update the Inbox on the desktop computer.	Slows down the HotSync process. Use only when required.

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

Synchronization Logic

The user can make changes to data on either the handheld or the desktop computer. Depending on your conduit design, its synchronization logic might need to handle a variety of modification situations to ensure correct synchronization.

This section describes the logic implemented by the classic Generic Conduit Framework. Synchronization logic for schema databases is not covered in this document.

Record Synchronization

The logic shown in Table 4.5 lists the resultant action for each record modification condition. These conditions are based on the status bits maintained for records, which specify whether a record has been added, modified, deleted, or archived.

Table 4.5 Synchronizing records

Handheld Record Status	Desktop Record Status	Action
Add	No record	Add the handheld record to the desktop database.
Archive	Delete	Archive the handheld record and delete the record from both the handheld and desktop databases.

Table 4.5 Synchronizing records *(continued)*

Handheld Record Status	Desktop Record Status	Action
Archive	No change	Archive the handheld record and delete the record from both the handheld and desktop databases.
Archive	No record	Archive the handheld record.
Archive, change	Change	If the changes are identical, archive both the handheld record and the desktop record. If the changes are not identical, do not archive the handheld record; instead, add the desktop record to the handheld database, and add the handheld record to the desktop database.
Archive, no change	Change	Do not archive the handheld record; instead replace it with the desktop record.
Change	Archive, change	If the changes are identical, archive the handheld record and then delete the records from both the handheld and desktop databases. If the changes are not identical, do not archive the desktop record; instead, add the desktop record to the handheld database and add the handheld record to the desktop database.
Change	Archive, no change	Do not archive the desktop record; instead, replace the record in the desktop database with the handheld record.
Change	Change	If changes are identical, no action. If changes are not identical, add the handheld record to the desktop database and add the desktop record to the handheld database.
Change	Delete	Do not delete the desktop record; instead, replace the desktop record with the handheld record.
Change	No change	Replace the desktop record with the handheld record.

Table 4.5 Synchronizing records *(continued)*

Handheld Record Status	Desktop Record Status	Action
Delete	Change	Do not delete the handheld record; instead, replace the handheld record with the desktop record.
Delete	No change	Delete the record from the desktop and handheld databases.
No change	Archive	Archive the desktop record, then delete the record from both databases.
No change	Change	Replace the handheld record with the desktop record.
No change	Delete	Delete the record from the desktop and handheld databases.

Category Synchronization

The logic shown in Table 4.6 lists the resultant action for each category modification condition.

Table 4.6 Synchronizing categories

Category Name Conditions	Category ID Conditions	Category Index Conditions	Actions
Desktop name matches handheld name	N/A	Desktop index matches handheld index	No action required.
Desktop name matches handheld name	N/A	Desktop index does not match handheld index	Change all desktop records in the category to use the handheld's category ID.

Table 4.6 Synchronizing categories *(continued)*

Category Name Conditions	Category ID Conditions	Category Index Conditions	Actions
Desktop name not found on handheld and desktop name has been modified	Desktop ID matches handheld ID	N/A	Change category name on the handheld to match desktop.
Desktop name not found on handheld	Desktop ID not found on handheld	Desktop index is not in use on handheld	Desktop category added to handheld.
Desktop name not found on handheld	Desktop ID not found on handheld	Desktop index is already in use on handheld	Desktop category is added with next free index on handheld, and desktop records in the category are updated to use new index.

5

Introducing HotSync Manager

This chapter introduces HotSync® Manager for Windows. It presents only HotSync Manager's basic settings and functions as well as topics relevant to conduit developers.

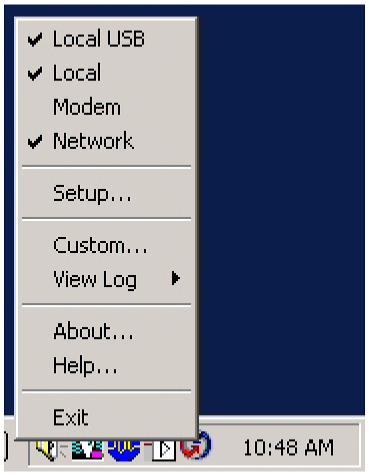
This chapter describes the following topics:

For details on all HotSync Manager user settings, see HotSync Manager's online help (**HotSync Manager** > **Help**).

HotSync Manager Settings

Figure 5.1 shows the HotSync Manager menu that appears when you click the HotSync Manager icon. You find the HotSync Manager icon in your Windows taskbar.

Figure 5.1 HotSync Manager menu



Local, **Local USB**, **Modem**, **Network**, and **Infrared**—identify the type of connections that can be used between the handheld and the desktop computer. More than one option can be selected.

Setup—displays a dialog box for setting HotSync Manager preferences and configuring the selected type of connection.

Custom—displays a dialog box listing all the conduits registered for the current Windows user and the synchronization preferences of the selected HotSync user. See "User's Conduit Synchronization Preferences" on page 62 for more information.

View Log—displays the HotSync log for the specified HotSync user. HotSync Manager 6.0.1 and later display a formatted log in the desktop's default web browser; earlier versions display the log as a text file.

About—provides HotSync Manager version.

Help—displays online help.

Exit—exits HotSync Manager.

First Synchronization

The first time you perform a HotSync operation, HotSync Manager on the desktop prompts you either to create a new HotSync user or to choose an existing one. If you create a new one, you must enter a HotSync user name on the desktop, which HotSync Manager transfers to the handheld along with the HotSync **user ID** it generates. If you choose an existing HotSync user, then HotSync Manager restores all of that HotSync user's data to the handheld, including the user name and ID. HotSync Manager recognizes your handheld during all subsequent HotSync operations on the current desktop and does not ask you for a user name nor does it generate a user ID for that handheld again.

HotSync Manager versions earlier than 6.0 require that you perform your first HotSync operation with a local, direct connection, rather than using a modem. Subsequent synchronizations can be done with any valid connection type. HotSync Manager for Windows, versions 6.0 and later, permits you to perform your first synchronization using any valid connection type. See "Types of Connections" on page 59.

Types of Connections

HotSync Manager allows different kinds of connections between a handheld and a desktop computer. Each connection type causes HotSync Manager to initiate synchronization activities in a different manner, as described in Table 5.1.

NOTE: The HotSync process can be started *only* from the handheld. PalmSource does not support any method of starting a HotSync operation from the desktop.

Table 5.1 HotSync connection types

HotSync Connection Type	HotSync Activation Description
Direct cable using the cradle	The cradle's cable connects to the desktop computer using a serial or USB port. When the user presses the HotSync button on the cradle, two pins are shorted on the handheld. This activates the HotSync client on the handheld, which "awakens" HotSync Manager on the desktop computer.
Modem	The handheld is connected to a modem that is dialed into a desktop computer. The user sets up HotSync Manager on the desktop computer to use a modem. On handhelds running Palm OS versions earlier than Palm OS Cobalt, the user must sets up Modem Sync in the HotSync client on the handheld. When the user taps the HotSync button, the handheld dials into the desktop computer and connects with HotSync Manager. On handhelds running Palm OS Cobalt, the user selects a connection profile for a modem. The HotSync client on the handheld has no special Modem Sync selection. Instead the user taps Network and specifies a desktop to synchronize with over the modem connection to a network. When the user taps the HotSync button, the handheld uses the current network connection or dials into the specified network and connects with HotSync Manager on the selected desktop computer.

Table 5.1 HotSync connection types *(continued)*

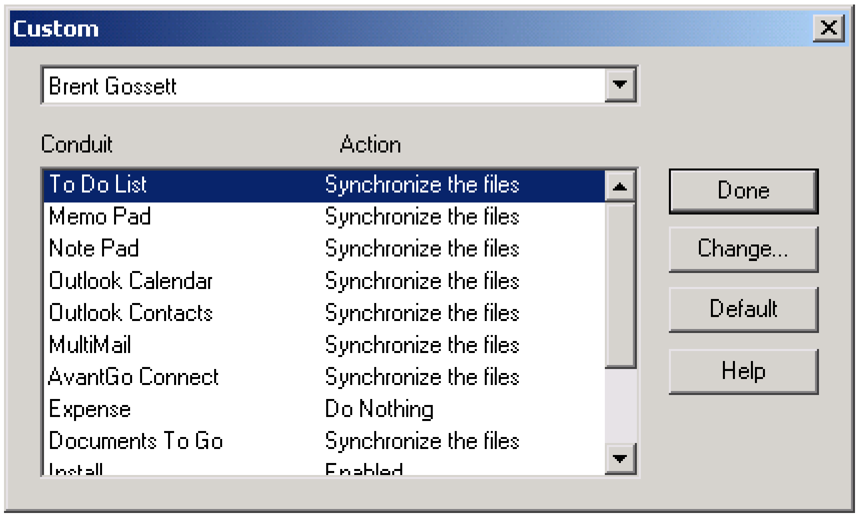
HotSync Connection Type	HotSync Activation Description
Network	The cradle’s cable is connected to a desktop computer on a local area network or the handheld itself is directly connected to a network. The target desktop computer is also connected to the network and HotSync Manager is running on that desktop computer. When the user presses the HotSync button, the HotSync client on the handheld connects to HotSync Manager running on the target desktop computer.
Infrared	If the handheld has an infrared port, it can synchronize with a desktop computer equipped with an infrared (IR) port that supports the IrCOMM implementation of the Infrared Data Association (IrDA) standard. The user sets up HotSync Manager to use the desktop’s IR port and selects the IR option in the HotSync client on the handheld.

IMPORTANT: Typically, design your conduit to be used with all supported types of connections (you do not want to design a new conduit for each type of connection). Therefore avoid implementing pop-up windows or other user interfaces in your conduit that require user input during synchronization. The reason is that users are typically not present at the desktop computer that they synchronize with via modem or network connections.

User's Conduit Synchronization Preferences

HotSync Manager's **Custom** dialog box (Figure 5.2) displays the selected HotSync user's synchronization preference for each conduit registered for the current Windows user.[1]

Figure 5.2 HotSync Manager's Custom dialog box

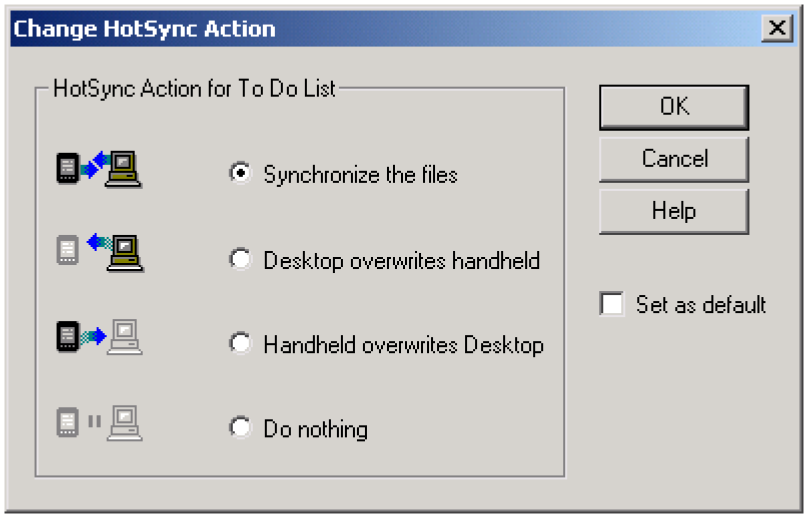


Conduit	Action
To Do List	Synchronize the files
Memo Pad	Synchronize the files
Note Pad	Synchronize the files
Outlook Calendar	Synchronize the files
Outlook Contacts	Synchronize the files
MultiMail	Synchronize the files
AvantGo Connect	Synchronize the files
Expense	Do Nothing
Documents To Go	Synchronize the files
Install	Enabled

When the user selects a conduit and clicks **Change**, HotSync Manager calls a C API-based conduit's `CfgConduit()` entry point (for its equivalent COM-based entry point, see "Conduit Entry Points" on page 44). Though this entry point is optional, PalmSource recommends that conduits implement it. All of the default conduits that PalmSource ships with HotSync Manager implement it and present users with a **Change HotSync Action** dialog box (Figure 5.3) that they can use to choose how they want a conduit to synchronize.

1. HotSync Manager, versions 6.0 and later, enables a registered conduit to opt out of appearing in the **Custom** dialog box. A conduit must return a specific value to opt out, otherwise HotSync Manager displays it in this dialog box.

Figure 5.3 Change HotSync Action dialog box



(●) Synchronize the files
( ) Desktop overwrites handheld
( ) Handheld overwrites Desktop
( ) Do nothing
☐ Set as default

When HotSync Manager calls your conduit's `CfgConduit()` or equivalent entry point, it passes in information that identifies the selected HotSync user and that user's stored permanent and temporary synchronization preferences. A user's synchronization preference specifies what the user wants the conduit to do during subsequent HotSync operations. The default conduits prompt the user to select **Synchronize the files, Desktop overwrites handheld**, **Handheld overwrites Desktop**, or **Do nothing**. HotSync Manager also supports a custom preference that you can define as you wish for your conduit.

HotSync Manager distinguishes between permanent and temporary synchronization preferences.

temporary synchronization preference: The user's preference for what a conduit should do during *only the next* HotSync operation. All HotSync operations thereafter revert to the user's permanent synchronization preference.

permanent synchronization preference: The user's preference for what a conduit should do during *all subsequent* HotSync operations. This preference is "permanent" unless the user changes the permanent or temporary preference again later.

The default conduits display the temporary preference if it is set; otherwise they display the permanent preference. If the user selects a preference, selects **Set as default**, and clicks **OK**, then the default conduits store it as a permanent synchronization preference. If **Set as default** is not selected, then they store it as a temporary synchronization preference. You can implement a similar interface in your conduit's **Change HotSync Action** (or equivalent) dialog box.

When the user changes the synchronization preference for your conduit, whether it is a temporary or permanent preference, your conduit's `CfgConduit()` or equivalent entry point must pass back to HotSync Manager the new preferences, which it then stores for you.

For more information on the `CfgConduit()` or equivalent entry point, see the *C/C++ Sync Suite Companion* or *COM Sync Suite Companion*.

For a description of how the user's preference affects the type of synchronization your conduit should perform, see "Effect of User Synchronization Preferences" on page 100.

Support for Multiple Users

HotSync Manager versions *6 and later* support multi-user versions of Windows. This means that HotSync Manager can be installed once for all Windows users and each Windows user can have different settings. For example, each user can use a different set of conduits, because conduits can be registered either for the current Windows user or for all Windows users (the system).

NOTE: HotSync Manager versions *earlier than 6.0* can be installed for only the current Windows user, so they have no knowledge of multiple Windows users. To use earlier versions of HotSync Manager with versions of Windows that support multiple users, you must install HotSync Manager separately for each Windows user. The rest of this section discusses only HotSync Manager versions 6.0 and later.

HotSync Manager versions 6 and later support two classes of users on the desktop, defined below and described in the following subsections:

HotSync User: Represents the user of a single Palm Powered handheld that synchronizes with the desktop using HotSync Manager. There is a one-to-one relationship between handhelds and HotSync users.

Windows User: Represents a single Windows login on a multi-user version of Windows.

Figure 5.4 illustrates the relationships between these two classes of users on the desktop.

Figure 5.4 Relationship between multiple HotSync users and multiple Windows users

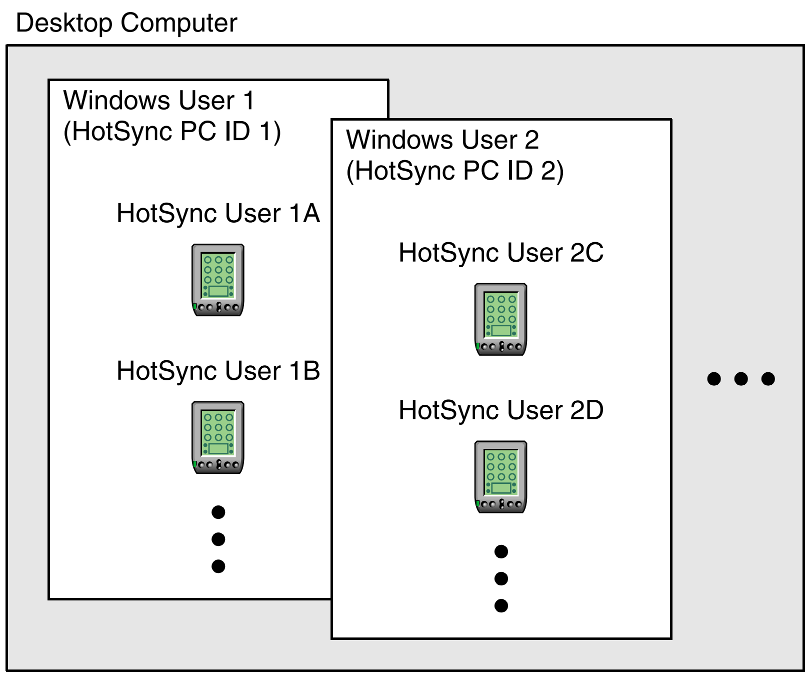
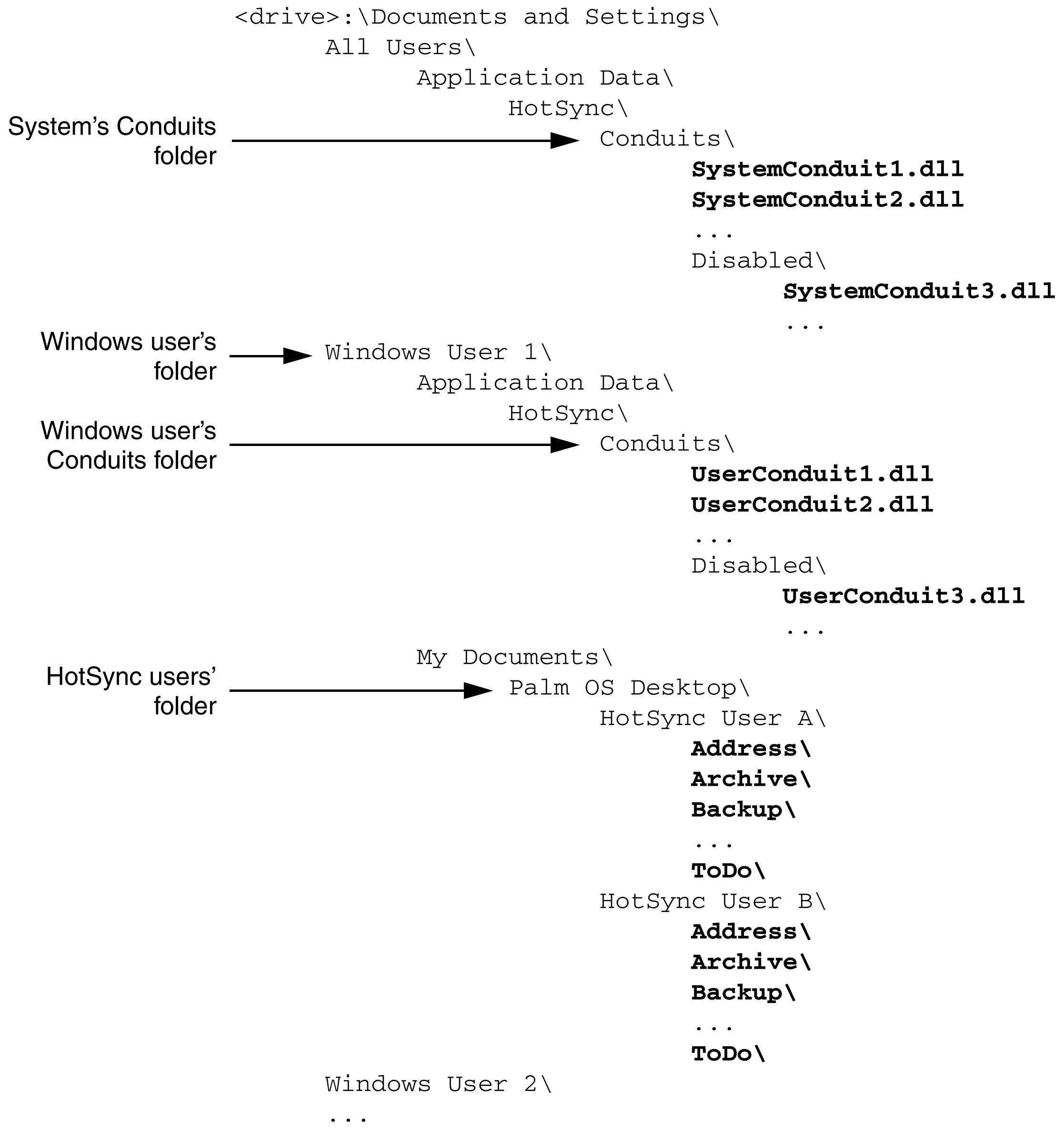


Figure 5.5 shows the relationship between the Windows users' and HotSync users' folders. Conduit filenames and conduit folder names are shown in bold.

Figure 5.5 Windows and HotSync users' folders



```
<drive>:\Documents and Settings\
      All Users\
            Application Data\
                  HotSync\
                        Conduits\
                              SystemConduit1.dll
                              SystemConduit2.dll
                              ...
                              Disabled\
                                    SystemConduit3.dll
                                    ...
      Windows User 1\
            Application Data\
                  HotSync\
                        Conduits\
                              UserConduit1.dll
                              UserConduit2.dll
                              ...
                              Disabled\
                                    UserConduit3.dll
                                    ...
            My Documents\
                  Palm OS Desktop\
                        HotSync User A\
                              Address\
                              Archive\
                              Backup\
                              ...
                              ToDo\
                        HotSync User B\
                              Address\
                              Archive\
                              Backup\
                              ...
                              ToDo\
      Windows User 2\
      ...
```

HotSync User

A HotSync user can be created by:

- HotSync Manager when a handheld synchronizes with it for the first time
- the Palm OS Desktop software, either when the user is prompted by the installer or by selecting **Tools** > **Users**
- the User Data API when called by a third-party application to add a new HotSync user

All versions of HotSync Manager associate one handheld per HotSync user. All HotSync users created in a single Windows user login share the same set of conduits; you cannot install a conduit for some HotSync users and not others within the same Windows user login. However, each HotSync user can have different conduit configuration settings—that is, the settings for each conduit displayed in the **Custom** dialog box can be different for each HotSync user.

Within a single Windows user login, HotSync Manager creates one HotSync user for each handheld that synchronizes with it.[2] This supports two scenarios:

- one person using a single Windows user login and who has multiple handhelds
- multiple people using a single Windows user login, each with a handheld

The following list details what must be the same for all HotSync users and what is different for each HotSync user in a single Windows user login:

- All HotSync users share the *same:*
  - HotSync Manager executable and libraries[3]
  - set of conduits (including backup and install) and notifiers

2. Two handhelds cannot reliably synchronize with the same HotSync user created in the same Windows user login.
3. An exception to consider is when you have two instances of HotSync Manager 6.0 or later installed: the one installed with Palm OS Desktop and the one installed with the CDK. But this is not the usual case for end users.

  - HotSync users' folder, where each user's desktop data is stored in separate subfolders (see Figure 5.5 on page 66)
  - HotSync local connection settings—that is, COM ports and modem settings in the **Setup** dialog box
  - PC ID, generated by HotSync Manager when it is run for the first time by each Windows user
- Each HotSync user has a *different:*
  - HotSync user name and ID
  - HotSync user's desktop data folder, where each conduit can store data for each HotSync user separately (see Figure 5.5 on page 66)
  - set of conduit synchronization preferences for each conduit—for example, Synchronize, Desktop overwrites handheld, etc.—specified via the **Custom** dialog box
  - HotSync log

Windows User

A Windows user can be created when Windows is first installed or at any later time. All the versions of Windows supported by HotSync Manager 6.0 and later allow you to create multiple Windows user logins, each with separate Windows user settings. When HotSync Manager is installed, it creates a set of system-level settings. HotSync Manager uses these system settings both for the current Windows user at install time and when other Windows users run HotSync Manager for the first time.

When a system has multiple Windows user logins, each Windows user can create a separate set of one or more HotSync users.[4] This supports a broad range of user scenarios: multiple people, each with a different Windows user login and each with one or more handhelds.

The following list details what is the same for all Windows users and what is different for each Windows user:

4. Two handhelds can synchronize with the same HotSync user created in two different Windows user logins. This is the same as synchronizing with two different desktop computers.

- All Windows users share the *same:*
  - HotSync Manager executable and libraries
  - set of *system*-registered conduits (including backup and install) and notifiers, which can be overridden by *user*-registered conduits with the same creator ID
  - communications port access (one Windows user cannot start a HotSync operation while another's is in progress)
- Each Windows user has a *different:*
  - PC ID, generated by HotSync Manager when it is run for the first time by each Windows user
  - set of HotSync users, whose data and settings are completely separate from those of HotSync users created by a different Windows user[5]
  - set of *user*-registered conduits and notifiers, which override *system*-registered conduits with the same creator ID

For more on conduits registered for the system vs. for the current Windows user, see "User- and System-registered Conduits and Notifiers" on page 78.

5. For each Windows user, HotSync Manager defines a separate `Core\Path` setting, which is the path of the directory that holds all HotSync users' directories created by the current Windows user. This directory is located in the current Windows user's `My Documents\Palm OS Desktop` folder.

Interfacing with Windows Desktop Applications

Other Windows desktop applications can interact with HotSync Manager. Of course, conduits can do so for the purpose of accessing data on the handheld, but desktop applications and installers can interface with HotSync Manager also. The following subsections introduce two interfaces that HotSync Manager for Windows offers:

HotSync Manager API

The HotSync Manager API enables your desktop application or installer to control HotSync Manager in the following ways:

- Stop, start, restart, or refresh HotSync Manager.

  HotSync Manager versions earlier than 6.0 require that you restart or refresh HotSync Manager after registering a conduit; otherwise it does not discover your newly registered conduit. HotSync Manager versions 6.0 and later does not require this. See "Resolving Conduit Conflicts" on page 80.
- Call HotSync Manager dialogs to customize conduits or change connection settings or to display the HotSync Log.
- Check or set the connection status type for serial, USB, and network connections.
- Check whether HotSync Manager is idle or synchronizing.

The HotSync Manager API is available on Windows only. This API is natively a C API declared in the C/C++ Sync Suite for Windows, but the COM Sync Suite provides an interface to it via the `PDHotSyncUtility` object. See the *C/C++ Sync Suite Companion* or *COM Sync Suite Companion* for more information.

Desktop Notifiers

When both an application on the desktop computer and its corresponding conduit can modify user data, HotSync Manager can notify the desktop application when a HotSync operation is starting so that both are not changing data on the desktop at the same time. For example, the Address Book application in the Palm OS® Desktop software and its conduit can both modify the user's Address Book data on the desktop. To prevent this from happening at the same time, a notifier tells Palm OS Desktop when a HotSync operation is starting so that it can prevent the user from changing data until after the HotSync operation is complete.

NOTE: If your conduit accesses desktop data but does not share that data with a desktop application, you do not need to provide a notifier.

HotSync Manager calls all registered notifier DLLs, or **notifiers**, at the beginning of the HotSync process, sending a message that a HotSync operation is starting, and again at the end to signal that the HotSync process is finished and whether it was successful. A notifier must implement an entry point for HotSync Manager to call. In your notifier's entry point, you must implement whatever mechanism you want to act on messages that HotSync Manager sends it. Typically, your notifier notifies your desktop application in a format that it can understand; then your desktop application does whatever it needs to do. Because a notifier is a Windows DLL, you can implement whatever logic you wish; it is designed for, but not limited to, notifying a desktop application.

For a demonstration of the behavior of a notifier, you can try to launch or edit a record in Palm OS Desktop during a HotSync operation. Because its notifiers signal it when a HotSync operation begins, Palm OS Desktop prevents all actions that might modify its data files.

Before HotSync Manager can call your notifier, it must be registered with HotSync Manager at install time. See "Registering Conduits and Notifiers with HotSync Manager" on page 73.

Developing a notifier is supported only in the C/C++ Sync Suite for Windows. See the *C/C++ Sync Suite Companion* for more information.

6

Registering Conduits and Notifiers with HotSync Manager

Conduit and notifier registration is a one-time operation performed outside of the HotSync process by a desktop installer application that provides HotSync Manager with basic information about a conduit or notifier. This information enables HotSync Manager to uniquely identify and to find a conduit or notifier when it needs to call any of a its entry points.

NOTE: The most common reason why a conduit does not run is that it is not properly registered.

Conduits and notifiers must be registered with HotSync Manager before the HotSync process begins. This chapter has the following sections:

Methods of Registering a Conduit or Notifier

There are two fundamental ways to register a synchronization conduit with HotSync Manager for Windows, which are described in the following subsections:

These two methods are equivalent; HotSync Manager does not treat synchronization conduits differently based on how they are registered.

NOTE: Notifiers, install conduits, and the backup conduit can be conventionally registered only; they cannot be folder-registered.

Conventional Registration

Conventional registration requires an installer to write registration information in the configuration entries at install time. This method is supported by all versions of HotSync Manager for Windows for registering the following types of conduits and notifiers, described these subsections:

IMPORTANT: The Conduit Configuration (CondCfg) utility enables you to conventionally register a synchronization conduit or notifier during development and testing without writing code to call Conduit Manager. However, CondCfg is not an end-user utility, so PalmSource does not permit you to redistribute this developer-only utility with your conduit or notifier.

Synchronization Conduits

To conventionally register a *synchronization conduit*, an installer calls the Conduit Manager API to write several conduit configuration entries. The minimum required set of entries are:

`Conduit:` The filename (or full path) of a C API-based conduit DLL.

`Creator:` The creator ID, usually of the application on the handheld associated with the conduit. This is the key by which HotSync Manager uniquely identifies a registered conduit. It is required for conduits created with any Sync Suite.

`COMClient:` The name or ProgID of a COM-based conduit. Not required for other types of conduits.

`ClassName` and `ClassPath13`: The class name and path of all classes used by a Java-based conduit. Not required for other types of conduits.

You can specify values for other standard, optional conduit configuration entries—for example, `Priority`, whose value ranges from 0 to 4, specifies the relative order in which HotSync Manager should run a conduit. Or you can create and set your own additional entries for your own use. See the *Sync Suite Companion* document for the Sync Suite describes in more detail how to use this method to register a conduit.

NOTE: HotSync Manager versions *earlier than 6.0* must be refreshed or restarted after registering a conduit for a newly registered conduit to be recognized. However, *versions 6.0 and later* automatically refresh the registered conduits list so that you do not need to refresh or restart HotSync Manager.

Install Conduits

To conventionally register an *install conduit*, an installer calls the Install Conduit Manager API to write several install conduit configuration entries. The minimum required set of entries are:

`Mask:` A bit mask value that uniquely identifies an install conduit on the current desktop computer.

`CreatorID:` A unique ID associated with an install conduit.

`Extensions:` The file type extensions of the files that an install conduit can install.

`Directory:` Name of the install directory from which a specific install conduit transfers files from the desktop to a handheld.

`Module:` The filename of an install conduit.

Backup Conduit

To conventionally register a *backup conduit*, an installer calls the Conduit Manager API to write one of the HotSync Manager configuration entries:

`HotSync Manager\BackupConduit:` Filename of the conduit that HotSync Manager calls as the backup conduit near the end of the HotSync process.

Notifiers

To conventionally register a *notifier*, an installer calls the Notifier Install Manager API to write one of the HotSync Manager configuration entries:

`HotSync Manager\NotifierN:` Filename of a notifier that HotSync Manager calls at the start and end of the HotSync process.

Folder-based Conduit Registration

Folder-based conduit registration requires that you:

- Implement logic in your conduit's `GetConduitInfo()` entry point to respond to HotSync Manager's request for registration information at run time.
- Copy your conduit to the current Windows user's (or the system's) `Conduits` folder at install time.

NOTE: Folder-based conduit registration is supported only for C API-based synchronization conduits and HotSync Manager for Windows, versions 6.0 and later.

HotSync Manager discovers a folder-registered conduit by looking in the `Conduits` folders and then calling each conduit's `GetConduitInfo()` entry point. If the conduit DLL is not present or it does not return registration information when HotSync Manager requests it, then a conduit is not folder-registered and cannot be called by HotSync Manager.

The `Conduits` folder contains a `Disabled` subfolder. Conduit Manager APIs allow you to disable a folder-registered conduit, which simply moves the file from the `Conduits` folder to the `Disabled` subfolder. A user can disable and enable folder-registered conduits also by moving the conduit file from one folder to the other.

See Figure 5.5 on page 66 for the paths of the `Conduits` and `Disabled` folders for both system-registered and user-registered conduits.

Comparison of Conduit Registration Methods

Folder-based conduit registration has the following advantages over the conventional method:

- You set your conduit's registration information at *compile time*, so your installer does not have to make multiple Conduit Manager calls to write registration information at install time.
- Your installer can be simpler. It only needs to find a folder and copy a file to it.

- It is a quicker way to get HotSync Manager to run your conduit while you're developing and testing it. You do not have to write an installer or use the CondCfg utility to register your conduit.

However, you may want to use conventional conduit registration if you want your installer to set your conduit's registration information set at *install time*. This way you can customize this information then or change it later by calling the Conduit Manager. For folder-registered conduits, this information is read-only after you compile your conduit.

Of course, you can take advantage of both methods: register your conduit by folder during development and testing, because it is easier; then write an installer that registers it conventionally when you deploy your conduit, if you need to customize or change its registration information at install time.

User- and System-registered Conduits and Notifiers

HotSync Manager for Windows, versions 6.0 and later, distinguishes between conduits and notifiers registered for the current Windows user and those registered for the system. At the beginning of the HotSync process, Conduit Manager compiles a list of the creator IDs of all nonconflicting, registered conduits. Conduit Manager passes this list to HotSync Manager, which uses it to determine which ones it will actually call when the current Windows user performs a HotSync operation. Conduit Manager looks at all registered conduits. It does not distinguish between how a conduit is registered (conventional or folder-based), but it does look at who it is registered for. If a system- and a user-registered conduit have the same creator ID, then Conduit Manager puts only the user-registered conduit on its list.

NOTE: A user-registered conduit always takes precedence over a system-registered one with the same creator ID.

Table 6.1 shows what Conduit Manager[1] compares when it reconciles its list of user-registered conduits and notifiers with its system-registered list.

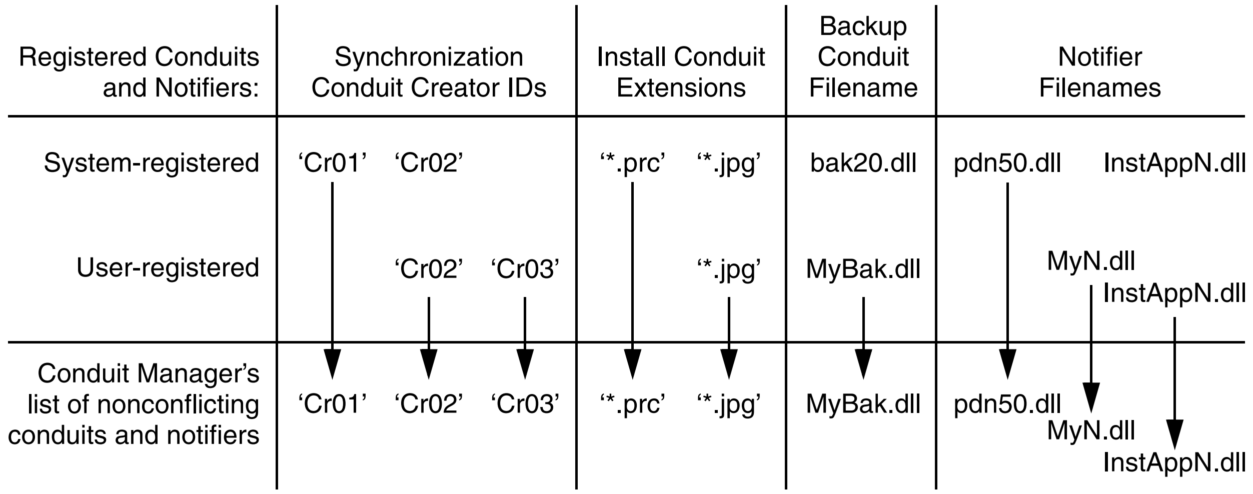
Table 6.1 How Conduit Manager reconciles user- vs. system-registered conduits and notifiers

For two...	a user-registered one takes precedence over a system-registered one...
Synchronization conduits	with the same creator ID
Install conduits	with the same registered filename extension (see “Running Install Conduits” on page 105)
Backup conduits	always, because only one backup conduit can be run per Windows user
Notifiers	with the same path and filename

Figure 6.1 shows an example of how Conduit Manager decides which conduits and notifiers to put on its list.

1. “Conduit Manager” includes Install Conduit Manager and Notifier Install Manager here.

Figure 6.1 Example: precedence of user- over system-registered conduits and notifiers



Registering a conduit or notifier for the system registers it for all Windows users who do not already have a corresponding conduit or notifier registered. This behavior enables an installer to register a conduit or notifier for all Windows users on a system without requiring each Windows user to run the installer. This scheme also enables each Windows user to register a different set of conduits and notifiers. However, note that all HotSync users created within a given Windows user login share the same set of conduits and notifiers, though each HotSync user can configure these conduits differently. See "Support for Multiple Users" on page 64 for more on the differences between HotSync users and Windows users.

Resolving Conduit Conflicts

The HotSync process is based on the principle that only one conduit can synchronize an application's data during a HotSync operation; otherwise a second conduit that synchronizes the same data in the same HotSync operation could not trust the status of the database and record modification flags. Conduit registration tries to prevent this situation.

Conventional conduit registration does not allow a second user conduit to register with the same creator ID as another user conduit registered for the current Windows user; the same is true about two system-registered conduits for the system. Conduit Manager prevents such double registration and returns an error on the second attempt to conventionally register using the same creator ID. Conduit Manager allows two conduits to be registered with the same creator ID only if one is registered for the current Windows user and the other for the system. The user-registered conduit always takes precedence, as described in "User- and System-registered Conduits and Notifiers" on page 78, so corresponding user- and system-registered conduits are never in conflict.

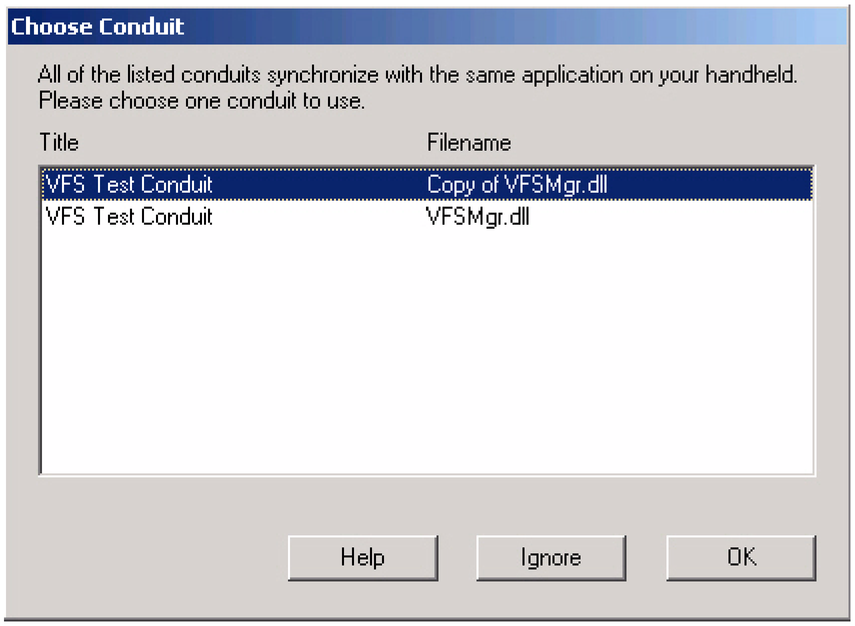
However, it is possible for one folder-registered conduit to have the same creator ID as another folder-registered or conventionally registered conduit. The reason is that Conduit Manager does not validate folder-registered conduits until it is called upon to create a list of all registered conduits. Conduit Manager creates this list whenever any of the following occurs:

- The user starts a HotSync operation.
- The user or the HotSync Manager API does any of the following:
  - Starts or restarts HotSync Manager or causes it to refresh its list of conduits.
  - Selects HotSync Manager's **Custom** menu item.
- An installer or application calls the Conduit Manager API to do any of the following:
  - Conventionally register a conduit.
  - Enable a disabled folder-registered conduit.
  - Return a list of registered conduits.

Any of these events can result in Conduit Manager discovering two or more conduits with the same creator ID. When this happens, the Conduit Manager API returns an error.

When HotSync Manager receives this error from Conduit Manager for the first time for a given HotSync user and conduit creator ID, it displays a **Choose Conduit** dialog box like the one in Figure 6.2.

Figure 6.2 HotSync Manager prompts the user to resolve conflicting conduits


Choose Conduit

All of the listed conduits synchronize with the same application on your handheld.
Please choose one conduit to use.

Title	Filename
VFS Test Conduit	Copy of VFSMgr.dll
VFS Test Conduit	VFSMgr.dll

If the user chooses a conduit to run and clicks **OK**, HotSync Manager stores this preference for the current HotSync user in the user data store on the desktop, so it does not prompt the user to resolve the same conflict again. All the other conflicting conduits on the list are effectively unregistered, so HotSync Manager does not run them or show them in its **Custom** dialog box.

However, if the user clicks **Ignore**, the conflict remains unresolved. HotSync Manager removes all of the conflicting conduits from its list of conduits to run[2] and does not list them when the user selects the HotSync Manager's **Custom** menu item. Each time HotSync Manager prompts the Conduit Manager to create a list of registered conduits, as described previously, HotSync Manager displays a **Choose Conduit** dialog box again for the user to resolve the conflict.

HotSync Manager provides the **Choose Conduit** dialog box as a last resort. PalmSource recommends that before you copy your folder-

2. Because conflicting conduits are not run, the Backup conduit can back up databases on the handheld with the conflicting creator ID as usual. Therefore the databases are at least backed up, if not synchronized, despite the conflict.

based conduit to the `Conduits` folder, check whether any conduits with the same creator ID are already registered. If so, prompt the user to choose whether to unregister the existing conduit, so you can add yours to the `Conduits` folder, or to exit. Handling potential conduit conflicts within the context of your specific product makes more sense to the user than relying on HotSync Manager's generic **Choose Conduit** dialog box.

7

Understanding the HotSync Process

This chapter presents the HotSync® process, step-by-step from the moment the user presses the HotSync button to backing up databases and logging. The sections in this chapter are:

Overview

The **HotSync Manager** application on the desktop oversees the **HotSync process**. The HotSync process is PalmSource's patented,[1] one-button method of automatically synchronizing all data between a handheld and a desktop computer. HotSync Manager invokes conduits on the desktop to handle data for each handheld application that requires data synchronization. An instance of the HotSync process, a **HotSync operation** or session, begins when the user presses the HotSync button (on the cradle or on the display) and ends after the backup conduit runs and final cleanup is complete. In addition to data synchronization, the HotSync process:

- assigns HotSync user names and IDs to handhelds
- manages individual or multiple handhelds with a single desktop computer
- backs up handheld data onto a desktop computer
- installs new applications and databases on a handheld
- restores all data to a handheld after it is hard-reset

HotSync Manager runs in the background on the desktop computer and monitors one or more communications ports, waiting for a command from a handheld. Once that command is received, HotSync Manager determines which synchronization operations need to be run and begins the process.

NOTE: The HotSync process can be started *only* from the handheld. PalmSource does not support any method of starting a HotSync operation from the desktop.

Table 7.1 summarizes the entire HotSync process, including the actions of the user, HotSync Manager, and the handheld. The following sections describe each of the actions in more detail.

1. United States patent number 6,000,000 (cool, eh?).

Table 7.1 Summary of the HotSync process

User action or display	HotSync Manager actions	Handheld actions
1. User presses HotSync button	Reads the HotSync user ID from the handheld and validates the user against the desktop user data store. Locates the user's data on the desktop computer. Reads system and state information from the handheld. Calls notifier to tell desktop application that synchronization is about to begin. See "Intitializing the HotSync Process" on page 90.	HotSync Manager process starts on the handheld and sends a signal to the desktop computer. The HotSync client sends a notification to handheld applications that a HotSync session has started.
2. "Connecting with desktop"	Retrieves a list of creator IDs of all applications (database of type `'appl'`) on the handheld and a list of all conduits registered on the desktop. Creates a list of conduits to run: those that have applications with matching creator IDs on the handheld and those that specify they should be run even if there is no matching application on the handheld.[1] If no conduit is installed for the creator ID, but the backup bit is set for the database, adds the database to the list for the Backup conduit. Also adds a database to the backup list, even if there is a conduit installed for the creator ID, as long as the database is not of type `'DATA'`. See "Intitializing the HotSync Process" on page 90.	

Table 7.1 Summary of the HotSync process *(continued)*

User action or display	HotSync Manager actions	Handheld actions
3. “Synchronizing...”	Calls install conduits to install any applications or databases that have been queued on the desktop computer for that purpose. Runs each synchronization conduit in its list in priority order by calling specific entry points. Conduits in turn call Sync Manager on the desktop to read and write data on the handheld. Calls install conduits again to handle any installations that were generated by other conduits.[2] Calls the Backup conduit to copy any databases that are in the backup list. See “Running Install Conduits” on page 105, “Running Synchronization Conduits” on page 107, and “Running the Backup Conduit” on page 109.	The HotSync client responds to calls from Sync Manager on the desktop, which in turn calls Data Manager and other Palm OS managers.

Table 7.1 Summary of the HotSync process *(continued)*

User action or display	HotSync Manager actions	Handheld actions
4. Completion message(s)	Calls notifiers to tell desktop applications that synchronization has completed. Updates synchronization information, including the sync time and user ID, on the handheld. If any of the conduits added messages to the log, the HotSync Manager displays a message to the user who can then examine those log entries. See "Finishing the HotSync Process" on page 110.	Sublaunches handheld applications whose databases were modified. Sublaunches any applications just installed that the HotSync operation is finished. Notifies any application that registered with the Notification Manager for the "sync complete" notification.

1. Only HotSync Manager for Windows, versions 6.0 and later, can query a C API-based conduit to specify that it should be run even if there is no matching application on the handheld.
2. Only HotSync Manager for Windows, versions 3.01 or later, calls each install conduit a second time, if it has files queued to instal, after it runs all synchronization conduits. Versions earlier than 3.01 do not perform this step.

Intitializing the HotSync Process

When the user starts a HotSync operation, but before any conduits run, HotSync Manager performs these tasks in sequence, described in the following subsections:

Getting the Current Desktop User ID (PC ID)

The current desktop OS user ID (**PC ID**) is a pseudo-random number that HotSync Manager generates when it is first run by each separate desktop user. Note that HotSync Manager ensures that the PC ID for each desktop operating system user is unique from the others on the same computer. HotSync Manager retrieves the PC ID for the current desktop user and stores it on the handheld.

HotSync Manager versions 6.0 and later also retrieve the desktop's host name and IP address to help identify the desktop to the handheld when HotSync Manager determines whether the handheld "trusts" the desktop. See "Getting the Desktop Trust Status" on page 92.

Getting the Current HotSync User

Each Palm Powered™ handheld has a unique **HotSync user ID** and a user-specified HotSync user name associated with it after it has been synchronized. The user ID is a pseudo-random number that HotSync Manager generates during a handheld's first HotSync operation after a complete reset, or during its first-ever HotSync operation.

HotSync Manager stores the HotSync user ID and name on the handheld. It also maintains a **user data store** on the desktop for each desktop OS user. This store consists of user names, IDs, and other information about all HotSync users created by the current desktop OS user. This store also tracks the most recent synchronization date for the desktop computer.

NOTE: HotSync Manager does not necessarily generate the same HotSync user ID for the same user name on two different handhelds. However, HotSync Manager does ensure that each user ID is unique among the IDs in its user data store for the current desktop OS user.

At the start of the HotSync process, HotSync Manager requests from the handheld its HotSync user name and HotSync user ID, which it then compares with those saved in its user data store for the current desktop user. The possible results are detailed in Table 7.2.

Table 7.2 Determining the current HotSync user

If the handheld...	then HotSync Manager...
Has no user ID, either because this is the first HotSync operation for this handheld or it has been hard-reset	Prompts the desktop user to do one of the following: • Create a new user name. HotSync Manager then generates a new HotSync user ID. • Choose an existing HotSync user from a list. Writes the HotSync user name and user ID to the handheld, identifying the current HotSync user.
Has a user ID that matches one in the user data store	Has identified the HotSync user.
Has a user ID that does *not* match one in the user data store	Prompts the user whether to create a new HotSync user. If yes, creates a new HotSync user on the desktop using the name and ID it retrieved from the handheld. If no, aborts the HotSync operation.

Getting the Desktop Trust Status

Palm OS® Cobalt enables users to specify in the HotSync client which desktop computers they trust to have their secure databases backed up to or synchronized with.

The handheld establishes trust status by handing out a "shared secret" to the desktop during the next sync. Thereafter, the handheld challenges the desktop to check whether it has the shared secret key, and if so, permits a trusted sync.

At this point, HotSync Manager identifies whether the handheld trusts the current desktop. If trust is verified, then HotSync Manager permits synchronization conduits and the backup conduit to access secure databases; otherwise, it does not grant access to secure databases.

For more on trusted desktops, see Chapter 9, "Understanding Trusted Desktops," on page 149.

Calling Notifiers

HotSync Manager for Windows calls all registered notifiers. A notifier is a DLL that implements entry points, which HotSync Manager calls at the beginning and end of the HotSync process. Your implementation can then notify your desktop application that a HotSync operation is starting. This warning gives your desktop application a chance to save any pending changes to its desktop data that is about to be synchronized by your conduit or to cancel the HotSync operation.

For more on notifiers, see Chapter 8, "Writing a Desktop Notifier," on page 93 in the *C/C++ Sync Suite Companion*.

Determining the Sync Mode

The HotSync process is designed for maximum efficiency, which means enabling conduits to minimize, whenever possible, the data that they need to transfer between the desktop and handheld. Because the methods for making this determination differ for schema and non-schema databases, the details are discussed separately in the following subsections:

Sync Mode for Schema Databases

The Data Manager on the handheld tracks more detailed change information for schema databases than for non-schema databases. A conduit can use this to significantly minimize how much data it must transfer "over the wire" to and from the handheld. The details of how this is done are discussed in "How the Schema Sync Manager Determines the Sync Mode" on page 95. The remainder of this section describes how your conduit gets the sync mode and how it should perform in each mode.

To get the sync mode, a conduit must call `SyncDbGetSyncMode()` (C/C++ Sync Suite) or `GetSyncTypeInfo()` (COM Sync Suite) for *each* schema database that it is going to synchronize. These functions read the change tracking information for the database you specify, compare it with the **change context** cached on the desktop, and return the type of synchronization that your conduit should perform for each **sync atoms**. The synchronization mode applies only to the specified schema database and only for the currently running conduit, not for the entire HotSync operation.

NOTE: HotSync Manager also passes back a sync type via your conduit's `OpenConduit()` entry point (C/C++ Sync Suite) or via the `PDHotsyncInfo.SyncType` property (COM Sync Suite). The returned `eFast` and `eSlow` sync types are based solely on whether this handheld's last HotSync operation was with the current desktop. For non-schema databases, this is sufficient for a conduit to determine whether to perform a fast or slow sync. However, for schema databases, this value is not sufficient. Conduits synchronizing schema databases must call the "get sync mode" function (described above) to determine the most efficient synchronization to perform.

The following sync types for each sync atom in a schema database are determined by calling `SyncDbGetSyncMode()` or `GetSyncTypeInfo()`:

Fast sync: Instances of the sync atom have changed and all change flags are valid. This mode enables a conduit to transfer only the data that has changed since the last HotSync operation with the current desktop.

Fast sync after a purge: This sync type is passed back via `SyncDbGetSyncMode()` or `GetSyncTypeInfo()` as a fast sync, but with an additional "Deletes Purged" flag set. Therefore it is the same as a fast sync, except that the schema Sync Manager can determine that instances of a sync atom that were deleted on the handheld have been purged since the last HotSync operation with the current desktop. All change flags are valid, so a conduit needs only to read the ID lists of the sync atoms that were purged and compare them with those on the desktop to identify the purged instances of each atom. Then the conduit can proceed as in a fast sync. This ability to track purged data enables a conduit to perform a fast sync across multiple desktops, an efficiency available only with schema databases.

Slow sync: The change flags for a sync atom are not reliable. Therefore a conduit that performs a mirror-image synchronization must compare each instance of each sync atom in the handheld database with the corresponding data on the desktop to determine how to synchronize them.

No change: No instances of the sync atom have changed in the specified schema database

The sync types listed above depend on the specified schema database and sync atom, are determined by the schema Sync Manager, and retrieved by calling `SyncDbGetSyncMode()` or `GetSyncTypeInfo()`. A conduit must also consider the user's **synchronization preference**, if the conduit implements this feature, when determining what type of synchronization to perform. If the user wants the "desktop to overwrite handheld," then despite whether a fast sync is possible, your conduit should do what the user wants. See "Effect of User Synchronization Preferences" on page 100 for how the standard user synchronization preferences affect what action your conduit should take.

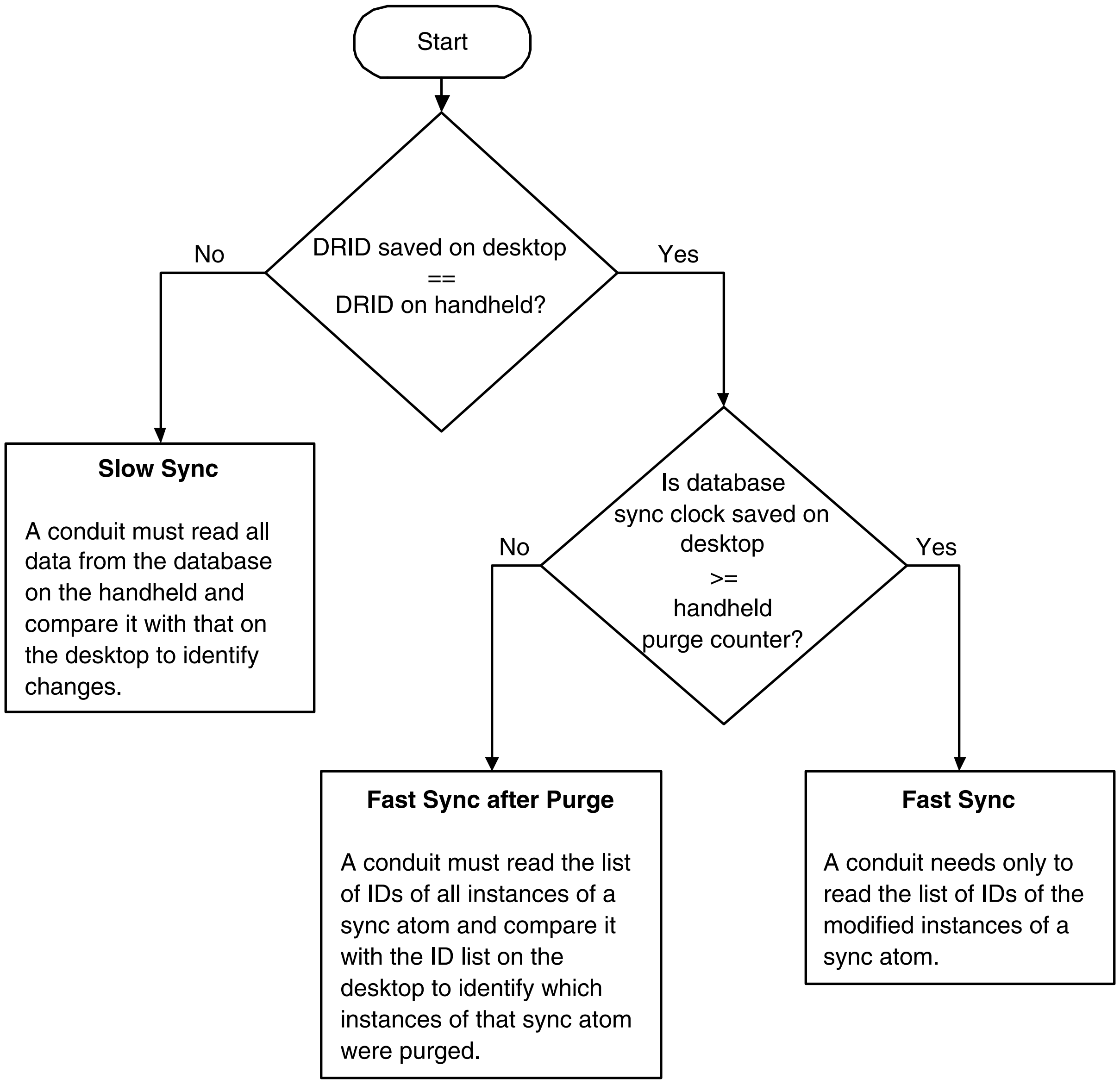
The following subsection details how the schema Sync Manager determines whether fast or slow sync mode is appropriate.

How the Schema Sync Manager Determines the Sync Mode

At the end of each HotSync operation, the schema Sync Manager caches the **database reset identifier** (DRID) of each schema database on the desktop. To determine whether a fast or slow sync is possible for a given database, the schema Sync Manager compares the DRID cached during the last HotSync operation with that of the schema database on the handheld. If they are equal, then a fast sync is possible because the **sync clock**, **change counter**, and **purge counter** are dependable. If they are not equal, then only a slow sync is possible; a conduit must read all data from the database on the handheld and compare it with that on the desktop to identify changes.

Figure 7.1 shows how the schema Sync Manager determines whether a conduit can perform a fast sync with a schema database.

Figure 7.1 Determining whether to perform a fast or slow sync for a schema database


Start
DRID saved on desktop == DRID on handheld?
No
Yes
Slow Sync
A conduit must read all data from the database on the handheld and compare it with that on the desktop to identify changes.
Is database sync clock saved on desktop >= handheld purge counter?
No
Yes
Fast Sync after Purge
A conduit must read the list of IDs of all instances of a sync atom and compare it with the ID list on the desktop to identify which instances of that sync atom were purged.
Fast Sync
A conduit needs only to read the list of IDs of the modified instances of a sync atom.

Also at the end of each HotSync operation, the schema Sync Manager caches on the desktop the sync clock value of each schema database. If a fast sync is possible for a schema database, then the schema Sync Manager compares the database's sync clock value cached on the desktop during the last HotSync operation with the purge counter *for each sync atom* in the database on the handheld. If the sync clock value is *greater than or equal to* a sync atom's purge counter, then that sync atom has not been purged; therefore a conduit needs only to read the list of IDs of the modified instances of that sync atom provided by a call to the schema Sync Manager. Otherwise, that sync atom has been purged; therefore a conduit must read the list of IDs of all instances of that sync atom and compare it with the ID list on the desktop to identify which instances of that sync atom were purged. Then it can proceed as in a fast sync, reading an ID list (or the data itself) of only the modified instances of a sync atom. For more about change tracking in schema databases, see "Change Tracking Services for Schema Databases" on page 139.

Sync Mode for Non-schema Databases

The Data Manager on the handheld tracks less change information for non-schema databases than for schema databases. A conduit can still use this to minimize how much data it must transfer "over the wire" to and from the handheld. The details of how this is done are discussed in "How the Classic Sync Manager Determines the Sync Type" on page 98. The remainder of this section describes how your conduit gets the sync mode and how it should perform in each mode.

To get the sync type, a conduit must read the sync type passed to it by HotSync Manager via the conduit's `OpenConduit()` entry point (C/C++ Sync Suite) or via the `PDHotsyncInfo`.`SyncType` property (COM Sync Suite). This sync type applies to all non-schema databases for the entire HotSync operation.

The following sync types are possible for non-schema databases:

Fast sync: All database and record "dirty" flags are valid. This mode enables a conduit to send data between the handheld and desktop *only* if a change has occurred since the most recent HotSync operation. A conduit performing a fast sync should not transfer data that has not been modified.

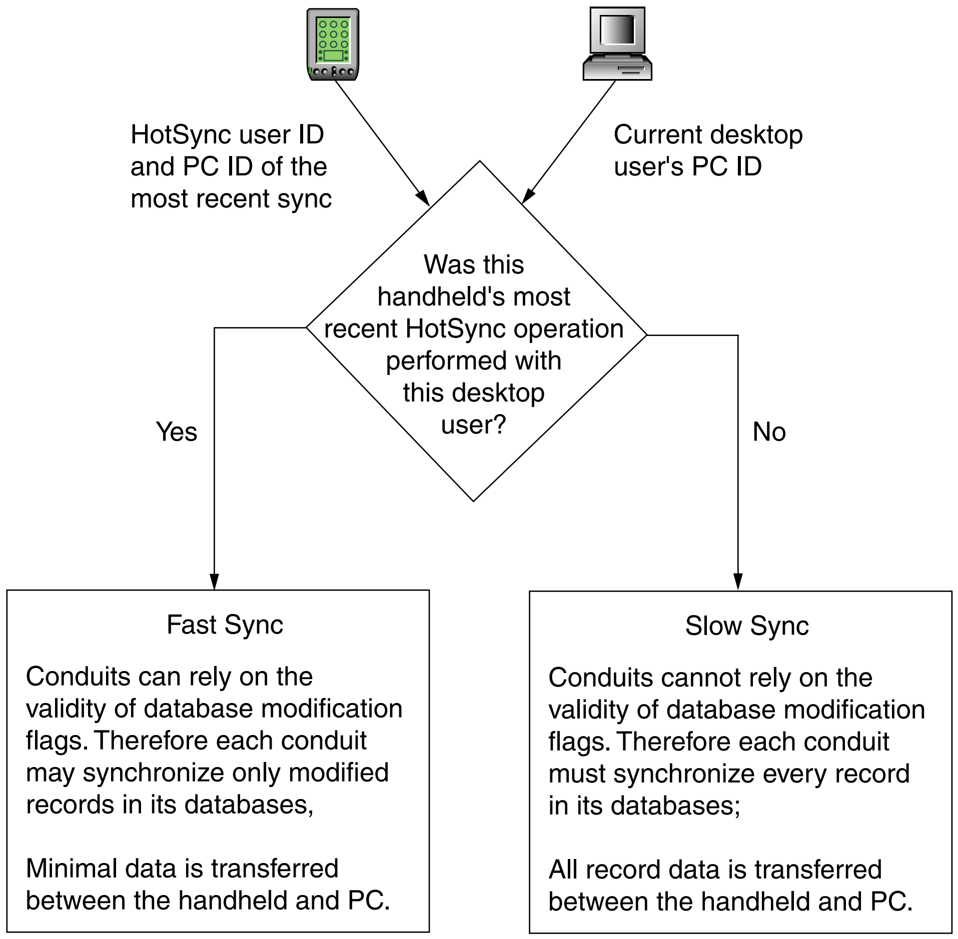
Slow sync: The "dirty" flags are not reliable, because the handheld has synchronized with a different desktop more recently than the current desktop. Therefore a conduit that performs a mirror-image synchronization must compare each record in the handheld database with the corresponding record on the desktop to determine how to synchronize the records.

These sync types depend on whether the handheld has synchronized with a different desktop since the last HotSync operation with the current desktop. A conduit must also consider the user's **synchronization preference**, if the conduit implements this feature, when determining what type of synchronization to perform. If the user wants the "desktop to overwrite handheld," then despite whether a fast sync is possible, your conduit should do what the user wants. See "Effect of User Synchronization Preferences" on page 100 for how the standard user synchronization preferences affect what action your conduit should take.

How the Classic Sync Manager Determines the Sync Type

HotSync Manager retrieves the HotSync user ID and the PC ID of the most recent synchronization from the handheld. It compares these values with the values stored on the desktop computer during the previous HotSync operation. If the PC ID values match, then conduits can perform a fast sync. Figure 7.2 shows how HotSync Manager determines whether conduits can perform a fast sync.

Figure 7.2 Determining whether to perform a fast or slow sync for non-schema databases


HotSync user ID and PC ID of the most recent sync
Current desktop user's PC ID
Was this handheld's most recent HotSync operation performed with this desktop user?
Yes
No
Fast Sync
Conduits can rely on the validity of database modification flags. Therefore each conduit may synchronize only modified records in its databases,
Minimal data is transferred between the handheld and PC.
Slow Sync
Conduits cannot rely on the validity of database modification flags. Therefore each conduit must synchronize every record in its databases;
All record data is transferred between the handheld and PC.

Non-schema databases on the handheld include flags that HotSync Manager uses to improve the efficiency of synchronization operations:

- a database "dirty" flag, which indicates whether the non-schema database has changed since the most recent synchronization
- record "dirty" flags, which indicate whether each record has been changed since the most recent synchronization

If HotSync Manager determines that it can perform a fast sync, conduits can rely on the validity of the "dirty" flags and therefore work only with database records that have changed.

Note that there are some situations in which a conduit must synchronize all of the records in an non-schema database, even if they have not been changed since the last HotSync operation:

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

Effect of User Synchronization Preferences

As described in "User's Conduit Synchronization Preferences" on page 62, a standard but optional feature of a conduit is to allow the user to specify what the conduit does during the next HotSync operation. This sync type preference is stored in the user data store on the desktop and passed to a conduit via the conduit's `OpenConduit()` entry point (C/C++ Sync Suite) or via the `PDHotsyncInfo`.`SyncType` property (COM Sync Suite). Whether HotSync Manager passes back the `eFast` or `eSlow` sync type is based on whether the handheld last performed a HotSync operation with the current desktop. The `eHHtoPC`, `ePCtoHH`, and `eDoNothing` sync types are passed back based on the user's synchronization preference.

The standard user synchronization preferences are described below, along with the sync type that is passed back to a conduit for each:

Synchronize (`eFast` or `eSlow`): Perform a mirror-image synchronization. The Sync Manager determines whether a fast or slow sync is possible and passes back either `eFast` or `eSlow` to the conduit. Note that if the conduit is synchronizing a schema database, the sync type passed back as described above is not sufficient to determine whether a fast sync is actually possible. See "Sync Mode for Schema Databases" on page 93 for details. For non-schema databases, the result is sufficient, as described in "Sync Mode for Non-schema Databases" on page 97.

Handheld overwrites desktop (`eHHtoPC`): Discard the desktop data and overwrite it with the handheld data. This is a user preference, for which the HotSync Manager passes the conduit the value `eHHtoPC`. In response, the conduit deletes the data on the desktop and reads all of the data from the handheld database.

Desktop overwrites handheld (`ePCtoHH`): Discard the handheld data and overwrite it the desktop data. This is a user preference, for which the HotSync Manager passes the conduit the value `ePCtoHH`. In response, the conduit either removes the handheld database entirely or it removes all data (records, row data, table definitions, category information, category membership). Then it adds all data from the desktop.[2]

Do nothing (`eDoNothing`): Make no changes to data on either the desktop or handheld. This is a user preference, for which the HotSync Manager passes the conduit the value `eDoNothing`. The conduit does not need to exchange information with the handheld. Even though the conduit does not need to do anything, HotSync Manager still calls it in case the conduit needs to do something—for example, record for its own purposes that it was run.

Creating a List of Conduits to Run

Before it runs the first conduit, HotSync Manager must determine which synchronization conduits it should run during the current HotSync operation.

The following steps describe how HotSync Manager for Windows creates a list of which registered conduits to run during a HotSync operation:

1. HotSync Manager retrieves a list of the creator IDs of all applications on the handheld—that is, all databases of type `'appl'`.
2. Conduit Manager compiles a list of the creator IDs of all conduits that are registered on the desktop.

2. A conduit can delete a secure database, but it cannot create one. So to overwrite a secure database, a conduit must remove all of its data and write the desktop data into it instead.

3. HotSync Manager compares the two lists. It adds a conduit's creator ID to its list of conduits to run if *either* of the following is true:

   – a creator ID on the applications list matches one on the conduits list

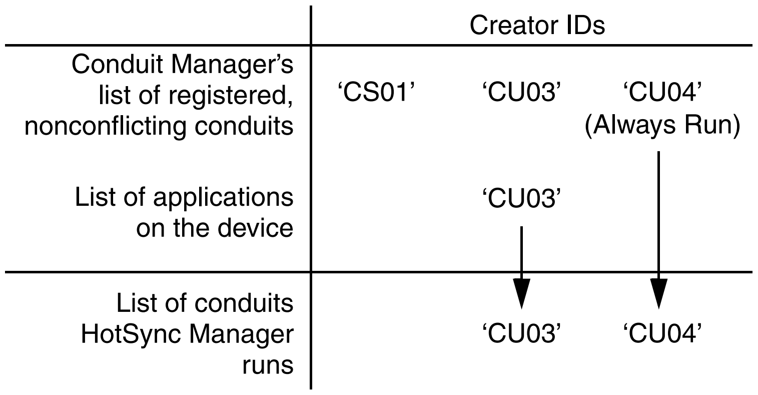
   or

   – a conduit responds to HotSync Manager's query that it must always run[3]

   Figure 7.3 shows an example of how HotSync Manager compares these lists. Notice that in this example, the conduit with creator ID `'CU04'` runs even without having a matching application on the handheld. This can happen only because this conduit responded to HotSync Manager's query that it should be run regardless of whether an application with the same creator ID is on the device.

4. HotSync Manager sorts its list of conduits to run in priority order. Priority values range from 0 to 4; HotSync Manager runs conduits with a value of 0 first and those with 4 last.[4]

Figure 7.3 Example: how HotSync Manager for Windows determines which conduits to run


Creator IDs
Conduit Manager's list of registered, nonconflicting conduits
'CS01'
'CU03'
'CU04' (Always Run)
List of applications on the device
'CU03'
List of conduits HotSync Manager runs
'CU03'
'CU04'

3. Only HotSync Manager versions 6.0 and later can query conduit to discover whether it should be run even if there is no matching application on the handheld.
4. You cannot predetermine the exact order in which HotSync Manager runs registered conduits; you can set only the relative order via the priority value.

Creating a List of Databases to Back Up

HotSync Manager runs the Backup conduit after all other conduits have run. But before any conduits run, HotSync Manager determines which databases on the handheld should be backed up.

HotSync Manager puts a database on its list of databases to back up only if *both* of the following are true:

- The backup bit is set in the database on the handheld.
- The database on the handheld has been modified since the last HotSync operation with this desktop.

  HotSync Manager detects that the database is modified if both of the following are true:

  – The database's Modify bit is set.

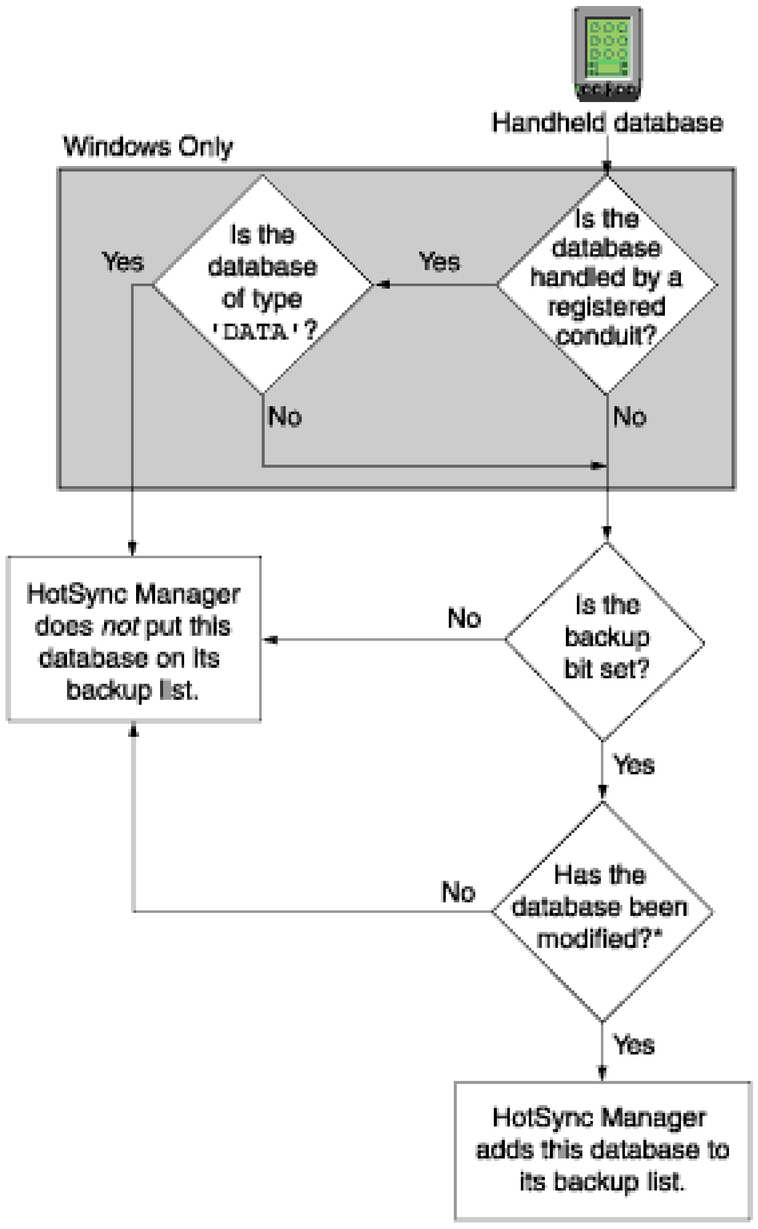
  – The database's modification date on the handheld is later than that of the corresponding database in the HotSync user's `Backup` folder on the desktop.

NOTE: *Exception*—HotSync Manager for Windows does *not* put a database on its backup list if its type is `'DATA'` (case sensitive) *and* a conduit registered with the same creator ID is on its list of conduits to run.

The reason for this exception is that if a registered conduit has already synchronized its databases, then they do not need to be backed up.

Figure 7.4 shows how HotSync Manager determines whether to put a handheld database on its list of databases for the Backup conduit to back up.

Figure 7.4 Determining whether a database is handled by the Backup conduit


Handheld database
Windows Only
Is the database handled by a registered conduit?
Yes
Is the database of type 'DATA'?
Yes
No
No
HotSync Manager does *not* put this database on its backup list.
No
Is the backup bit set?
Yes
No
Has the database been modified?*
Yes
HotSync Manager adds this database to its backup list.

Running Install Conduits

The first conduits that HotSync Manager runs are the install conduits. Install conduits typically transfer Palm OS applications and databases to a handheld's primary memory, or any type of file to a handheld's expansion card. All versions of HotSync Manager ship with at least one default install conduit.

HotSync Manager for Windows, versions 3.01 and later, runs install conduits *twice:*

1. Before running all other conduits but after performing the actions described in "Intitializing the HotSync Process" on page 90. This ensures that files queued for installation before the HotSync process begins are installed before any synchronization conduits that might need them.
2. After running all synchronization conduits but before running the backup conduit. See "Running Install Conduits Again" on page 108 for details.

NOTE: If you want your synchronization conduit to install your databases during a HotSync operation, it can call `SyncInstallDatabase()` (C/C++ Sync Suite) or `InstallDatabase()` (COM Sync Suite). This capability is available only with HotSync Manager versions 6.0 and later and works only with handhelds running Palm OS Cobalt.

Install Tool, an end-user application that ships with HotSync Manager, enables users to queue files for install conduits to transfer to the handheld during the next HotSync operation. To queue files for installation, Install Tool calls the Install Aide API. The **Install Aide** is a desktop library that Install Tool and any third-party application or conduit can call to queue files for install conduits to transfer to the handheld. The Install Aide API is documented in the *C/C++ Sync Suite Reference* and *COM Sync Suite Reference*.

HotSync Manager runs an install conduit only if the Install Aide has been called to queue a file with a filename extension that an install conduit is registered to handle. When an application or conduit calls Install Aide to install a file, Install Aide performs the following actions:

- Sets a flag bit for the selected HotSync user to indicate which install conduit is registered to handle files with the extension of the specified file.
- Copies the file to a specific location in the HotSync user's directory; for example, PRC and PDB files are copied to the HotSync user's `Install` directory.

HotSync Manager ships with the following install conduits:

Install: Installs image files of Palm OS databases into primary storage on a handheld. Can handle PRC, PDB, SDB, SSD, and PQA file types. Included with all versions of HotSync Manager.

Install Service Templates: Installs image files of Palm OS network configurations and network scripts into primary storage. Can handle PNC and SCP file types. Included with all versions of HotSync Manager. Works only with handhelds running a version of Palm OS earlier than Palm OS Cobalt.

Install to Card: Installs files of all types destined for an expansion card. Included with versions 4.0 and later of HotSync Manager.

HotSync Exchange: Installs files into primary storage if they are of types that have been registered with the Palm OS Cobalt HotSync Exchange library. Implements the desktop side of HotSync Exchange functionality and available with HotSync Manager versions 6.0 and later. Enables users to transfer files of all types between the desktop and handheld. For more information, see Chapter 3, "Using HotSync Exchange," on page 27.

Restore Operation

A restore operation is a special use of the Install conduit. In this case, HotSync Manager passes the Install conduit a synchronization type of `eInstall`, indicates that this is the handheld's first HotSync operation with an existing HotSync user on the current desktop, and passes it the `Backup` folder name. (HotSync Manager supplies this information via a `CSyncProperties` structure passed into `OpenConduit()` (C/C++ Sync Suite) or via a `PDHotsyncInfo` object (COM Sync Suite). In response, the Install conduit does the following:

1. Restores security data that is required to read secure databases upon restore, if the handheld is running Palm OS Cobalt and the desktop is trusted. If it fails to restore security data, the HotSync operation is aborted, because secure databases cannot be restored if the security data is not restored first.
2. Transfers each database image file in the `Backup` folder to the handheld as a Palm OS database, if a newer database is not already present on the handheld.

Running Synchronization Conduits

After it initializes the HotSync process, HotSync Manager runs the synchronization conduits on the list that it compiled as described in "Creating a List of Conduits to Run" on page 101. It runs these conduits one at a time in priority order, those with priority value 0 first and those with 4 last.

To run each conduit, HotSync Manager calls the conduit's main entry point—`OpenConduit()` for C API-based conduits, `IPDClientNotify->BeginProcess()` for COM-based conduits. At this time, HotSync Manager passes information to a conduit about the current HotSync operation, including:

- type of synchronization—based on the user's preference and whether the handheld last synchronized with the current desktop (see "Determining the Sync Mode" on page 93 for details)
- HotSync user name

- whether the current HotSync operation is the first for the handheld
- whether the handheld is connected via modem or not

While a conduit is running, it can do anything that similar binaries are allowed to do on the platform they are on. Some actions that only a conduit can perform during the HotSync process include:

- read and write data on the handheld by calling the Sync Manager API
- access expansion cards and file systems on them by calling the Expansion Manager API and the Virtual File System Manager API
- queue messages to be written to the current HotSync user's log file by calling the HotSync Log API

HotSync Manager proceeds to the next synchronization conduit only when the previous conduit returns a success value or a nonfatal error code from its main entry point.

Running Install Conduits Again

HotSync Manager for Windows, versions 3.01 and later, runs each install conduit a second time, if it has files queued to install, after all synchronization conduits have run but before the backup conduit runs. This second install phase ensures that files queued for installation by synchronization conduits that just ran are installed during the same HotSync operation.

See "Running Install Conduits" on page 105 for details.

Running the Backup Conduit

HotSync Manager runs the backup conduit after all other conduits have completed their operations. Only one conduit can be registered as the backup conduit. This section describes the behavior of the backup conduit named **Backup**, which PalmSource provides with HotSync Manager.

As with any conduit, HotSync Manager passes the backup conduit a synchronization type, which indicates whether the user wants databases backed up, wants databases transferred to the handheld from the `Backup` folder, or neither.

If the synchronization type is backup, the Backup conduit does the following:

1. Creates a database image file for each database on the list that HotSync Manager created as described in "Creating a List of Databases to Back Up" on page 103. These backup files are located in the current HotSync user's `Backup` folder and have unique filenames that include the database creator ID, type, and name.[5]
2. Backs up security data that is required to read secure databases upon restore, if the handheld is running Palm OS Cobalt and the desktop is trusted.
3. Moves files from the `Backup` folder to the `Archive` folder, if they are no longer on the handheld. For example, if the user deleted a database since the last HotSync operation, then the user does not want the database to be restored after a hard reset.

Because the Backup conduit backs up any databases (applications, databases, preferences, etc.) that have their backup bit set, HotSync Manager can restore a handheld that has been hard-reset to its original state.

5. The Backup conduit that ships with HotSync Manager for Windows, versions 6.0 and later, generates backup filenames using the creator ID, type, and name but only for handhelds running Palm OS Cobalt. Otherwise, it generates filenames based only on the database name.

IMPORTANT: PalmSource, Inc. strongly recommends that you set the backup bit for all applications and databases you create, with the exception of data that you do not want restored. Note that if you set the backup bit of an application, the Backup conduit backs it up only when necessary, so the user does not pay a performance penalty.

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

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

Finishing the HotSync Process

After running the conduits, HotSync Manager cleans up and stores state information. The following describe HotSync Manager's final steps:

1. HotSync Manager writes the HotSync user name and ID to the handheld, if this is the handheld's first HotSync operation.
2. If the handheld is running a version of Palm OS earlier than Palm OS Cobalt, HotSync Manager updates the password on the desktop for the current HotSync user, if the password on the handheld does not match that saved on the desktop. HotSync Manager cannot access the user's password on a handheld running Palm OS Cobalt.
3. HotSync Manager writes the PC ID of the current desktop user to the handheld.

4. HotSync Manager writes to the handheld the date and time that the HotSync operation finished.
5. HotSync Manager sends a message to all registered notifiers that the HotSync operation has completed successfully or unsuccessfully.
6. HotSync Manager notes the current time and stores it as the time of the last HotSync operation for the current HotSync user on the desktop.
7. HotSync Manager saves the HotSync log on the handheld and on the desktop for the current HotSync user.
8. Palm OS sublaunches each handheld application whose databases have been modified during the current HotSync operation. It also sublaunches any applications that were just installed that the HotSync operation is finished. An application can perform any action it requires—for example, to sort records—or do nothing.
9. The HotSync client sends notifications to all handheld applications that registered with the Notification Manager for the "sync complete" notification.
10. Finally, if any of the conduits added messages to the log, HotSync Manager displays a notification message to the user, who can then examine those log entries on the desktop.

8

Palm OS Databases

The work of most conduits is to read and write Palm OS® databases on a handheld. This chapter describes the types and layout of databases in the following sections:

Database Overview

A traditional file system first reads all or a portion of a file into a memory buffer from disk, using or updating the information in the memory buffer, and then writes the updated memory buffer back to disk. Because Palm Powered™ handhelds have limited amounts of dynamic RAM and use nonvolatile RAM instead of disk storage, a traditional file system is not optimal for storing and retrieving Palm OS user data. Thus, except when working with expansion media (an SD card, Memory Stick, and the like), Palm OS doesn't make use of a traditional file system. Instead of files, Palm OS applications work mainly with **databases**.

Databases organize related rows (for schema databases) or records (for non-schema databases); each belongs to one and only one database. A database may be a collection of all address book entries, all datebook entries, and so on. A Palm OS application or conduit can create, delete, open, and close databases as necessary, just as a traditional file system can create, delete, open, and close a traditional file. Applications call the **Data Manager** to perform these operations; conduits call the **Sync Manager** on the desktop, which in turn calls the Data Manager on the handheld to perform database operations.

The following subsections compare the different types of databases and their common features:

Schema vs. Non-schema Databases

For those new to Palm OS programming, the term "database" can be somewhat misleading. Palm OS Cobalt supports three different types of database, some of which look more like conventional databases than others. Schema databases bear a strong resemblance to relational databases. Data is organized into tables, which consist of rows and columns. **Schema databases** use the concept of a **schema** to define the structure of a table row. Unlike relational databases, however, schema databases don't allow you to perform joins and other complex operations.

The other two database types are classified as "**non-schema**" databases because they are significantly less structured. There are two supported non-schema database types:

- **Classic databases** are supported for compatibility with earlier versions of Palm OS. All versions of Palm OS back to Palm OS 1.0 support this database format, and this is the format used by applications running on Palm OS Cobalt through PACE.
- **Extended databases** are an "extended" version of classic databases. There are three primary differences between classic and extended databases:
  - records can exceed 64 KB in length (classic records cannot)—in fact, they can be almost 64 MB, if memory is available.
  - uniquely identified by a combination of name and creator ID (classic databases are uniquely identified by name alone)
  - can store data using the processor's native endianness (classic databases must store record data using big-endianness, for compatibility with the 68K-based Dragonball CPU used in the early Palm OS devices).

One of the strengths of the relational approach employed by schema databases is that you can deal with the data as information and, ideally, not worry about the details of how it is represented or physically maintained in the database itself. Having to deal with these kinds of implementation details makes extended and classic databases more difficult to manage.

Handheld applications and conduits that must remain compatible with Palm OS versions earlier than Palm OS Cobalt—perhaps a version of the application exists that runs on earlier versions of Palm OS and this application must be able to work with the earlier version's data—will use classic databases. Those handheld applications and conduits that don't have such a compatibility requirement should use either extended or schema databases instead. Which to use depends on the nature of the application. Schema databases provide a great deal of support for organizing the database contents and for security, at the expense of performance. Extended databases, on the other hand, are faster to read and write, but are less secure and less structured—meaning that your application and conduit have to do the work of maintaining and interpreting record contents themselves.

Non-schema databases treat their contents as lists of mostly opaque records. The Palm OS Data Manager knows just enough about each record to understand category assignment, modification status, and deletion status. Applications are entirely responsible for structuring and interpreting database record contents. Traditional Palm OS applications, written for 68K-based handhelds and for PACE, work exclusively with classic databases.

Schema databases add a layer of abstraction to the record contents. This extra layer of abstraction allows you to create more flexible applications, with improved sharing of data between applications. Because the Data Manager knows more about the structure of the database rows, it can provide more detailed change tracking than for non-schema databases. Changes to a database's schema definitions, row data, and category membership are each tracked independently, making synchronization simpler and more efficient.

Schema databases have other advantages as well:

- They provide more standardized data storage.
- Schema databases can be more easily extended with additional fields.
- It is much easier to create conduits for schema databases, and it is easier to integrate a schema database with a database on the desktop computer or on a server.

Resources and Resource Databases

Non-schema databases that are designated as resource databases tag each chunk of data with a unique resource type and resource ID. These tagged data chunks are called **resources**. Resource databases are almost identical in structure to other non-schema databases except for a slight amount of increased storage overhead per resource record (two extra bytes).

Resources are typically used to store the user interface elements of an application, such as images, fonts, dialog layouts, and so forth. Part of building an application involves creating these resources and merging them with the actual executable code. In the Palm OS environment, an application is, in fact, simply a resource database with the executable code stored as one or more code resources and the graphics elements and other miscellaneous data stored in the same database as other resource types.

Applications may also find resource databases useful for storing and retrieving application preferences, saved window positions, state information, and so forth. These preferences settings can be stored in a separate resource database.

Mutually Exclusive Database Characteristics

Table 8.1 shows which types of databases can contain records or resources and whether they can be secure. Note that database types (schema, extended, and classic) are mutually exclusive, and that a database cannot contain both records and resources. Only schema databases can be secure (see “Secure Databases” on page 132).

Table 8.1 Mutually exclusive database characteristics

Database Type	Can Contain Records or Resources?	Can Be Secure?
Schema	Records (rows) only	Yes
Extended	Either	No
Classic	Either	No

Uniquely Identifying Databases

Schema, extended, and classic databases exist in disjoint namespaces. Because the namespaces are disjoint, it is possible for up to three databases, one per namespace, to have the same name and creator ID.

In all versions of Palm OS, classic databases must be uniquely identified by name. Schema and extended databases, however, are uniquely identified by a combination of the database's name and its creator ID. Thus, schema and extended database names need only be unique for a single creator ID: two such databases with the same name can reside on a single handheld as long as their creator IDs differ.

Database Attributes

In addition to the records that make up the database's contents—and in addition to the schemas that define the structure of the rows in a schema database table—all Palm OS databases have a set of flags that describe various aspects of the database itself, plus a set of dates identifying when the database was created, last modified, and last backed up. As well, non-schema databases have an **application info block** to hold application settings and the like, and a **sort info block** to control the ordering of database records. Schema databases use the concept of cursors for applications to control row ordering; conduits do not have access to cursors.

Schema Databases

Schema databases consist of one or more tables. All of the rows in a given table have the same structure.

All data in a schema database table is represented in the form of two-dimensional tables. A **table** contains zero or more rows and one or more columns. All **rows** in a table have the same sequence of **columns**, but with a different series of values in those columns. Note that a row doesn't have to have a value for a column; the special value `NULL` can be used to indicate that the value is undefined. See Figure 8.1.

Figure 8.1 Schema database example

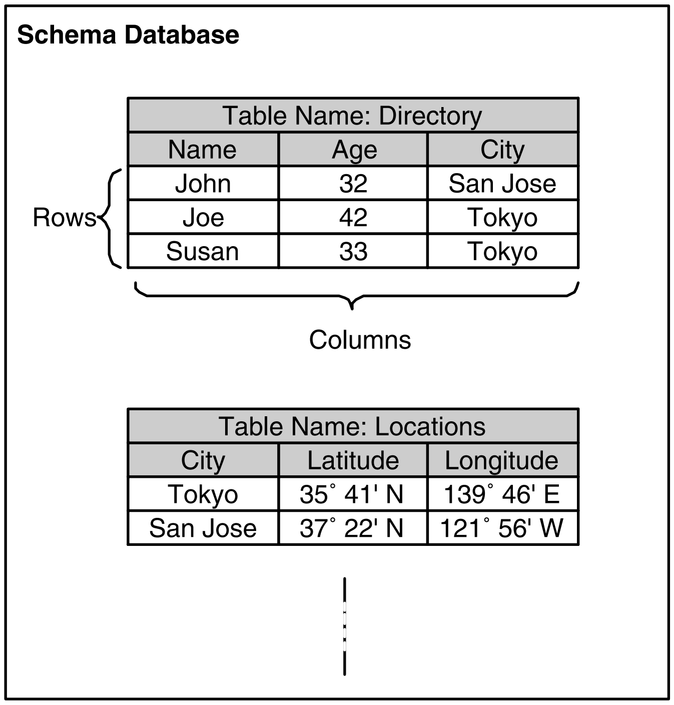


Table Name: Directory		
Name	Age	City
John	32	San Jose
Joe	42	Tokyo
Susan	33	Tokyo

Table Name: Locations		
City	Latitude	Longitude
Tokyo	35˚ 41' N	139˚ 46' E
San Jose	37˚ 22' N	121˚ 56' W

As with a relational database, operations are defined by logic, not by the position of a row within a table. That is, you ask for all rows where (x = 3) and not for the first, third, and fifth rows, for example. The rows of a schema database table are in arbitrary order—the order in which they appear doesn't necessarily reflect the order in which they were entered or in which they are stored.

The following subsections provide further details on schema databases, secure databases, and concurrent access:

Schema Database Header

The schema database header contains information that describes the database and includes pointers to the category info block, row sort indexes, tables, and rows. It contains all of the information in a non-schema database header, except that it lacks the optional sort info block, which is replaced by the sort indexes for each table, and the application info block, which does not exist in a schema database. The schema database header contains the following information:

Name: The null-terminated name of the database. Schema database names should be valid SQL identifiers that are no longer than 32 characters, including the null terminator.

Type: A four-byte string that allows Palm OS to distinguish among multiple databases with the same creator ID. Type values have the same requirements as creator IDs, but you do not register them since they are not unique. Certain types have special meaning—for example `'appl'` is for applications.

Creator: A four-byte string that uniquely identifies the creator of an application or database. You must register your creator ID with PalmSource to protect your application from conflicting with others. Values are case-sensitive and composed of ASCII characters in the range 32-126 (decimal). Values consisting of all lowercase letters are reserved for use by PalmSource.

Attributes: Flags that indicate such characteristics as whether the database contains records or resources, is read-only, has a modified application info block, should be backed up, is copy-protected, is open, and so on.

Version: An application-specific version number.

Creation Date: The date on which the database was created.

Modification Date: The date on which the database was last modified, either by an application or conduit.

Last Backup Date: The date on which the database was last backed up.

Modification Number: An integer that is incremented every time a record in the database is deleted, added, or modified.

Unique ID Seed: A value assigned and used only by the Data Manager to generate record IDs for this database.

DRID: The database reset identifier. The Data Manager pseudo-randomly generates this identifier for each schema database for synchronization use only. The desktop Sync Manager uses the DRID to determine whether the sync clock value and change counters are no longer dependable—for example, when the database has been restored after a hard reset.

Sync Clock: Each schema database has a local sync clock that is used to update the change counter. At the end of a sync operation, the Data Manager increments the sync clock.

Category Info: A pointer to the category info block. This area is not directly accessible. Conduits access category information only via the Sync Manager.

Table Definitions: A pointer to a list of all the tables defined in the database.

Default Sort Index: A pointer to a list of row pointers that indicate the default sort order of rows in a table. This is the only sort order used by the schema Sync Manager.

Sort Indexes: A pointer to an array of sort indexes that can be used only by the handheld application. Conduits have no access to these sort indexes.

Number of Tables: The number of tables stored in the database.

List of Tables: Each entry points to a table. Each table has a list of row IDs. Each row has attributes, category membership, and so on, and points to a list of its column values.

Schemas and Tables

A **schema** is simply the collective definitions of a table's columns. Each schema database can be heterogeneous in that it can support multiple tables. Because each table's definition includes the column definitions for that table—the schema—two tables can have the same schema, yet changes to one table's schema do not affect the other.

Tables can be defined at the time a database is created, or added later.

Schema access is gated by the access restrictions for the database. Read-only access to a database implies read-only access to all of that database's schemas (and thus any attempt to modify the schema will fail). See "Secure Databases" on page 132 for more information on database access restrictions.

Logical (External) vs. Physical (Internal) Views

Schemas allow the Sync Manager to decouple the logical (external) view of your data from the physical (internal) view. When working with a schema database you manipulate row data in terms of data types defined in the column property sets—this is the **logical data view**. In actual fact, however, the Data Manager stores row data internally in an unpublished variant format: the **physical data view**. This decoupling facilitates changes to internal data formats without affecting existing database consumers.

Data types defined in column property sets are Palm OS primitives or their vectors. The Data Manager converts between its physical data types and the logical data types that are enforced during field get and set operations.

Column Properties

A schema is a collection of column property sets. Each column property set contains the following **built-in properties**:

ID: A 32-bit application-defined identifier. This ID must be unique for a given table.

Name: An application-defined name for the column. The column name must be unique for a given table. It can be up to 32 bytes in length, including the terminating null character, and must be a valid SQL identifier consisting only of 7-bit ASCII characters. The column name is stored in a single application-defined language encoding.

Data Type: The type of data contained within the database column.

Size: The maximum size, in bytes, for columns that contain variable-length strings, blobs, and vectors.

Attributes: A set of flags that indicate whether the column data can be modified, whether the column was added to the table after the table was created, and whether or not the column data will be synchronized. (Modifications made to a "non-syncable" column's data don't change the modification state for the row, and thus by themselves don't cause the row to be flagged as modified.)

These built-in column properties are provided by the Data Manager and cannot be removed. In addition to these built-in properties, you can define custom properties for a column: properties that facilitate application-specific semantics for columns. For more information on manipulating the column definitions that make up a schema, see "Working with Column Definitions" on page 126.

Column Data Types

Schema databases support the column data types listed in Table 8.2.

Table 8.2 Supported schema column data types

Palm OS Primitive/ Logical Types	Description	Storage Requirement	Range/Size
`uint8_t`	Unsigned char	1 byte	0 to 255
`uint16_t`	Unsigned short int	2 bytes	0 to 65535
`uint32_t`	Unsigned int	4 bytes	0 to 4294967295
`uint64_t`		8 bytes	
`int8_t`	Signed char	1 byte	-128 to 127
`int16_t`	Signed short int	2 bytes	-32768 to 32767

Table 8.2 Supported schema column data types *(continued)*

Palm OS Primitive/ Logical Types	Description	Storage Requirement	Range/Size
`int32_t`	Signed int	4 bytes	-2147483648 to 2147483647
`int64_t`		8 bytes	
`float`	Float	4 bytes	
`double`	Double	8 bytes	
`Boolean`	True /False value	1 byte	0 or 1
`DateTimeType`	Date-Time type	14 bytes	
`DateType`	Date expressed as an absolute date	2 bytes	
`TimeType`		2 bytes	
`time_t`	(`dbDateTimeSecs`) Time in seconds since the UNIX epoch	4 bytes	-2147483648 to 2147483647
`char`	Fixed-length character string	*m* bytes, where m is the statically-defined length and 1 <= *m* <= 255	1 <= *m* <= 255, where *m* is the maximum defined length.
VarChar	Variable-length character string	*n*+4, where *n* is the actual string length and where *n* <= *m*. *m* is the maximum defined length and 1 <= *m* <= 2^{32}	1 <= *m* <= 2^{32}, where *m* is the maximum defined length.

Table 8.2 Supported schema column data types *(continued)*

Palm OS Primitive/ Logical Types	Description	Storage Requirement	Range/Size
blob	Variable-length array of bytes.	n+4, where n is the actual string length and where $n <= m$. m is the maximum defined length and $1 <= m <= 2^{32}$	$1 <= m <= 2^{32}$, where m is the maximum defined length.
Vector	Variable-length vectors of Palm primitive numeric, string, and date-time types. See Table 8.3, below, for a list of supported vector types.	n+4, where n is the number of bytes needed to contain the vector.	2^{32} bytes.

Table 8.3 Supported vector types

Vector Types	Usage
`uint8_t` vectors	`uint8_t[]`
`uint16_t` vectors	`uint16_t[]`
`uint32_t` vectors	`uint32_t[]`
`uint64_t` vectors	`uint64_t[]`
`float` vectors	`float[]`
`double` vectors	`double[]`
`Boolean` vectors	`Boolean[]`
`DateTimeType` vectors	`DateTimeType[]`

Table 8.3 Supported vector types *(continued)*

Vector Types	Usage
`DateType` vectors	`DateType[]`
`TimeType` vectors	`TimeType[]`
String vectors	Array of null-terminated strings, with an extra terminating null character marking the end of the vector. For instance, using 7-bit ASCII: `"String1\0String2\0String3\0\0"`

NOTE: In a string vector, the null characters must be interpreted as encoding-dependent null characters instead of null bytes. A null character may be multi-byte for a specific encoding scheme.

Database, Table, and Column Identifiers

Schema databases are uniquely identified by a combination of their name and their creator ID.

Database tables are identified by name. There is no need for a numeric "table identifier." Table names should be valid SQL identifiers up to 32 bytes in length, including the terminating null character.

A column is uniquely identified by either the column's descriptive name or by a 32-bit ID (both must be unique). These application-defined column names and IDs allow multiple applications within a given application context to share a common semantic understanding of a given column type. For instance, two applications might select a name of "EMNO" for the employee number column of the "EMPLOYEE" database and use column-based search and retrieval of values in the column named "EMNO". The design-time specification of both column identifiers and table names facilitates the development of public metadata interfaces for databases and encourages generic data exchange based on these interfaces.

Creating, Modifying, and Deleting Tables

You can create tables either at the time you create a database or after the fact. Each table definition specifies the table's name and an array of column definitions.

You can remove a table from a database only if the table contains either no rows or only rows whose deleted bit is set. If the table contains non-deleted rows, delete or remove them first. Once the table is empty (it can still contain rows that are marked as deleted), you can remove the table from the database.

When modifying an existing table, you are limited to adding columns and modifying custom column properties.

Working with Column Definitions

Each table maintains a list of column definitions, each with a column ID unique within the table. This ID is necessary to work with individual columns, but isn't needed to obtain the complete set of column definitions that make up a schema.

In addition to any custom properties you define for a column definition, all columns have a set of built-in properties. These built-in properties are read-only, to prevent applications from modifying existing data row columns in a way that can impact other data consumers. The following are the built-in properties for a column:

- Name (must be unique)
- Data type
- Size (maximum byte size for variable-length strings, blobs, and vectors)
- Attributes

Unlike the built-in properties, custom properties may be read, written and deleted. Custom property IDs must fall outside the built-in property ID range.

For a given column, the schema Sync Manager allows you to define custom properties, if the specified property ID does not exist. If the specified property ID exists, its value is updated to the new value. You can also retrieve the value of a custom or built-in property, but you can remove only custom properties. If you remove a custom column property, only the property is removed, not the column values.

Row Attributes

Schema database rows can have the attributes listed in Table 8.4.

Table 8.4 Schema database row attributes

Attribute	Description
Archive	The row's data is preserved until the next HotSync. When the archive bit is set, the delete bit is set as well, so archived rows are otherwise treated like deleted rows.
Delete	The row has been deleted.
Read-only	The row is read-only, and cannot be written to.
Secret (private)	The row is private.

NOTE: Neither the Data Manager on the handheld nor the schema Sync Manager places any semantics on the read-only attribute. It is up to the application or conduit to enforce the read-only semantics.

The read-only attribute is used to support certain record sharing scenarios that allow a user to view a record, but not to modify it. Note that schemas also allow the definition of "always writable" columns that allow particular fields to be writable in a read-only row. This might be used, for example, in a calendar event for a TV show that is read-only (you can't reschedule the show); the field containing the alarm information would be "always writable" allowing each user the option of setting an alarm.

Categories

Categories are a user-controlled means of grouping or filtering records or rows. Non-schema databases allow records to be a member of only one of 15 categories, or "Unfiled." Schema database rows, on the other hand, can be a member of any combination of up to 255 categories (or none—the equivalent of "Unfiled"). Thus, where in an extended database a record might, say, have to either fall into the "Personal" or "Business" category, in a schema database a row could fall into both.

As with non-schema databases, category information is local to a database. However, unlike non-schema databases which store information about that database's categories in the application info block, schema databases rely upon an internal "category info" block to contain this information.

Information about the database's categories, such as the number and names of the categories, as well as the order in which they occur in a UI list, is controlled by the Category Manager on the handheld. The Data Manager is responsible only for managing the category membership of individual database rows. The Sync Manager on the desktop enables conduits to access this category information.

Category membership for a row is limited to the maximum number of categories that can be defined locally in a schema database. Because the maximum number of categories a database can support is limited to 255, any given row can only be a member of up to 255 categories.

In a non-schema database, records are always in one category ("Unfiled" is just a specific category). In a schema database, rows may be in one category, multiple categories, or none. The notion of "Unfiled" as a category doesn't make sense here, because rows shouldn't be able to be in the "Unfiled" category and in other categories at the same time. Because applications and conduits can perform other operations on rows with no category membership, a row that is a member of no database categories could be thought of as "Unfiled." Note that the Category Manager on the handheld controls how rows with no category membership are displayed to end users.

The Data Manager stores category local IDs as category membership information for a record or row. Storing category local

IDs abstracts the Data Manager from any modifications performed on the internal category structure, such as adding or deleting a category.

The schema Sync Manager enables conduits to manipulate categories or a row’s category membership in the following ways:

- Set and get the category membership for single or multiple rows.
- Add, delete, and modify category names.
- Get the number of categories a row is a member of.
- Remove a row from a category.
- Retrieve the number, names, and IDs of categories.

Also, the schema Sync Manager lets conduits manipulate rows that meet the given category membership criteria:

- Get the number of all rows whose category membership matches the specified category membership criteria.
- Delete all rows with category membership matching the specified category membership criteria.
- Replace one or more categories with the specified category for all rows, depending on the given match mode criteria.
- Remove category membership in the specified categories from all rows in the database, depending on the match mode criteria.
- Determine whether a row is a member of the specified categories, depending on the given match mode criteria.
- Get a list of all the modified rows whose category membership matches the specified category membership criteria.

Application Info Block

Schema databases don’t have a dedicated application info block. For application-specific data of the type found in a non-schema database’s application info block, create a database table specifically for this purpose.

Schema Database Rows

As discussed in "Schemas and Tables" on page 121, a schema database table can have zero or more rows, and each row within the table shares a common structure, or schema.

Rows are identified by a 32-bit identifier that is unique within the database. You supply the row ID when reading, writing, deleting, and removing rows. In the rare instance that you find yourself with a row ID independent of the table from which it came, you can call a schema Sync Manager function to determine to which table the row belongs.

Creating New Rows

The schema Sync Manager allows you to create a new row and add it to a table; you can't add a row to a database without adding it to a table. Assuming that the row was added to the table successfully, the schema Sync Manager returns the row ID of your new row. Optionally, you can add an "empty" row by not supply any data.

NOTE: Row IDs are assigned to new rows by the Data Manager on the handheld; conduits do not assign or change row IDs. The only exception is during a restore operation, which happens during the HotSync operation after a handheld has been hard reset.

Rows added to a table are added to the end of the database. You aren't given the opportunity to specify the position of the row within the table. The schema Sync Manager APIs also don't include a function for altering the position of a row within a table. That is because when applications work with schema database rows, they often are working within the context of a cursor, within which applications can perform such operations. The schema Sync Manager does not provide conduits access to cursors.

Reading and Writing Data

Columns in a row are identified by a 32-bit application-defined ID. The Data Manager also identifies them by an index, but the schema Sync Manager enables conduits to access them only by ID.

The schema Sync Manager provides the means to read and write:

- a partial column value specified by column ID, offset from the beginning of the column value, and the number of bytes to read (C/C++ Sync Suite only)
- one or more complete column values specified by column ID
- all the column values in a row specified by record ID
- multiple rows in a single call

Reading and writing partial column values is useful for columns containing large strings or blobs where, for space efficiency it makes sense to read or write only a portion of the column's data.

Deleting Rows

When a user "deletes" a row in a schema database on the handheld, the row's data chunk is freed (effectively destroying the data), the local ID stored in the row entry is set to 0, and the delete bit is set in the row's attributes. When the user "archives" a row, the delete bit is set, but the data chunk is not freed and the local ID is preserved. This way, the next time the user performs a HotSync operation, a conduit can quickly determine which records to remove (because their row entries are still around on the handheld). "Remove" in this sense means that the row's data chunk, if present, and row entry are disposed of. In the case of archived rows, a conduit can save the row data on the desktop before it permanently removes the row entry and data from the handheld. For deleted rows, the conduit just has to delete the same row from the desktop before permanently removing the row entry from the handheld.

To clarify the use of terminology in the schema Sync Manager APIs:

Delete: Set the row's delete bit, retain the row's entry in the header, and dispose of a row's data. Usually only handheld applications delete rows, but a conduit can delete rows also.

Remove: Dispose of a row's data and its entry in the header. This leaves no trace of the row. Either an application or a conduit may remove rows.

Archive: Set the row's delete bit and retain the row's data and entry in the database header. Only handheld applications archive rows. This allows a conduit the opportunity to transfer the row data to the desktop before removing it from the handheld.

Purge: Means the same as "remove" but removes all rows with their delete bit set. This is what a conduit usually does with a handheld database after it has synchronized it. Note that this usage differs somewhat from the terminology used in classic and extended Sync Manager APIs.

Maintaining Row Order

The handheld application is responsible for determining the order of rows in a schema database. An application can define a default sort index and multiple sort indexes for a table in a schema database, but a conduit can specify only the default sort index when it works with a table. Therefore when a conduit reads multiple rows, they are retrieved in the order specified by the database's default sort index.

Secure Databases

Some applications need to create secure databases that restrict access to the database. The Data Manager supports the creation of secure, schema databases that are protected by application-defined access rules, which are also known as **rule sets**.

For more about creating and defining access rules for secure databases, see *Exploring Palm OS: Memory, Databases, and Files*.

Synchronizing Secure Databases

The Data Manager restricts access to a secure database to only those applications and users authorized by the database's access rules. During a HotSync® operation, the HotSync client on the handheld uses Data Manager functions to access the handheld databases on behalf of the conduits running on the desktop. The HotSync client application must be able to access secure databases that need to be synchronized or backed up.

For an application to ensure that its secure database is syncable, it must modify the database access rules so that the HotSync client has special "bypass" access using the `AzmLibSetBypass()` function (see *Exploring Palm OS: Security and Cryptography*). When the HotSync client is given bypass access, any conduit on the desktop is able to access the database (the HotSync process does not provide a way to restrict access on a per-conduit basis). The bypass access must be made for each action needed. Because you can grant the HotSync client bypass access for each action separately, you can, for example, give the HotSync client read access, but not write or delete access.

If the HotSync client is not given bypass access, it is subject to the normal access rules as defined by the application. For example, if an application defines the access rules for its database so that only signed applications have access (read, write, or delete), during a HotSync operation the database isn't syncable because the HotSync client doesn't have the proper signature required to access the data. Therefore to allow syncing of the database the application must give "bypass" access to the HotSync client, which essentially grants access both to the HotSync client and to any properly-signed application.

The HotSync client on the handheld maintains a notion of **trusted desktops**. The HotSync process allows syncing or backing up of secure databases to only trusted desktops, unless the user has not set a password (user token) on the handheld. Whenever a user with a password set performs a HotSync operation with a desktop for the first time, the HotSync client on the handheld prompts the user to indicate whether the user trusts the desktop they are synchronizing with. If the user does not trust the desktop, then conduits on that desktop can access only databases that are not secure. During a HotSync operation, conduits can call the Sync Manager to determine whether the handheld trusts the desktop. Conduits that need to access secure databases can use the trust status to determine what to do when secure databases are unavailable.

For more on trusted desktops, see Chapter 9, "Understanding Trusted Desktops," on page 149.

Creating Secure Databases

A conduit by itself cannot create a secure database even during a HotSync operation at a trusted desktop. There are no Sync Manager APIs to create a secure database from the desktop. A secure database can be created only by an application on the handheld.

However, a conduit can work around this restriction by using the C `SyncCallDeviceApplication()` or COM `CallDeviceApplication()` API to call an application on the handheld. This target application can then create the secure database and assign appropriate access privileges. The target application should ensure that it assigns appropriate privileges to itself before accessing the secure database; otherwise it could cause UI to pop up in the middle of the HotSync operation (because the application is sublaunched and does not benefit from the sync bypass rule).

Because using this API causes the HotSync client on the handheld to locally sublaunch the target application, the application context that gets implicitly passed in is the one for the HotSync client. Moreover, the HotSync client turns off its own access to all secure databases (via the bypass rule) before sublaunching the target application. (Otherwise, this would be a security breach, because the sublaunched application could then access any secure database accessible by the HotSync client.) Therefore the target application sublaunched in this way can create a secure database and set up the appropriate security privileges for the target application to access the database outside of a HotSync operation, but it cannot open and read/write the secure database it created. Neither can the target application access other databases that it previously secured with an application fingerprint or signature-based token. This restriction may not be a problem, if you only need the sublaunched target application to create a secure database, which the conduit can then read/write like any other secure database using the Sync Manager API. But if you need the sublaunched target application itself to write data in the secure database it creates during a HotSync operation, it can do the following:

1. Create the secure database.
2. Temporarily assign read/write privileges to all or only to the sync application, whether it is the HotSync client or another

sync application. (The target application doesn't need to hardcode the sync application's token fingerprint. This can be discovered and set dynamically. Hence, the target application is not bound to any single sync solution.)

3. Write data to the database as needed.
4. Assign the permanent privileges—that is, create and set a token based on the application's fingerprint or its digital signature.

For more information about security in Palm OS Cobalt, see *Exploring Palm OS: Security and Cryptography* and *Exploring Palm OS: Memory, Databases, and Files*.

Backing up Secure Databases

When a secure database is backed up to the desktop, it is sent to the desktop in encrypted form and is saved on the desktop encrypted. During a backup operation the Data Manager encrypts the data so that only the Data Manager can decrypt it. This differs from a sync operation; when data is sent to the desktop during synchronization it is sent "in the clear"—it is not encrypted.

The desktop Sync Manager provides functions that conduits can use to back up and restore both classic as well as secure and nonsecure schema databases. But only the handheld Data Manager can encrypt and decrypt secure databases. Therefore the desktop Sync Manager includes functions that conduits can use to backup and restore the security data that the Data Manager requires to decrypt secure databases after they are restored to the handheld, say after the handheld is hard-reset.

Note that the Sync Manager can restore only the secure databases that it backed up to the desktop. It is not possible to create secure databases on the desktop and install them on the handheld. Only the Data Manager on the handheld can create secure databases.

Concurrent Schema Database Access

Consider two types of concurrent access to schema databases:

- *Two* entities accessing *one* database concurrently
- *One* entity accessing *two or more* databases concurrently

Two Entities, One Database

When you open a non-schema database with write access, you have exclusive access to that database: no one else can open that database while you have it open, even if they are just opening it with read access. Or, when you open a non-schema database with read access, no one else can open that same database with write access. This can be somewhat restrictive: on a communicator-style device, for example, if you are editing a record in the address book when the phone rings, the phone application running in another process couldn't open the address book in order to perform a caller-ID lookup.

Schema databases don't have this problem because they support concurrent access to a single database. Note that schema databases don't support concurrent write access: *only one writer and multiple readers are allowed.*

Examples of entities accessing the same database concurrently are a single conduit that runs in multiple threads and needs to open the same database in more than one thread, and processes on the handheld that access the same database at the same time a conduit does. Therefore the Data Manager enforces the same concurrent access rules for conduits as it does for applications.

When opening a schema database you specify a **share mode** in addition to an access mode (or open mode). The following share mode constants are supported for schema databases. Only one share mode can be specified when opening a database.

None: No one else can open this database.

Read: Others can open this database with read access.

Read/write: Others can open this database with read or write access.

Concurrent write access to the same database is not supported. That is, specifying an access mode of read/write and a share mode of read/write is not supported; an error is returned if you attempt to open a database with this combination of access and share modes.

Table 8.5 shows all of the allowed combinations of access modes and share modes and identifies which combinations can be used together (those that are marked **Yes**).

Table 8.5 Allowable concurrent access/share mode combinations

	Mode=R Share=None	**Mode=R Share=R**	**Mode=R Share=R/W**	**Mode=R/W Share=None**	**Mode=R/W Share=R**
Mode=R Share=None	No	No	No	No	No
Mode=R Share=R	No	**Yes**	**Yes**	No	No
Mode=R Share=R/W	No	**Yes**	**Yes**	No	**Yes**
Mode=R/W Share=None	No	No	No	No	No
Mode=R/W Share=R	No	No	**Yes**	No	No

When sharing is enabled (that is, when the database is opened with shared read or shared read/write), the Data Manager server synchronizes access to the database. The synchronization is done at the database level. Each Data Manager (and hence Sync Manager) call is atomic, thus providing data integrity at the function level. Because the Data Manager doesn't support multiple applications writing to the same database, it doesn't have to deal with issues around concurrent updates.

Guidelines for Sharing Access to a Schema Database

Because concurrent access is allowed, you must be careful how you open a database so that you do not unintentionally block another process from accessing the same database. For example, a phone application requires access to the Address Book database to look up caller ID information for an incoming call. At the same time, a

HotSync operation could be in progress, during which a conduit needs to read/write the same database to synchronize it with the desktop.

To prevent one process from unintentionally locking out database access to another process, you should follow these guidelines when opening a database:

1. Use the "Share None" mode only when you need exclusive access to the database—that is, when you do not want any other application to have even read-only access to the database.
2. Use the "Share Read" mode when you can allow another application to have read access to the database. For example, a conduit should open the Address Book database in "Open Read/Write" and "Share Read" modes, so that the conduit can read and modify the database, and at the same time allow an application such as the phone to open the Address Book database in "Open Read Only" and "Share Read/Write" modes so that it can look up caller information.
3. Use "Share Read/Write" mode when you need to allow other processes (such as PIM applications or the HotSync client through conduits) to have both read and write access to the database. With this share mode, you must use "Open Read Only" mode.

One Entity, Two or More Databases

The schema Sync Manager also supports *one* entity accessing *two or more* schema databases concurrently. Therefore a conduit can have multiple schema databases open at the same time. (The Sync Manager enforces a limit of only one open non-schema database at a time.) Your conduit must still close all open databases before your it completes, though.

Change Tracking Services for Schema Databases

The Data Manager on the handheld tracks changes in each schema database in much more detail than in each non-schema database. The most important capability that these extra services enable is that a conduit can fast-sync schema databases at multiple desktops in the most common synchronization scenarios; whereas a conduit always slow-syncs non-schema databases when the handheld synchronizes with a different desktop.

When a schema database is first created, the Data Manager stores within the database header a 16-bit sync clock value. The database **sync clock** indicates when this database was last synchronized. At the time of its creation, a database has a sync clock value that starts at 1. Then every time a database is synchronized, the schema Sync Manager increments the database's sync clock by 1 at the end of the HotSync operation, but only if the conduit closes the database with a special "synced all changes" flag.

To provide a conduit with more detailed change information, the Data Manager defines several **sync atoms**, units of information in a schema database for which the Data Manager tracks changes between HotSync operations. The following elements of a schema database are sync atoms:

- table schema definitions
- category definitions
- row column values (not individual column values)
- row category membership

For each sync atom, the Data Manager maintains the following change information:

change counter: When any sync atom is modified, added, archived, or deleted, the Data Manager updates this counter with the current database sync clock value.

purge counter: When all the deleted instances of a sync atom are purged from a database, the Data Manager updates this counter with the current database sync clock value. This counter tracks the relative time of the last purge and applies to the entire database. When a desktop **purges** a database, the deleted sync atoms are completely removed from the handheld. When an out-of-date desktop (a purge was done after its last HotSync operation) synchronizes with the handheld, it must identify the purged atoms by comparing ID lists.

Table schema can have one or more columns with a **non-syncable attribute** set. When a row is modified, its change counter is updated only if the change was made to a "syncable" column. If only non-syncable columns are modified for a row, the schema Sync Manager does not recognize the row as modified when you use any API that works with "modified" rows. However, the schema Sync Manager does allow non-syncable columns to be explicitly read from the desktop.

Each schema database has a pseudo-randomly generated identifier for synchronization use only. The schema Sync Manager generates a new **database reset identifier** (DRID) for each schema database when it is created or restored to a handheld after a hard reset. The DRID enables the schema Sync Manger to identify situations in which the sync clock value and change counters are no longer dependable—in which case, HotSync Manager tells a conduit that it cannot perform a fast sync. For example, when a database is restored after the handheld is hard reset, or when the database sync clock value rolls over. See "Determining the Sync Mode" on page 93 for more information.

The HotSync process uses these Data Manager services to return information via its APIs to conduits about the sync mode in which they should execute and which instances of each sync atom have been modified since the last HotSync operation for a given schema database.

These change tracking services for schema databases provide the following advantages over the simple "dirty" flags available for non-schema databases and records:

- Change counters allow changes to be tracked forever, or until the handheld is hard reset or the counter wraps.
- Deleted sync atoms are tracked until they are purged, which enables fast sync at multiple desktops.
- Purge counters keep track of purges so that a conduit can improve its performance even during a slow sync.

- Categories are managed by the Category Manager (for the Data Manager) and have a change counter, which provides category name change tracking that is not provided for non-schema databases.
- For enterprise applications, change tracking at the column level is especially useful, because it enables a conduit to synchronize only the modified column values between the handheld and a database on a backend server.

Non-Schema Databases

Schema databases impose a structure upon the data, organizing it into tables, rows, and columns. Non-schema databases, on the other hand, impose less overhead and are significantly more flexible. Of course, your application and conduit generally must do more work when dealing with non-schema databases, because they are entirely responsible for interpreting the structure of each record.

Non-schema databases can either be record or resource databases. A **record database** holds application data. Each record can be structured in any fashion that the application desires. **Resource databases** are used to contain executable code, application resources, and the like.

In Palm OS Cobalt, non-schema databases come in two “flavors”: classic and extended. Classic databases are provided for compatibility with previous versions of Palm OS (and with applications running on Palm OS Cobalt through PACE). Because of a couple of long-standing limitations, however, unless your application needs this level of compatibility, it should use extended or schema databases instead. Both classic and extended databases can be either record or resource databases.

Extended databases are very similar to classic databases. Table 8.6 describes the differences.

Table 8.6 Comparison of non-schema database types

Classic Database	Extended Database
Must have record sizes smaller than 64 KB	Can have record sizes up to 4 GB
Are uniquely identified by name.	Are uniquely identified by a combination of name and creator ID.
Data should be stored in big-endian format (for 68K compatibility).	Data can be stored in either big-endian or little-endian format.

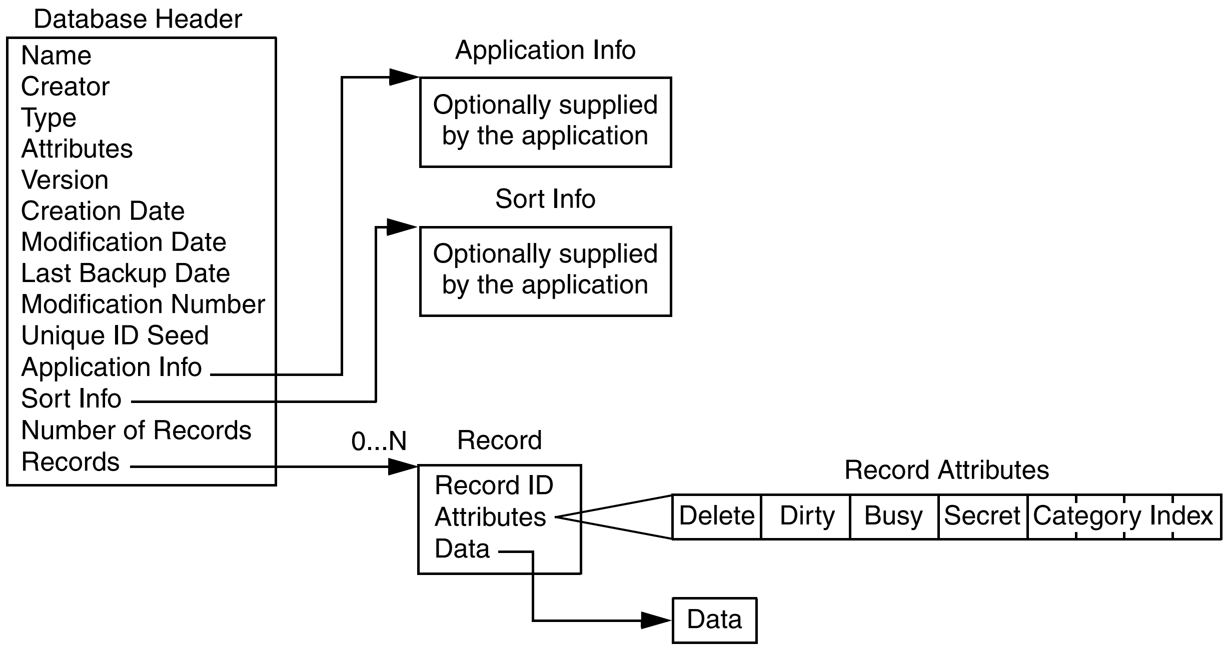
Even though the two non-schema database types are so similar, the desktop Sync Manager provides two separate APIs, each working with databases of only one type.

The following subsections provide further details on non-schema databases:

Structure of a Non-Schema Database

Figure 8.2 illustrates the structure of a non-schema database. The database header contains information that describes the database and includes pointers to the application info block, sort info block, and records. Arrows indicate pointers to data or arrays of data. Each record entry in the header has the record ID, 8 attribute bits, and a 3-byte unique ID for the record.

Figure 8.2 Non-schema database structure



The header of a non-schema database contains the following information:

Name: The null-terminated name of the database. Non-schema database names must consist of only 7-bit ASCII characters from 0x20 through 0x7E. A name can be no longer than 32 characters, including the null terminator.

Creator: A four-byte string that uniquely identifies the creator of an application or database. You must register your creator ID with PalmSource to protect your application from conflicting with others. Values are case-sensitive and composed of ASCII characters in the range 32-126 (decimal). Values consisting of all lowercase letters are reserved for use by PalmSource.

Type: A four-byte string that allows Palm OS to distinguish among multiple databases with the same creator ID. Type values have the same requirements as creator IDs, but you do not register them since they are not unique. Certain types have special meaning—for example `'appl'` is for applications.

Attributes: Flags that indicate such characteristics as whether the database contains records or resources, is read-only, has a modified application info block, should be backed up, is copy-protected, is open, and so on. See "Record Attributes" on page 145 for details.

Version: An application-specific version number.

Creation Date: The date on which the database was created.

Modification Date: The date on which the database was last modified, either by an application or conduit.

Last Backup Date: The date on which the database was last backed up.

Modification Number: An integer that is incremented every time a record in the database is deleted, added, or modified.

Unique ID Seed: A value assigned and used only by the Data Manager to generate record IDs for this database.

Application Info: A pointer to the application info block. This value is optional.

Sort Info: A pointer to the sort info block. This value is optional.

Number of Records: The number of record entries stored in the database header itself, which includes those marked deleted. The number of records exposed to conduits via the Sync Manager is the total number in the database.

List of Records: Each record entry includes the ID of the record, attribute bits (see "Record Attributes" on page 145), and a pointer to the record data. The record ID must be unique for each record within a database. It remains the same for a particular record no matter how many times the record is modified. It is used during synchronization to track records on the handheld with the same records on the desktop.

Record Attributes

Non-schema database records can have the attributes listed in Table 8.7.

Table 8.7 Non-schema database record attributes

Attribute	Description
Delete	The record has been deleted. Only its record ID and attributes remain. Classic database records do not have an Archive bit. Instead the Sync Manager provides an Archive attribute that is set if both the Delete bit is set and if the record data is still present on the handheld.
Dirty	The record has been modified since the last HotSync operation.
Busy	The record is locked by a handheld application for reading or writing.
Secret	The record is private and should be displayed only if the user wishes.
Category Index	The category that the record is in. This is an index from 0 to 15, which references one of the categories defined in the application info block.

Categories and the Application Info Block

Categories are groups of related records in a single database. Applications usually use categories to enable the end user to organize and filter records. Non-schema databases allow records to be a member of only one of 15 categories, or in the “Unfiled” category.

In a non-schema database, the **application info block** can contain both application-specific data and category information. The Sync Manager provides functions for manipulating the application info block. Otherwise, you read and write the application info block in much the same way as you read and write any other non-schema database record.

If you choose to use the handheld's category API in an application that uses a non-schema database, you must set up the database appropriately. The Palm OS category functions expect to find information in a certain format in the application info block. These functions also expect to find in the record's attributes the category that each record belongs to. The Sync Manager provides conduits access to the same category information.

Working with Non-Schema Database Records

A conduit can access non-schema database records either by index or by a record ID that is unique within the database.

Reading and Writing Data

The structure of non-schema database records is application-specific, so the Sync Manager provides the means to read and write records only as opaque data. In addition to reading records directly by index or record ID, you can also iterating through all records, all records in a category, only modified records, or only modified records in a category.

Deleting Records

When a user "deletes" a record on the handheld, the record's data chunk is freed (effectively destroying the data), the local ID stored in the record entry is set to 0, and the delete bit is set in the attributes. When the user "archives" a record, the delete bit is also set but the chunk is not freed and the local ID is preserved. This way, the next time the user performs a HotSync operation, a conduit can quickly determine which records to delete (because their record entries are still around on the handheld). In the case of archived records, a conduit can save the record data on the desktop before it permanently removes the record entry and data from the handheld. For deleted records, the conduit just has to delete the same record from the desktop before permanently removing the record entry from the handheld.

The Sync Manager provides the means to delete one record by ID, all records in a given category, all records with the delete bit set, and all records in the database.

Maintaining Record Order

The handheld application is responsible for determining the order of records in a non-schema database, as in a schema database. Therefore when a conduit reads records by index, they are retrieved in the order that the application has sorted them in. When a conduit creates a new record, the Sync Manager adds it after the last record.

Concurrent Non-Schema Database Access

Consider two types of concurrent access to non-schema databases:

- *Two* entities accessing *one* database concurrently (not supported)
- *One* entity accessing *two or more* databases concurrently

Two Entities, One Database

The handheld Data Manager, and thus the desktop Sync Manager, does *not* support *two* entities accessing *one* non-schema database concurrently. When you open a non-schema Palm OS database with write access, you have exclusive access to that database: no one else can open that database while you have it open, even if they are just opening it with read access. Or, when you open a non-schema database with read access, no one else can open that same database with write access. If you need to be able to share access to databases, then you must use a schema database. See "Concurrent Schema Database Access" on page 136.

One Entity, Two or More Databases

The Sync Manager via its C/C++ API does *not* support *one* entity accessing *two or more* classic databases concurrently. With this API, you can open only one database at a time. You must then close it before opening another.

However, the COM Sync module, which provides a layer between a conduit and the C/C++ Sync Manager API, *does* support *one* entity accessing *two or more* classic databases concurrently. The COM Sync module actually opens and closes databases as needed via the Sync Manager API, effectively freeing the conduit from the restriction of having only one non-schema database open at a time.

9

Understanding Trusted Desktops

The HotSync® client on handhelds running Palm OS® Cobalt provides additional security features beyond those provided by the handheld Data Manager and Sync Manager. The HotSync client introduces the concept of the **trusted desktop**—one that the user designates and is assumed to be trusted by the user for synchronizing and backing up secure databases.

The sections in this chapter are:

Overview

The HotSync client synchronizes and backs up secure databases *only* at trusted desktops. It does so even if the handheld does not have a password (or, in general, a user token) assigned.

By default, a desktop is assumed to be *not trusted*—the user must explicitly state that the desktop is trusted. The exception is when the user synchronizes the handheld for the very first time. See "Imprinting" on page 152.

Therefore, if a user performs a HotSync operation in a non-trusted environment such as a conference kiosk, none of the secure databases are synchronized or backed up.

Establishing Trust with a Desktop

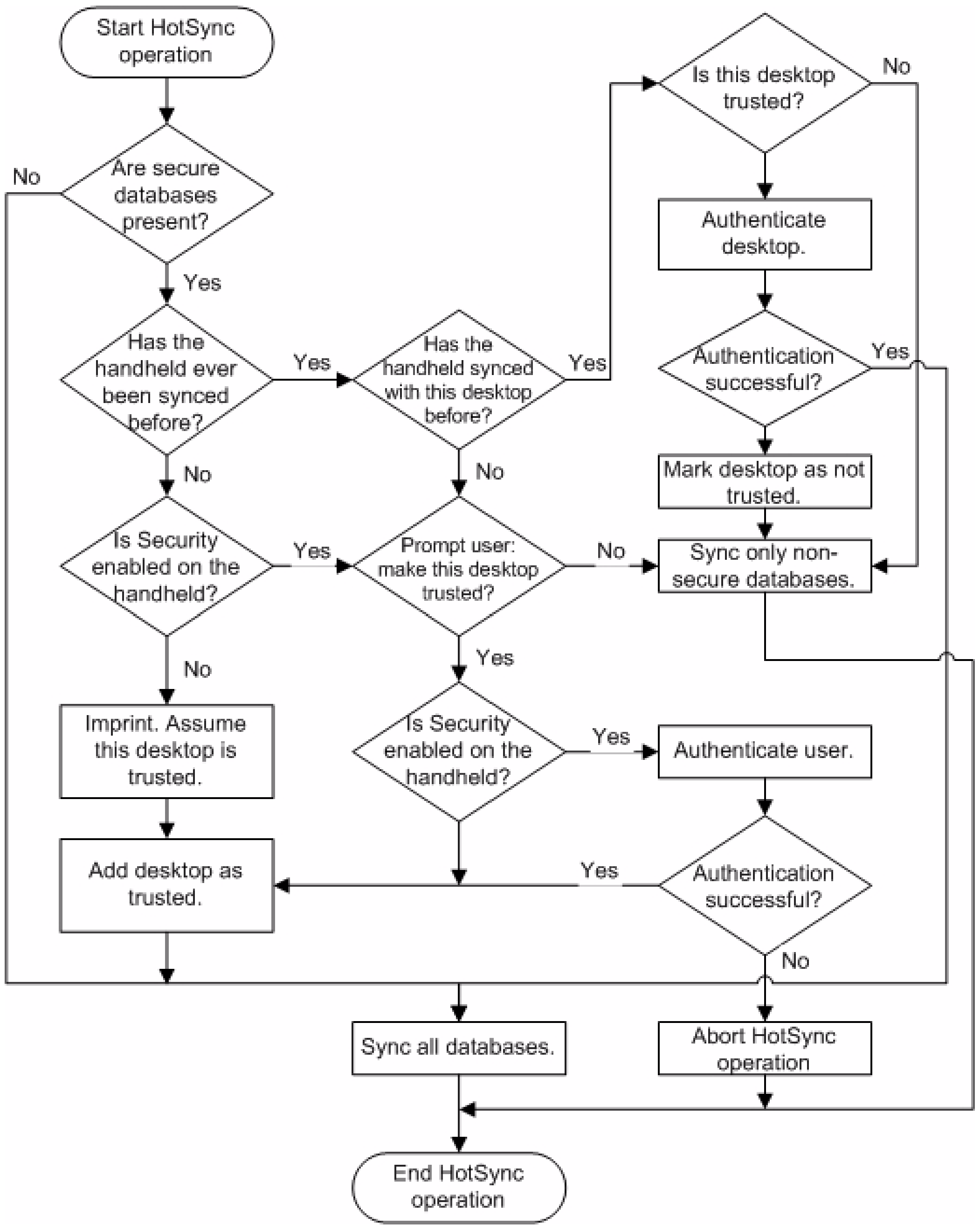
If the handheld's security setting is at its lowest ("None"), the HotSync client "imprints" with the first desktop that the handheld is synchronized with—that is, it marks the first desktop it sees as trusted. (Note that in the minimum-security setting, the handheld may not have a password). Subsequent synchronization sessions with this desktop are always trusted unless the user explicitly marks this desktop as not trusted.

On the other hand, if the handheld's security setting is "Medium" or "High", the HotSync client queries the user whenever it encounters a new desktop. If the user chooses to make the desktop trusted, user authentication is performed.

The HotSync client also retains a list of desktops with which the handheld has been synchronized in the past. The user may choose to view this list and change the trust status of a desktop.

Figure 9.1 on page 151 is a flow chart that shows how the HotSync client determines whether a desktop is trusted.

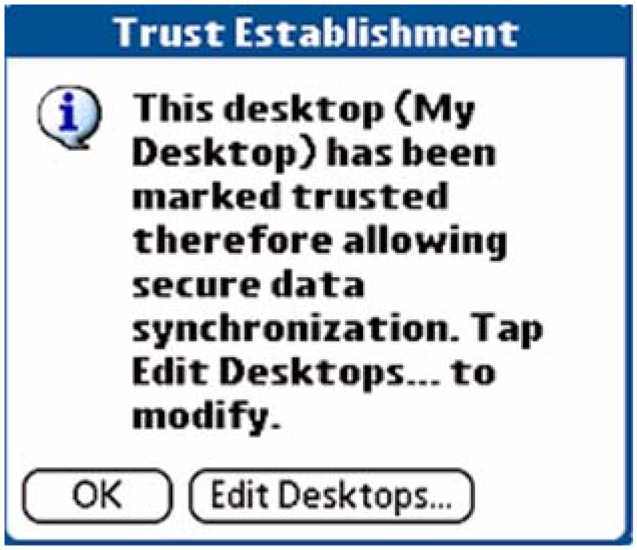
Figure 9.1 Determining whether a desktop is trusted


Start HotSync operation
Are secure databases present?
No
Yes
Has the handheld ever been synced before?
Yes
No
Has the handheld synced with this desktop before?
Yes
No
Is this desktop trusted?
No
Authenticate desktop.
Authentication successful?
Yes
Mark desktop as not trusted.
Is Security enabled on the handheld?
Yes
No
Prompt user: make this desktop trusted?
No
Yes
Sync only non-secure databases.
Imprint. Assume this desktop is trusted.
Is Security enabled on the handheld?
Yes
Authenticate user.
Add desktop as trusted.
Yes
Authentication successful?
No
Sync all databases.
Abort HotSync operation
End HotSync operation

Imprinting

If the handheld has never been synchronized before and if the security setting is "None", the HotSync client marks the first desktop it synchronizes with, as trusted—that is, it "imprints" with the first desktop. Note that this action is done only if the user approves the HotSync client to be a registered sync client. The user is informed of this action at the end of the sync (Figure 9.2). This is the minimum security mode for the HotSync client.

Figure 9.2 Informing user of imprinting with desktop

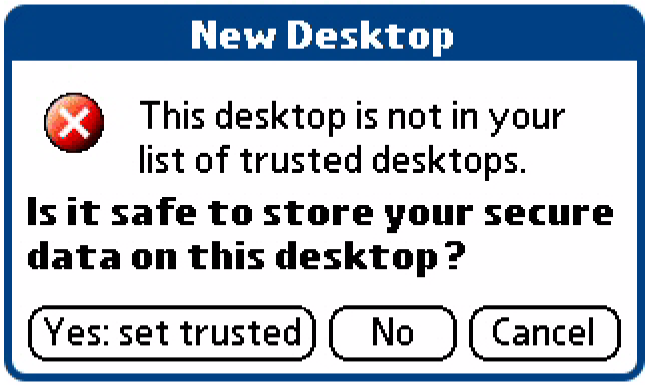

Trust Establishment

This desktop (My Desktop) has been marked trusted therefore allowing secure data synchronization. Tap Edit Desktops... to modify.

Query during the First HotSync Operation with a Desktop

When the handheld encounters a new desktop for synchronization, the user is presented with an alert (Figure 9.3) on the handheld if it has secure databases and if the security setting is "Medium" or "High".

Figure 9.3 Trust new desktop alert



New Desktop

This desktop is not in your list of trusted desktops.

Is it safe to store your secure data on this desktop?

If the user taps **Yes: set trusted**, the desktop is added to the list of trusted desktops. Depending on the handheld security setting (None, Medium, High), the HotSync client authenticates the user. User authentication depends on the type of the user token set up in the system. It could be a password that needs to be entered or a fingerprint that needs to be supplied to the biometric sensor on the handheld. If a password is required, a password prompt is displayed. (Authentication may not be required if the handheld security setting is “None”). Upon successful authentication, the desktop is marked trusted and secure databases are included in HotSync operations with this desktop. On completion, the user is presented with a dialog with which to name the desktop for future reference. In the common case, users never need to deal with this dialog again. However, users who synchronize with multiple desktops need to “trust” those desktops via the same process.

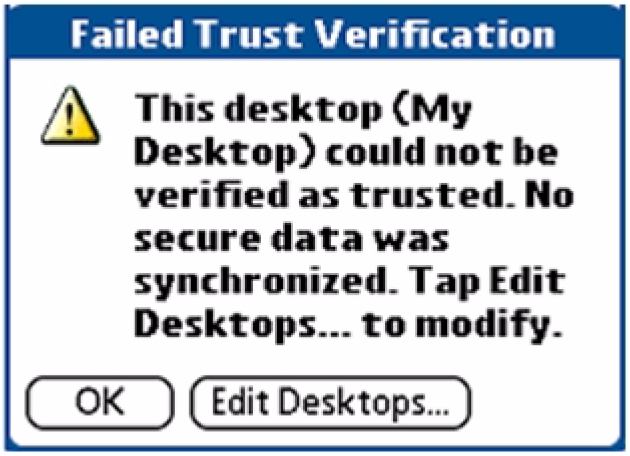
If the user taps **No** in the above alert, the HotSync client proceeds to perform a non-trusted HotSync operation—that is, secure databases are neither synchronized nor backed up at this desktop. Subsequent HotSync operations with this desktop will always be non-trusted, until the user changes the status of this desktop to “trusted”.

Tapping **Cancel** cancels the HotSync operation.

Failed Trust Verification

The HotSync client authenticates any desktop that is marked trusted. If authentication fails, the desktop is marked as not trusted and the HotSync operation does not involve secure databases. At the end of the HotSync operation, the user is informed that verification failed ().

Figure 9.4 Failed desktop trust verification

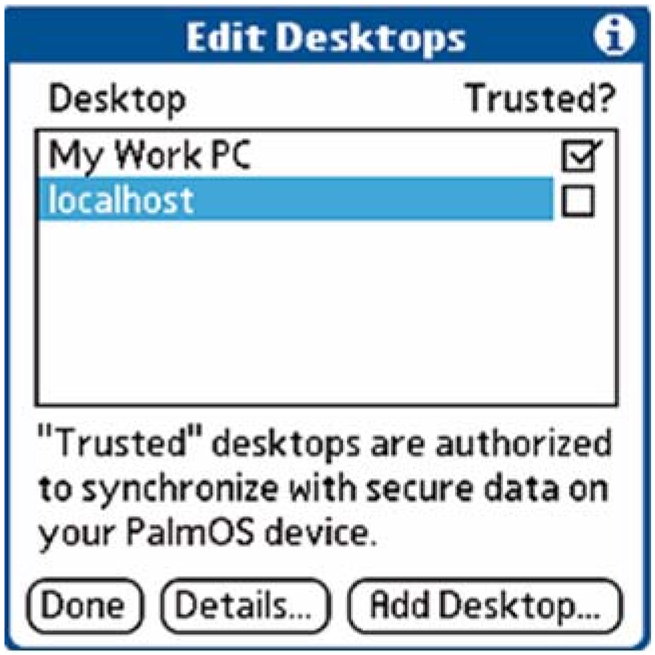

Failed Trust Verification

This desktop (My Desktop) could not be verified as trusted. No secure data was synchronized. Tap Edit Desktops... to modify.

Changing the Trust Status of a Desktop

A user may change desktop computers or otherwise need to edit the list of trusted desktops. Tapping the **Edit Desktop** button in the HotSync client displays the dialog in Figure 9.5.

Figure 9.5 Edit Desktops dialog



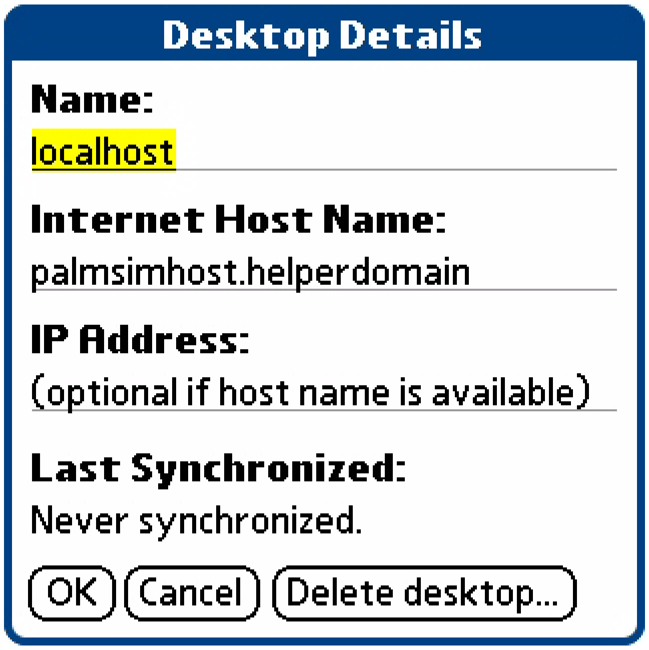
Desktop	Trusted?
My Work PC	✓
localhost	☐

"Trusted" desktops are authorized to synchronize with secure data on your PalmOS device.

The **Edit Desktops** dialog shows a list of all the desktops with which the user has performed a HotSync operation. The list shows trusted desktops first and non-trusted desktops last.

Tapping **Details** allows the user to change the name, Internet host name, and IP address of an individual desktop, or remove the entry altogether (Figure 9.6).

Figure 9.6 Desktop Details dialog


Desktop Details

Name:
localhost

Internet Host Name:
palmsimhost.helperdomain

IP Address:
(optional if host name is available)

Last Synchronized:
Never synchronized.

Trusted Desktop API

A conduit can call a single Sync Manager function to determine whether the handheld it is currently synchronizing with trusts the current desktop. If a conduit tries to access a secure database from a desktop that is not trusted, the call fails with an “access denied” error. A backup conduit should determine whether the desktop is trusted before it tries to backup secure databases.

In the C/C++ Sync Suite, the function to call is `SyncGetDesktopTrustStatus()`. In the COM Sync Suite, the method is `PSDDatabaseQuery`.`GetDeskTopTrustStatus()`.

10 Conduit Development

This chapter provides the background for deciding which development approach best meets your conduit’s requirements and which of the available CDK suites supports that approach.

This chapter describes the following topics:

Choosing a Starting Point

This section lays out the first decisions you need to make and then presents a feature comparison of the CDK’s sync suites.

- First Decisions
- Comparing Key Features

First Decisions

This section outlines the issues you should consider when deciding which sync suite best meets your needs:

1. **Choose whether to use the low-level interface or the Generic Conduit Framework.** As shown in Table 10.2, the C/C++ Sync Suite provides an implementation of the Generic Conduit Framework; all suites provide a lower-level interface that directly or indirectly corresponds to the C-based Sync Manager API.

   Use Table 10.1 to help you compare these starting points based on your design requirements.

Table 10.1 Comparison of Generic Conduit Framework and low-level API

Conduit Design Requirement	Low-level Sync Manager API[1]	Generic Conduit Framework[2]
Interfacing to standard data sources (Access, ODBC, CSV, etc.)		Best
MFC-based application		Good
Cross-platform source code		Best
Fast prototype application development	Good	Good (use Conduit Wizard[3])
Upload/download data transfer	Best	Good
Online/offline transaction processing	Best	

1. In all suites, a lower-level interface is available. In the C/C++ Sync Suite, conduits can access the Sync Manager directly. In the COM Sync Suite, conduits can access the Sync Manager indirectly via the COM Sync module.
2. Available in the C/C++ Sync Suite.
3. Available only in the C/C++ Sync Suite.

2. **Choose a suite.** Table 10.2 provides a high-level comparison of the CDK's sync suites. Use this table and the *Companion* documents that come with each suite to help you decide which best meets your needs (see "Related Documentation" on page viii).

   The main advantages of the different suites are:

   – The C/C++ Sync Suite provides a superset of all features, so that some developers must use it to meet all their requirements.

   – The COM Sync Suite enables you to develop a conduit in any COM-enabled language, including a procedural language like Visual Basic. The COM Sync Suite offers a simple COM interface for your conduit to access several of the underlying C APIs.

Comparing Key Features

Table 10.2 compares the key features of each sync suite. This table does not intend to be a comprehensive list of CDK features. To learn more about each suite, see "Related Documentation" on page viii.

Table 10.2 compares the following features:

- **Desktop Features Supported** lists features that are available via an API in each sync suite. Other features mentioned are important for developing desktop applications or conduit and handheld application installers.
- **IDEs and Development Platforms Supported** lists the integrated development environments that PalmSource, Inc. supports for each suite.
- **Deployment Platforms Supported** lists the platforms on which PalmSource, Inc. supports conduit deployment.

Table 10.2 Feature comparison of CDK sync suites

Suite	COM Sync Suite	C/C++ Sync Suite
Desktop Features Supported		
User Data (or User Manager) API—to access information about HotSync Manager users and expansion cards in handhelds	Yes	Yes
Conduit Manager—for an installer to register conduits with HotSync Manager	Yes	Yes
Notifier DLLs—to tell desktop application when HotSync operation starts/stops	No	Yes
Notifier Install Manager—for an installer to register notifiers with HotSync Manager	Yes	Yes
Install Conduit Manager—for an installer to register an install conduit with HotSync Manager	Yes	Yes
Install Aide—for an installer to install applications in handheld's primary memory or files on expansion cards	Yes	Yes
IDEs and Development Platforms Supported		
Microsoft Visual Studio .NET (Visual C++ and Visual Basic) 2003[1]: Windows 2000 and XP	Yes	Yes
Deployment Platforms Supported		
Windows 98SE, ME, 2000, XP	Yes	Yes
Extras		
Wizard to automatically generate project and starter code	No	Yes

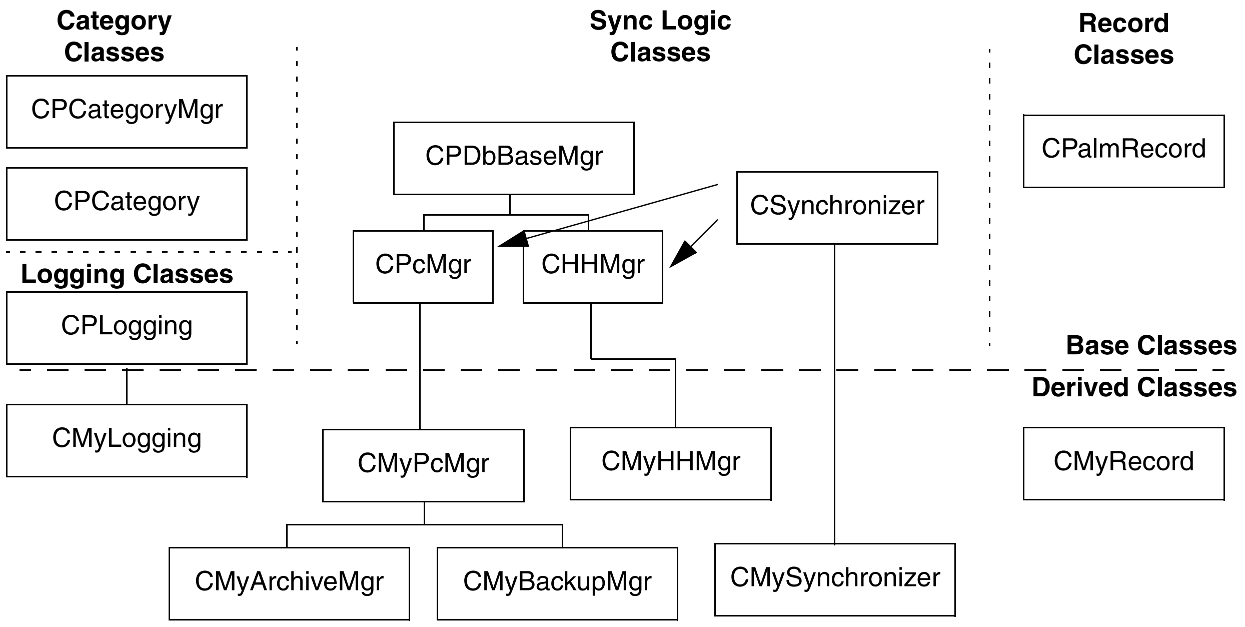
1. Visual Studio .NET 2002 and Visual Basic 6 have been verified to work, but they are not officially supported.

Generic Conduit Framework

Creating a conduit using the Generic Conduit Framework is a matter of deriving certain classes and customizing them for the data formats you wish to use on the handheld or the desktop computer.

Figure 10.1 illustrates the structure of the Generic Conduit Framework. These class names are the C++ Generic Conduit class names. To see the Generic Conduit structure in other suites, refer to the documents for each suite described in "Related Documentation" on page viii.

Figure 10.1 Generic Conduit Framework



The Generic Conduit Framework consists of the following base classes:

- **Category** classes access and manage standard Palm OS® category data.
- **Record** classes convert records between the format on the handheld and the format on the desktop computer.
- **Synchronization logic** classes divided into these groups:

  - **Database Manager** classes store and retrieve records from the handheld or the desktop computer.
  - **Synchronizer** classes synchronize records and the application info block from a database on the handheld. The core synchronization logic is implemented in these classes.
- The **Logging** class is used to print diagnostic information into a log file maintained by HotSync Manager.

The following steps summarize how to implement a conduit based on the Generic Conduit Framework and to customize it for your record format.

- Step 1: Describe your record format.
- Step 2: Implement the storage and retrieval of your records.
- Step 3: Implement the storage and retrieval of a backup database or an archive database.
- Step 4: Implement the conversion of records to/from the generic `CPalmRecord` format.
- Step 5: Implement any synchronization of `AppInfo` data.

These basic steps are the same across all suites, but exact implementation varies. For more information, see the documentation for each suite.

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

1. HotSync Manager calls your `OpenConduit()` function (or in other suites, the method that starts your synchronization process), which must examine the synchronization properties passed to it to determine the type of synchronization (fast sync or slow sync).
2. Your `OpenConduit()` function instantiates a `CSynchronizer` object and performs the following tasks:
   a. Registers the conduit (only in C API-based conduits).
   b. Examines the synchronization properties passed to it to determine the type of synchronization, typically fast sync or slow sync.

c. Creates the Database Manager objects (handheld, desktop, archive, and backup).

d. Synchronizes category information using the provided synchronization logic.

e. Synchronizes application information (`AppInfo` block).

f. Synchronizes records using the Fast Sync, Slow Sync, HH overwrites desktop, Desktop overwrites HH, or No Action.

g. Deletes the Manager objects.

h. Logs results to the HotSync log.

See the sample conduits provided with each suite to follow the execution flow.

Classic Sync Manager API

This section describes the flow of calls from a typical conduit to the classic Sync Manager in the C/C++ Sync Suite. These basic concepts are the same across all suites, but exact implementation varies. For more information, see the documentation for each suite described in "Related Documentation" on page viii.

Calls to the classic Sync Manager fall into the following categories:

- registering the conduit (only in C API-based conduits)
- opening a database
- reading and writing records
- working with record categories
- closing the database
- uninstalling the conduit

The code snippets in this section help briefly demonstrate how a sample conduit, written with the C/C++ Sync Suite, uses classic Sync Manager calls. The usage and primary steps are analogous to those in the other suites.

Registering Your Conduit with Sync Manager

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

NOTE: Be careful not to confuse registering with the Sync Manger and registering with HotSync Manager. Your conduit must register with the *Sync* Manager at run time. You must register your conduit with *HotSync* Manager at install time, otherwise HotSync Manager does not run your conduit (see "Writing an Installer" in each suite's *Companion* document).

Opening Your Database

You can use the `SyncCreateDB()` function to create a new, opened database on the handheld, or you can use the `SyncOpenDB()` function to open an existing handheld database. Listing 10.1 shows a section of code from an example `OpenDataBase()` function. This function calls both the `SyncCreateDB()` and `SyncOpenDB()` functions.

Listing 10.1 Opening a handheld database

```
BOOL OpenDataBase( CLogMgr& theLog, const char * pName, DWORD dbCreator,
   DWORD dbType, BYTE& hDataBase, WORD& dbRecordCount,
   BOOL clearDB )
{
long retVal;
   // Open the remote database.
   retVal = SyncOpenDB( pName, 0, hDataBase );
   if ( retVal )
   {
      if ( retVal == SYNCERR_FILE_NOT_FOUND )
      {
         TRACE( "SyncOpenDB( %s ) failed [ %lX ], attempting to create DB.\n",
            pName, retVal );

         // The database does not exist, so create it.
         CDbCreateDB  createInfo;
         memset( &createInfo, 0, sizeof( CDbCreateDB ) ) ;

         createInfo.m_Creator   = dbCreator;
         createInfo.m_Type      = dbType;
         createInfo.m_Flags     = eRecord;          // eRecord;
         createInfo.m_CardNo    = 0;                // Target card number
         createInfo.m_Version   = 0;

         ASSERT( strlen( pName ) <= DB_NAMELEN );
         strcpy( createInfo.m_Name, pName );

         retVal = SyncCreateDB( createInfo );
         if ( retVal )
         {
            CString theDBName( pName );
            // fatal error
            theLog.Message( IDS_CANT_CREATE_DB, theDBName, TRUE );


            TRACE( "SyncCreateDB( %s ) failed [ %lX ].\n", pName, retVal );

            return FALSE;
         }
         else
         {
            // Database is open with handle createInfo.m_FileHandle.
            hDataBase = createInfo.m_FileHandle;

            TRACE( "SyncCreateDB( %s ) succeeded.\n", pName );
         }
      }
      else
      {
         CString theDBName( pName );
         // fatal error
         theLog.Message( IDS_CANT_OPEN_DB, theDBName, TRUE );

         TRACE( "SyncOpenDB( %s ) failed [ %lX ].\n", pName, retVal );

         return FALSE;
      }
   }
   // If clearDB, then clear all the existing records in the database.
   if ( clearDB )
   {
      retVal = SyncPurgeAllRecs( hDataBase );
      if ( retVal )
      {
         CString theDBName( pName );
         // fatal error
         theLog.Message( IDS_CANT_PURGE_DB, theDBName, TRUE );

         TRACE( "SyncPurgeAllRecs( %s ) failed [ %lX ].\n", pName, retVal );

         return FALSE;
      }
      else
      {
         TRACE( "SyncPurgeAllRecs( %s ) succeeded.\n", pName );
      }
   }

   // Determine how many records are in the remote database.
   retVal = SyncGetDBRecordCount( hDataBase, dbRecordCount );
   if ( retVal )
   {
      if ( hDataBase ) SyncCloseDB( hDataBase );

      CString theDBName( pName );
      // fatal error
      theLog.Message( IDS_CANT_GET_DBRECCOUNT, theDBName, TRUE );

      TRACE( "SyncGetDBRecordCount( %s ) failed [ %lX ].\n", pName, retVal );

      return FALSE;
   }

   return TRUE;
```

```
} // OpenDataBase
```

Working with Databases

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

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

Reading Records from a Handheld Database

As shown in Table 10.3, the Sync Manager provides a number of functions for retrieving records and resources from a database on the handheld. Each of these functions reads the record or resource into an object of the `CRawRecordInfo` class.

You must fill in certain of the raw record object fields before calling each function. The function retrieves the record from the handheld and fills in other fields with the record information. The fields required for each function are described in the documentation for each function.

Table 10.3 Functions for reading records from handheld databases

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

Listing 10.2 shows an example `ImportPrefs()` method that calls `SyncReadRecordByIndex()` to read records by index.

Listing 10.2 Reading records from a handheld database

```
BOOL CNetPrefs::ImportPrefs( BYTE hHHDataBase, WORD recIndex )
{
   BYTE  readBuf[ READ_BUF_SIZE ];
   BOOL  importSuccess = TRUE;

   fHDataBase = hHHDataBase;


   // Read from the HH database at index recIndex.
   CRawRecordInfo rrInfo;

   ::memset( &rrInfo, 0, sizeof( rrInfo ) );

   rrInfo.m_FileHandle  = hHHDataBase;
   rrInfo.m_RecIndex    = recIndex;
   rrInfo.m_pBytes      = (BYTE *) &readBuf[ 0 ];
   rrInfo.m_RecSize     = READ_BUF_SIZE;
   rrInfo.m_TotalBytes  = READ_BUF_SIZE;

   long retVal = ::SyncReadRecordByIndex( rrInfo );
   if ( retVal )
   {
      importSuccess = FALSE;
      fPLogMgr->Message( IDS_CANT_READ_DB );
   }
   else
   {
      // The read succeeded, so copy the data onto fCompressedBuffer.
      fHHDBRecID = rrInfo.m_RecId;

      ASSERT( rrInfo.m_RecSize != 0 );

      if ( fCompressedBuffer != NULL ) delete fCompressedBuffer;
      fCompressedBuffer  = new BYTE[ rrInfo.m_RecSize ];
      fCompressedLength  = rrInfo.m_RecSize;

      ::memcpy( (void *) fCompressedBuffer, (void *) rrInfo.m_pBytes,
         fCompressedLength );
   }

   return importSuccess;

} // CNetPrefs::ImportPrefs
```

Iterating Through a Database

The Sync Manager also provides functions that you can use to iterate through a database on the handheld. To iterate through records in a database, you first reset the current index into the database by calling the `SyncResetRecordIndex()` function, and then repeatedly call one of the following retrieval functions:

- `SyncReadNextRecInCategory()` retrieves the next record in a certain record category.
- `SyncReadNextModifiedRec()` retrieves the next record that has been modified (since the last synchronization).
- `SyncReadNextModifiedRecInCategory()` retrieves the next record in a specific record category that has been modified.

The Sync Manager automatically resets the current database iteration index for a database upon opening the database. To reset the index for subsequent iterations, you must call the `SyncResetRecordIndex()` function.

Listing 10.3 shows a section of code that uses `SyncReadNextModifiedRec()` to retrieve each record that has been modified in the handheld database.

Listing 10.3 Using an iterator function to read handheld records

```
CRawRecordInfo rawRecord;

retval = AllocateRawRecordMemory(rawRecord, EXPENSE_RAW_REC_MEM);
   // Read in each modified remote record one at a time.
while (!err && !retval)
{
   if (!(err = SyncReadNextModifiedRec(rawRecord)))
   {
      // Convert from raw record format to CExpenseRecord.
      if (!m_pDTConvert->ConvertFromRemote(remRecord, rawRecord))
      {
         // Synchronize the record obtained from the handheld.
         retval = SynchronizeRecord(remRecord, locRecord,
            backRecord);
      }
      else
         retval = CONDERR_CONVERT_FROM_REMOTE_REC;
   }
   memset(rawRecord.m_pBytes, 0, rawRecord.m_TotalBytes);
   }
if (err && err != SYNCERR_FILE_NOT_FOUND)
   LogBadReadRecord(err);
```

Modifying a Database While Iterating

The Sync Manager does not support the interleaving of iterating through and modifying a database at the same time, which means that you need to structure your logic not to modify a database while iterating through it. Beginning with version 2.0 of Palm OS, you can safely delete records from a database while iterating through it, with the exception of the `SyncPurgeAllRecsInCategory()` function.

Writing Records to a Handheld Database

The Sync Manager provides several functions for writing records and resources to a database on the handheld. Each of these functions sends record information that you have stored in an object of the `CRawRecordInfo` class. You can use these functions to modify an existing record or to add a new record to a database.

To write a record to a handheld database, fill in certain fields in the raw record object and then call one of the Sync Manager record-writing functions. The function sends the record to a database on the handheld. You can use the `SyncWriteRec()` function to write a record to a handheld record database, and you can use `SyncWriteResourceRec()` to write a resource to a resource database on the handheld. Listing 10.4 shows an example `ExportPrefs()` method that calls the `SyncWriteRec()` function.

Listing 10.4 Writing records to a handheld

```
BOOL CNetPrefs::ExportPrefs( BYTE hHHDataBase )
{
   BOOL  exportSuccess = TRUE;

   fHDataBase = hHHDataBase;

   if ( fCompressedLength != 0 )
   {
      ASSERT( fCompressedBuffer != NULL );

      // write to the HH database.

      CRawRecordInfo  rrInfo;
      ::memset( &rrInfo, 0, sizeof( rrInfo ) );

      rrInfo.m_FileHandle   = hHHDataBase;
      rrInfo.m_RecId        = fHHDBRecID;
      rrInfo.m_pBytes       = fCompressedBuffer;
      rrInfo.m_RecSize      = fCompressedLength;

      long retVal = ::SyncWriteRec( rrInfo );
      if ( retVal )
      {
         exportSuccess = FALSE;
         fPLogMgr->Message( IDS_CANT_WRITE_DB );
      }

   }

   return exportSuccess;

} // CNetPrefs::ExportPrefs
```

Deleting Records in Handheld Databases

When you use a Sync Manager function to delete a record or resource from a database, the record or resource is immediately deleted and removed from the database. This is in contrast to some calls that applications on the handheld make to delete records, which only mark those records for deletion but do not remove them immediately.

NOTE: Classic Sync Manager functions that begin with `Delete` and `Purge` both delete objects *immediately*.

The Sync Manager provides several functions for deleting records or resources from databases. To use these functions, you assign data to specific fields in an object of the `CRawRecordInfo` class. You then call one of the functions described in Table 10.4.

Table 10.4 Functions for deleting single records

Function	Description
`SyncDeleteRec()`	Deletes a record from a record database.
`SyncDeleteResourceRec()`	Deletes a specific resource from a resource database.

You can delete multiple records by calling one of the functions described in Table 10.5.

Table 10.5 Functions for deleting multiple records

Function	Description
`SyncPurgeDeletedRecs()`	Removes all of the records in a record database that have previously been marked as deleted or archived.
`SyncPurgeAllRecs()`	Removes all records in a record database on the handheld.

Table 10.5 Functions for deleting multiple records *(continued)*

Function	Description
`SyncPurgeAllRecsInCategory()`	Removes all of the records in a specific category from a record database on the handheld.
`SyncDeleteAllResourceRec()`	Removes all resources from a resource database.

Cleaning Up Records

You can use the function shown in Table 10.6 at the end of your synchronization operations to clear the record status flags.

Table 10.6 Functions to clean up records

Function	Description
`SyncResetSyncFlags()`	Resets the modified flag of all records in the opened record database on the handheld. Resets the backup date for an opened record or resource database.

Closing Your Database

After you have finished with a handheld database, you must call the `SyncCloseDB()` function to close it, as shown in Listing 10.1. C API-based conduits allow only one non-schema database to be open at any time. COM-based conduits can open more than one database at once.

IMPORTANT: If you open a non-schema database in a C API-based conduit, you must close it before exiting your conduit; otherwise, other conduits will not be able to open their databases.

A

Configuration Entries

The **configuration entries** are name/value pairs that the Conduit Manager and HotSync Manager for Windows store for conventionally registered conduits, notifiers, and for HotSync Manager itself. The interfaces provided in the C/C++ and COM Sync Suites enable you to read and write these values.

IMPORTANT: Do not manipulate configuration entries directly in the Windows registry. Instead always use the APIs documented in the CDK to access configuration entries. The storage of configuration entries may change in the future, but PalmSource will ensure that the documented APIs continue to work. If you directly access the registry, your code will likely break in the future.

For details on how to use the APIs in each sync suite to access configuration entries, refer to the "Writing an Installer" chapter in either the *C/C++ Sync Suite Companion* or the *COM Sync Suite Companion*.

The configuration entries are divided into these groups, as described in the following sections:

Conduit Configuration Entries

The **conduit configuration entries** are name/value pairs that the Conduit Manager for Windows stores for each conventionally registered conduit. These are written at the time you register your conduit. The Conduit Manager API in the C/C++ Sync Suite and the `PDCondMgr` and `PDConduitInfo` objects in the COM Sync Suite enable you to read and write these values after registration. These APIs also enable you to create and access your own entries, so that you can store information that your installer, desktop application, or conduit might need to associate with your conduit. For more about conduit registration, see Chapter 6, "Registering Conduits and Notifiers with HotSync Manager," on page 73.

Table A.1 describes each of the PalmSource-defined conduit configuration entries. Some of these entries are required for conduit registration and some are not. Setting the entries required by HotSync Manager registers a conduit so that HotSync Manager can call it. For each registered conduit, there is a separate set of these entries. Additionally, Conduit Manager versions 3 and later store a different set of these entries for each Windows user, enabling different conduits to be registered for each Windows user.

The "Req." column in Table A.1 shows whether the entry is required—that is, whether HotSync Manager must read the value and act upon it, and if so, what versions of HotSync Manager require it. Many entries are entirely optional. If an entry is not marked as required for a version of HotSync Manager that you want to support, then you do not need to set a value for it.

The "Data Type" column shows how each is stored in the conduit configuration entries (the type passed to the Conduit Manager API may differ).

Some of these entries are also accessible with the Conduit Configuration (CondCfg) utility. For more information about CondCfg, see Chapter 3, "Conduit Configuration Utility," on page 11 in the *Conduit Development Utilities Guide*.

Table A.1 Conduit configuration entries

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`Arguments`	Opt.	string	*Deprecated as of CDK 6.0.* No version of HotSync Manager uses this value. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorArgument()`, `CmSetCreatorArgument()`
`ClassName`	Java	string	Full name of the Java-based conduit class (including package). This entry is used only for Java-based conduits. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorValueString()`, `CmSetCreatorValueString()` COM Sync Suite (`PDConduitInfo` property): `JavaClassName`
`ClassPath13`	Java	string	Directory that contains all the classes used by the selected Java-based conduit. This is the value of `CLASSPATH` required to find all the classes invoked by the Java-based conduit. The `CLASSPATH` setting in the Windows environment variable (NT) or `autoexec.bat` files is *ignored* by the Java-based conduit at runtime. The `ClassPath13` entry is used only for Java-based conduits. (Conduits written for an older version of the JSync module based on JRE 1.1.3 use the `ClassPath` entry instead. All conduits written for JSync based on JRE 1.3 must use the `ClassPath13`). Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorValueString()`, `CmSetCreatorValueString()` COM Sync Suite (`PDConduitInfo` property): `JavaClassPath`

Table A.1 Conduit configuration entries *(continued)*

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`COMClient`	COM	string	If your conduit is a standard EXE, then this value is the full path and filename of your client conduit. If you are debugging, then this is the path of your IDE executable—for example, `C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\IDE\devenv.exe`. If your conduit is an ActiveX server, this value is the notification object's ProgID (also called the Programmatic ID)—for example, `SimpleDb.CNotify`. This entry is used only for COM conduits. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorValueString()`, `CmSetCreatorValueString()` COM Sync Suite (`PDConduitInfo` property): `COMClassID`
`Conduit`	All	string	Name of the conduit DLL file. If this entry is only a filename, the DLL must be in the HotSync Manager executable's directory or in the current Windows user's `PATH`. If it is a path and filename, you can put the DLL in any directory. If this is a Java-based conduit, this entry must be set to `jsync13.dll` for conduits developed with CDK 4.02 or later; it must be `jsync.dll` if your conduit must work with a version of the JSync module prior to the one that ships with CDK 4.02. If this is a COM-based conduit, this entry must be set to `COMConduit.dll`. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorName()`, `CmSetCreatorName()`, `CmConduitType2`

Table A.1 Conduit configuration entries *(continued)*

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`Creator`	All	`DWORD`	The **creator ID** of the application on the handheld your conduit is responsible for synchronizing. This value is the unique key by which HotSync Manager identifies your conduit, so only one conduit can be registered with a particular creator ID at a time. Though it is stored as a `DWORD`, the creator ID is a special four-character string. Some APIs represent it as a `DWORD` and others as a string. Use the provided utility functions as necessary to convert a creator ID from one representation to the other. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorIDList()`, `CmInstallCreator()`, `CmConduitType2`, `CmConvertCreatorIDToString()`, `CmConvertStringToCreatorID()` COM Sync Suite (`PDConduitInfo` property): `CreatorID`

Table A.1 Conduit configuration entries *(continued)*

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`Directory`	Opt.	string	Conduit's directory name. This is the name of a subdirectory in the user's directory on the desktop computer (not a fully qualified path). Within each user's directory, each conduit can have a directory for file storage. For example, if the `Directory` value is "DateBook", then its path is typically `C:\Documents and Settings\<WinUsername>\My Documents\Palm OS Desktop\<HotSyncUsername>\DateBook`. It could hold support files, such as record ID mapping files, needed to accurately perform a record-level synchronization with a third-party database. HotSync Manager passes this value to a conduit via its `OpenConduit()` or equivalent entry point Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorDirectory()`, `CmSetCreatorDirectory()`, `CmConduitType2`, `CSyncProperties` COM Sync Suite (`PDConduitInfo` property): `DeskTopDataDirectory`
`File`	Opt.	string	Name of the desktop file that your conduit synchronizes with the handheld database. Your conduit can synchronize with more than one file, however. Note that this configuration entry can be either a full path and filename, or only a filename. If the value is only a filename, the file can be found in the directory specified by the `Directory` entry. HotSync Manager passes this value to a conduit via its `OpenConduit()` or equivalent entry point Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorFile()`, `CmSetCreatorFile()`, `CmConduitType2`, `CSyncProperties` COM Sync Suite (`PDConduitInfo` property): `DeskTopDataFile`

Table A.1 Conduit configuration entries *(continued)*

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`Information`	Opt.	string	*Deprecated as of CDK 6.0.* No version of HotSync Manager uses this value. Instead of using this entry, create your own. This entry stores any string you want to associate with your conduit. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorInfo()`, `CmSetCreatorInfo()`
`Integrate`	Opt.	`DWORD`	*Deprecated as of CDK 6.0.* No version of HotSync Manager uses this value. Installers for older versions of the Palm OS® Desktop software use this value to specify whether the application specified in `Module` is integrated into the Palm OS Desktop software application. Most conduits that are installed by developers are not integrated conduits, and newer versions of Palm OS Desktop do not use this value at all. "0" – not integrated. "1" – integrated. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorIntegrate()`, `CmSetCreatorIntegrate()` `CmInstallCreator()` `CmInstallConduit()`
`Module`	Opt.	string	*Deprecated as of CDK 6.0.* No version of HotSync Manager uses this value. Name of a plug-in executable associated with this conduit that is to be integrated into the Palm OS® Desktop software. This entry is used only by older versions of Palm OS Desktop software; newer versions do not use it. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorModule()`, `CmSetCreatorModule()`

Table A.1 Conduit configuration entries *(continued)*

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`Name`	Opt.	string	Display name of the conduit. HotSync Manager displays this string as the name of the conduit in its user interface—for example, in the **Custom** dialog box. If you do not set this entry, HotSync Manager shows the name that your conduit provides when called by `GetConduitInfo()` or `GetConduitName()` in C/C++ Sync, `Conduit::name()` in JSync, or `IPDClientNotify->GetConduitInfo()` in COM Sync). Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorTitle()`, `CmSetCreatorTitle()`, `CmConduitType2` COM Sync Suite (`PDConduitInfo` property): `DisplayName`
`Priority`	Opt.	`DWORD`	Execution priority for this conduit. This value is in the range 0 to 4. If no value is specified, then HotSync Manager uses a default value of 2. HotSync Manager runs conduits with a value of 0 first and those with 4 last. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorPriority()`, `CmSetCreatorPriority()`, `CmConduitType2` COM Sync Suite (`PDConduitInfo` property): `Priority`

Table A.1 Conduit configuration entries *(continued)*

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`Remote`	Opt.	string	Name of a handheld database to be accessed by this conduit. This optional entry can be used by conduits that are not hard-coded with specific database names. This value is passed to the conduit to enable it to open the database on the handheld. A conduit can also use this name to create the database on the handheld if the database did not exist before synchronization. Database names are case-sensitive. HotSync Manager passes this value to a conduit via its `OpenConduit()` or equivalent entry point Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorRemote()`, `CmSetCreatorRemote()`, `CmConduitType2`, `CSyncProperties` COM Sync Suite (`PDConduitInfo` property): `HandHeldDB`
`Username`	Opt.	string	*Deprecated as of CDK 6.0.* No version of HotSync Manager uses this value. Conduit Manager API (C/C++ Sync Suite): `CmGetCreatorUser()`, `CmSetCreatorUser()`

1. "All" = required for all conduits.
"Opt." = optional for all conduits.
"C" = required for C API-based conduits only.
"Java" = required for Java-based conduits only.
"COM" = required for COM-based conduits only.

Install Conduit Configuration Entries

The **install conduit configuration entries** are name/value pairs that the Install Conduit Manager for Windows stores for each registered install conduit. These are written at the time you register your install conduit. The Install Conduit Manager API in the C/C++ Sync Suite and the `PDInstallConduit` and `PDInstallConduitInfo` objects in the COM Sync Suite enable you to read and write these values after registration. These APIs also enable you to create and access your own entries, so that you can store information that your installer, desktop application, or install conduit might need to associate with your install conduit. For more about conduit registration, see Chapter 6, "Registering Conduits and Notifiers with HotSync Manager," on page 73.

Table A.2 describes each of the PalmSource-defined install conduit configuration entries. As this table shows, most of these entries are required for install conduit registration. Setting the entries required by HotSync Manager registers an install conduit so that HotSync Manager can call it. For each registered install conduit, there is a separate set of these entries. Additionally, Install Conduit Manager versions 3 and later store a different set of these entries for each Windows user, enabling different install conduits to be registered for each Windows user.

The "Req." column in Table A.2 shows whether the entry is required—that is, whether HotSync Manager must read the value and act upon it, and if so, what versions of HotSync Manager require it. Many entries are entirely optional. If an entry is not marked as required, then you do not need to set a value for it.

The "Data Type" column shows how each is stored in the conduit configuration entries (the type passed to the Install Conduit Manager API may differ).

Table A.2 Install conduit configuration entries

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`CreatorID`	All	`DWORD`	A unique ID associated with your install conduit. This value is required to register your install conduit with HotSync Manager; however, HotSync Manager uses the `Mask` value to uniquely identify your install conduit instead. It is the responsibility of the installer to ensure that the `CreatorID` value is unique. This value is not necessarily related to the creator ID you register with PalmSource, Inc. However, one way to ensure your value is unique is to use the creator ID you registered with PalmSource, Inc. Install Conduit Manager API (C/C++ Sync Suite): `ImRegister()`, `ImRegisterID()` `ImRegisterSystem()`, `ImRegisterSystemID()` COM Sync Suite (PDInstallConduitInfo property): `UniqueId`
`Directory`	All	string	Name of the install directory associated with your install conduit. This is a subdirectory in the user's directory on the desktop computer. The Install Aide copies files here to be installed during the next HotSync operation. For more information, see "Install Directory Terminology" on page 131 in the *C/C++ Sync Suite Companion*. Install Conduit Manager API (C/C++ Sync Suite): `ImGetDirectory()`, `ImSetDirectory()` `ImGetSystemDirectory()`, `ImSetSystemDirectory()` COM Sync Suite (PDInstallConduitInfo property): `Directory`

Table A.2 Install conduit configuration entries *(continued)*

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`Extensions`	All	string	The file type extensions of the files that this install conduit can install. This string is in the standard Windows `CFileDialog` format—for example, `Palm Applications(*.prc)|*.prc|Palm Databases (*.pdb)|*.pdb|Palm Query Application (*.pqa)|*.pqa` Install Conduit Manager API (C/C++ Sync Suite): `ImGetExtension()`, `ImSetExtension()` `ImGetSystemExtension()`, `ImSetSystemExtension()` COM Sync Suite (PDInstallConduitInfo property): `Extension`
`Mask`	All	`DWORD`	A unique bit mask value associated with your install conduit. HotSync Manager uses this mask value to identify your install conduit. Your installer is responsible for ensuring that this value is unique. Install Conduit Manager API (C/C++ Sync Suite): `ImGetMask()`, `ImSetMask()` `ImGetSystemMask()`, `ImSetSystemMask()` COM Sync Suite (PDInstallConduitInfo property): `Mask`

Table A.2 Install conduit configuration entries *(continued)*

Config. Entry	Req.[1]	Data Type	Description, Functions to Get/Set
`Module`	All	string	The filename of the install conduit—for example, `inscn20.dll`. HotSync Manager looks for this file first in the HotSync executable path, then in the paths specified by the Windows `PATH` environment variable. Install Conduit Manager API (C/C++ Sync Suite): `ImGetModule()`, `ImSetModule()` `ImGetSystemModule()`, `ImSetSystemModule()` COM Sync Suite (PDInstallConduitInfo property): `Module`
`Name`	Opt.	`DWORD`	User-visible name of the install conduit. In the **Custom** menu, HotSync Manager displays this string as the name of the install conduit. Install Conduit Manager API (C/C++ Sync Suite): `ImGetName()`, `ImSetName()` `ImGetSystemName()`, `ImSetSystemName()` If this entry is not present, the value returned by a C API-based conduit's `GetConduitInfo()` (or `GetConduitName()`) entry point is displayed instead. COM Sync Suite (PDInstallConduitInfo property): `Name` If this entry is not present, the value returned by a COM-based conduit's `GetConduitInfo()` entry point is displayed instead.

1. "All" = required for all install conduits.
"Opt." = optional for all install conduits.

HotSync Manager Configuration Entries

The **HotSync Manager configuration entries** are name/value pairs that the Conduit Manager or HotSync Manager for Windows stores for HotSync Manager itself. These entries are not conduit-specific, but affect how HotSync Manager operates for all Windows users. The Conduit Manager, HotSync Manager, Install Aide, and User Manager APIs in the C/C++ Sync Suite and the `PDCondMgr` and `PDHotSyncUtility` objects in the COM Sync Suite enable you to read (and write many of) these values.

Table A.3 describes each of the HotSync Manager configuration entries. The “Data Type” column shows how each is stored in the configuration entries (the type passed to an API function may differ).

Table A.3 HotSync Manager configuration entries

Config. Entry	Data Type	Description, API Functions to Get/Set

`Core\Path`	string	Path of the HotSync users directory for the current Windows user—for example, `C:\Documents and Settings\<WinUsername>\My Documents\Palm OS Desktop`. It does not include a user directory's name, though.
		C/C++ Sync Suite:
		`CmGetCorePath()`, `UmGetRootDirectory()`, `PltGetPath()`
		COM Sync Suite:
		`PDUserData.GetRootDirectory()`, `PDInstall.GetPath()`

`Core\HotSyncPath`	string	Path and filename of the HotSync Manager executable on the desktop computer—for example, `C:\Program Files\PalmSource\Desktop\HotSync.exe`. To get the path of support DLLs like `CondMgr.dll` and `Instaide.dll`, strip off the `HotSync.exe` filename.
		C/C++ Sync Suite:
		`CmGetSystemHotSyncExecPath()`, `PltGetPath()`
		COM Sync Suite:
		`PDInstall.GetPath()`

`HotSync Manager\DirectComPort`	string	COM port that HotSync Manager monitors for direct serial (cable, cradle) synchronization—for example, "COM1". C/C++ Sync Suite: `CmGetComPort()`, `CmSetComPort()` COM Sync Suite: N/A

Table A.3 HotSync Manager configuration entries *(continued)*

Config. Entry	Data Type	Description, API Functions to Get/Set
`HotSync Manager\ModemComPort`	string	COM port that HotSync Manager monitors for modem synchronization—for example, "COM1". C/C++ Sync Suite: `CmGetComPort()`, `CmSetComPort()` COM Sync Suite: N/A

`HotSync Manager\BackupConduit`	string	Filename of the conduit that HotSync Manager calls as the backup conduit.
		C/C++ Sync Suite:
		`CmGetBackupConduit()`,
		`CmSetBackupConduit()`
		COM Sync Suite:
		`PDCondMgr.GetBackupConduit`

Table A.3 HotSync Manager configuration entries *(continued)*

Config. Entry	Data Type	Description, API Functions to Get/Set
`HotSync Manager\PCIdentifier`	`DWORD`	The PC ID, a unique identifier for each Windows user on the desktop computer. Different HotSync users using the same Windows login have the same PC ID. C/C++ Sync Suite: `CmGetPCIdentifier()` COM Sync Suite: N/A

`HotSync Manager\NotifierN`	string	Filename of a notifier that HotSync Manager calls at the start and end of the HotSync process. The Notifier Install Manager assigns the value N when its API is called to register a notifier.
		C/C++ Sync Suite:
		`NmRegister()`,
		`NmRegisterSystem()`,
		`NmGetByIndex()`,
		`NmGetSystemByIndex()`
		COM Sync Suite:
		`PDCondMgr.RegisterNotifier`,
		`PDCondMgr.GetNotifierList`

Glossary

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

68K application Palm OS applications have traditionally been developed using the 68K APIs. Applications developed with the 68K APIs can run on all versions of Palm OS. The early versions of Palm OS, Palm OS 1 through Palm OS 4, run on handhelds with 68K-family processors. 68K applications run natively on these devices.

Palm OS Garnet (the Palm OS 5 releases) and Palm OS Cobalt (the Palm OS 6 releases) are designed for handhelds with ARM-based processors. Both include PACE—the Palm OS Application Compatibility Environment—which enables 68K applications to run on these Palm OS versions as well. Note that on ARM-based Palm-Powered devices, you can also employ PNOs (PACE Native Objects) to speed up time-critical portions of your application. See also **Palm OS Protein application**.

The 68K APIs are documented in the *Palm OS Programmer's Companion* and the *Palm OS Programmer's API Reference*.

ActiveX client An ActiveX client allows you to access a COM-based conduit from anywhere on the network through DCOM.

application conduit A conduit associated with a third-party application, not an application integrated within the Palm OS Desktop software. Contrast with component conduit.

application info	Optional data in a database that an application can use for its own purposes. For example, it might be used to store user display preferences for a particular database. If your application uses the handheld's category API to manage category information for a classic database, then the start of the application info block must be an `AppInfoType` structure. Otherwise, it can hold whatever you wish. Schema databases define the application info block as a single record that uses the application info schema. Category data is not stored in a schema database's application info block. This schema is application-defined, so this record can hold whatever you wish.
application preferences	Handheld applications running on version 2.0 or later of Palm OS can store their nonvolatile preferences in one of two preference databases: Preferences stored in the "saved" preference database are backed up during synchronization operations and are automatically restored when required. Preferences stored in the "unsaved" preference database are never backed up or restored by HotSync Manager. The structure of the data in the preferences block is application dependent. The Sync Manager does not modify this data in any way when retrieving it. This means that multi-byte integer data is stored using big-endian byte ordering, with the most significant byte stored at the lower address in memory. Your conduit is responsible for performing any necessary byte swapping. You must check the version and size of the preference block to ensure compatibility with the application. For more information, see *Exploring Palm OS: System Management*.
archiving	Saving deleted records on the desktop computer during synchronization operations. Archived records are not synchronized even if the user modifies them.

backup conduit	A conduit that copies to the desktop any handheld databases and applications that have their backup bit set. The HotSync® Manager includes a backup conduit (called Backup) by default. Backup conduits are intended for handheld databases and applications that do not have synchronization conduits associated with them.
built-in applications	ROM-based Palm OS applications that are built into the handheld. For example, the Memo Pad, Date Book, Mail, To do List, and Address Book applications.
C API-based conduit	A conduit that is written in the C/C++ language with the C/C++ Sync Suite. Its entry points present a C interface, which HotSync Manager calls directly. See also **COM-based conduit**.
category name	Name of a category in a **schema database**. Category names should be 7-bit ASCII characters from 0x20 through 0x7E. The maximum size of a category name is 32 characters including the null terminator value.
C/C++ Sync Suite	The components of the CDK for Windows used to create a C API-based conduit. It includes APIs, the C++ Generic Conduit Framework, samples, documents, and utilities. It also includes APIs to access user information, to install databases and files on a handheld, and to perform other actions that do not necessarily happen during a HotSync operation.
CDK	Palm OS® Conduit Development Kit from PalmSource, Inc.
change context	An opaque array of bytes of variable length that the Sync Manager retrieves from the handheld and stores per user, conduit, and schema database after a conduit has synchronized a schema database. During a subsequent HotSync operation, the Sync Manager uses the stored change context to determine changes in the schema database since the last HotSync operation The Sync Manager then specifies to the conduit the type of synchronization required: slow, fast, or fast with deleted **sync atoms**s purged. The Sync Manager API allows a conduit to store the change context itself, perhaps in a central location, so that it can be used to enable the user to synchronize faster from another location.

change counter	When any **sync atoms** is modified, added, or deleted, the Data Manager updates this counter with the current database **sync clock** value.
classic database	The record database format supported in Palm OS® versions 1.0 and later. This is the format used by applications running on Palm OS Cobalt through PACE. Classic databases contain records that are limited to 64 KB in length, are uniquely identified by name alone, and must store record data using big-endianness, for compatibility with the 68K-based Dragonball CPU used in the early Palm OS devices. See **extended database** and **schema database**.
column name	Name of a table column in a **schema database**. Column names should be valid SQL identifiers.
COM Sync module	The software layer between a COM-based conduit and HotSync Manager. See "COM-based Conduit Fundamentals" on page 2 in *COM Sync Suite Companion*.
COM Sync Suite	The components of the CDK for Windows used to create a COM-based conduit. Includes COM objects/methods/properties, the IPDClientNotify interface, COM Sync software layer (`COMClient.dll`, `COMConduit.dll`, `COMServer.dll`), documents, utilities, and the COM Sync Installer.
COM-based conduit	A conduit that is written in any COM-enabled language with the COM Sync Suite. It implements the IPDClientNotify interface, which HotSync Manager calls via the COM Sync module. See also **C API-based conduit**.
component conduit	A conduit associated with an application that is a component of the Palm OS® Desktop software (for example, Date Book). Contrast with application conduit.
conduit	A plug-in module for the HotSync Manager application. Typically, a conduit synchronizes data for an application on a handheld with data for a desktop application or in a networked database, converting data formats between the two as necessary. In most instances, "conduit" refers to a synchronization conduit.

Conduit Configuration utility	A developer utility for registering and unregistering conduits and notifiers with HotSync Manager.
conduit entry points	Functions implemented in a conduit that HotSync Manager calls to start, to get information from, and otherwise to communicate with a conduit.
configuration entries	The configuration information stored on a Windows desktop computer about HotSync Manager and each conduit it runs during a HotSync operation. When an installer conventionally registers its conduit with HotSync Manager, it must use the APIs documented in the CDK to add a configuration entry. Alternatively in HotSync Manager versions 6.0 and later, you can register a conduit simply by copying it to the `Conduits` folder. See **folder-registered conduit**.
Conduit Manager	For Windows, a library that provides functions to register conduits with HotSync Manager and to access the **configuration entries** on the desktop.
Conduit Wizard	A wizard for Microsoft Visual C++ that automatically generates a functional shell for a C API-based conduit.
conventionally registered conduit	Writing configuration entries is the conventional method supported for C API-, COM-, and Java-based conduits by all versions of HotSync Manager. To register a conduit in the conventional manner, use the Conduit Configuration utility or call the Conduit Manager API to write new information into the conduit configuration entries. Contrast this with a **folder-registered conduit**.
cradle	The special holder that interfaces a Palm OS handheld with a desktop computer, usually via a serial connection. When you press the HotSync button on the cradle, HotSync Manager on the desktop begins synchronizing.
creator ID	The unique, four-character ID associated with each application and its databases on the handheld. Each conduit is associated with a specific creator ID, which is almost always the same creator ID as the handheld application for which it synchronizes data. HotSync

	Manager for Windows uniquely identifies a conduit by its creator ID on a given desktop per Windows user. Therefore no two conduits for a given Windows user can have the same creator ID, unless one is registered for the system. PalmSource, Inc. maintains a database with which you must register your application's creator ID: www.palmos.com/dev/support/creatorid/. Note that PalmSource, Inc. reserves all creator IDs consisting of all lowercase characters.
database type	A four-byte, application-specific value assigned to handheld databases to assist Palm OS or an application in identifying the database. Palm OS associates special behavior with several identifier values, including `'DATA'`, `'data'`, `'appl'`, `'panl'`, and `'libr'`.
database name	Name of a Palm OS database. Schema database names should be valid SQL identifiers. Non-schema database names consist of only 7-bit ASCII characters from 0x20 through 0x7E. The maximum size of a database name is 32 characters including the null terminator value. Classic, extended, and schema databases exist in disjoint namespaces. Because the namespaces are disjoint, it is possible for up to three databases, one per namespace, to have the same name and creator ID.
database reset identifier	(DRID) The schema Sync Manager generates a new DRID for each schema database when it is created or restored to a handheld after a hard reset. The DRID enables the schema Sync Manger to identify situations in which the **sync clock** value, **change counter**, and **purge counter** are no longer dependable—in which case, HotSync Manager tells a conduit that it cannot perform a fast sync.
Data Manager	The Palm OS service that provides handheld applications access to databases. Conduits access databases on the handheld via the desktop Sync Manager, which in turn calls the Data Manager on the handheld.

default conduits	Conduits that PalmSource supplies with the HotSync Manager application. These include synchronization conduits for the Palm OS Desktop applications (Address Book, Date Book, To Do List, and Memo Pad) as well as install conduits and a backup conduit. Palm OS licensees or third-party products may replace some or all of these conduits, so the presence of these conduits is not guaranteed.
desktop application	An application program that runs on a Windows-based desktop computer.
desktop computer	A Windows-based computer. In conduit documentation, this term is often shortened to "desktop" and is used to distinguish it from the handheld.
direct cable connection	A connection between a desktop computer and a cradle that is achieved by connecting the cradle's cable to the desktop computer.
extended database	An "extended" version of a **classic database**. There are three primary differences between classic and extended databases: extended database records can exceed 64 KB in length (classic records cannot); extended databases are uniquely identified by a combination of name and creator ID (classic databases are uniquely identified by name alone); and extended databases can store data using the processor's native endianness (classic databases must store record data using big-endianness, for compatibility with the 68K-based Dragonball CPU used in the early Palm OS devices). Also see **schema database**
fast sync	A form of synchronization that is used when the most recent synchronization for the handheld was performed with the current desktop computer. In fast sync operations, only modified records should be copied between the desktop computer and handheld. A slow sync is performed when a single handheld is synchronized with multiple desktop computers. See **slow sync**.
file link	A feature of HotSync® Manager that allows users to automatically update and merge data from an external data source (a file) to the application data on their handhelds. For example, an address database conduit could link to a company-wide address database

	that is centrally maintained and accessed over a network. File links are possible only with HotSync Manager versions earlier than 6.0.1 for Windows. This feature has been discontinued in version 6.0.1 and later.
folder-registered conduit	A C API-based conduit that is in the current Windows user's or the system's `Conduits` folder. This type of conduit must implement in its `GetConduitInfo()` entry point the logic to respond to HotSync Manager's request for registration information. See **register**.
Generic Conduit Framework	C++ classes provided in the CDK that implement synchronization logic without requiring you to use MFC at all. PalmSource, Inc. highly recommends that you use the Generic Conduit for any mirror-image synchronization conduit.
handheld	A handheld computer (often used synonymously with "Palm Powered™ handheld").
HotSync button	The button on the cradle that the user presses to initiate synchronization operations.
HotSync Manager	The desktop application that oversees the HotSync process. Conduits are plug-in modules for this application.
HotSync process	PalmSource's patented, one-button method of automatically synchronizing all data between a handheld and a desktop computer. The HotSync process invokes a conduit on the desktop to synchronize the database s for each handheld application that has a conduit and automatically backs up the databases for those that do not have a conduit. For details, see Chapter 7, "Understanding the HotSync Process," on page 85 in *Introduction to Conduit Development*.
install	When referring to a conduit, includes at least registering the conduit with HotSync Manager and copying the conduit file(s) to the desktop computer. The same installer could also install a desktop application and queue a handheld application (and databases) to be installed on the handheld during the next HotSync operation.

Install Aide	A library of functions that your desktop software can use to install applications and databases from a desktop computer to a handheld. It can also transfer any type of file between the desktop and a card in a handheld expansion slot.
install conduit	A C API-based conduit that installs Palm OS applications and databases in a handheld's main memory or copies files from the desktop to handheld expansion cards during a HotSync operation. Unlike synchronization conduits, HotSync Manager for Windows runs install conduits twice: before and after it runs all synchronization conduits. PalmSource ships HotSync Manager with one or more default install conduits.
Install Conduit Manager	Part of the Conduit Manager library that provides functions to register install conduits with HotSync Manager for Windows.
installed user	A user who has completed a HotSync operation so that the user's user ID is on both the desktop computer and a handheld.
Java-based conduit	A conduit that is written in the Java language with the JSync Suite.
JSync	The software layer between a Java-based conduit and HotSync Manager.
JSync Suite	The components of CDK 4.03 for Windows used to create a Java-based conduit. Includes packages, classes, samples, documents, utilities, JSync software, and the JSync Installer. The JSync Suite enables you to synchronize only classic databases.
memory card	Palm OS versions earlier than Palm OS Cobalt allow for more than one primary storage area on the handheld, called memory cards. These cards are specified by number: the first memory card in the system is card number 0, and subsequent card numbers are incremented by 1. No handheld actually used more than one memory card, except for certain HandSpring handhelds. In Palm OS Cobalt, the concept of a memory card no longer exists. In any calls that take a card number, always specify 0.

mirror-image conduit	A conduit that performs mirror-image synchronization, which results in identical data for an application on the desktop computer and the handheld.
modem connection	One of the communications connections through which a user can perform a HotSync operation. In this connection, the handheld connects to the desktop computer using a modem.
multithreading	Multithreading makes it possible to run multiple threads, or client processes, that all connect with the same COM-based conduit.
network connection	One of the communications connections through which a user can perform a HotSync operation. In this connection, the handheld connects to the desktop computer using a local area network.
non-schema database	An **extended database** or a **classic database**.
notifier	An optional dynamic-link library that developers can provide to notify their desktop applications about synchronization activities. HotSync Manager calls all registered notifiers before synchronization begins, passing them a message that they act upon—for example, telling the desktop application to save and stop modifying data. Notifiers are called again after synchronization so they can in turn tell their desktop applications whether the synchronization completed successfully. The API you must implement for a notifier is specified in the C/C++ Sync Suite.
Notifier Install Manager	Part of the **Conduit Manager** library that provides functions to register notifiers with HotSync Manager for Windows.
one-directional conduit	A conduit that copies data in one direction: either from the desktop computer to the handheld, or from the handheld to the desktop computer.
Palm OS Desktop software	The desktop computer application software supplied with Palm OS handhelds.

Palm OS	The operating system and services that run on a Palm Powered handheld.
Palm OS application	An application program that runs on a Palm OS handheld.
Palm OS Cobalt	Family of operating system products from PalmSource, Inc. starting with version 6.0.
Palm OS Garnet	Family of operating system products from PalmSource, Inc. starting with version 5.0.
Palm OS platform	Includes the Palm OS, built-in applications, hardware architecture, conduits, and development tools.
Palm Powered handheld	Any handheld based on the Palm OS platform. Often shortened to “handheld” in Palm OS platform documentation.
PC ID	A pseudo-random number that HotSync Manager generates and uses to uniquely identify a desktop OS user. Note that each desktop OS user’s login on a given computer has a different PC ID.
permanent synchronization preference	The user’s **synchronization preference** that HotSync Manager passes to a conduit at the start of *all* HotSync operations, unless the user later changes the permanent preference or sets a **temporary synchronization preference**.
profile	See user profile.
Palm OS Protein application	An ARM-native application written using the Palm OS Protein APIs, which allow it to take advantage of the new features introduced in **Palm OS Cobalt**. Palm OS Protein applications can be multithreaded, can access schema and extended databases, and can employ Palm OS Cobalt’s multimedia framework. Note that Palm OS Protein applications won’t run on devices running **Palm OS Garnet** or on earlier Palm OS releases. The Palm OS Protein APIs are documented in the *Exploring Palm OS* series. See also **68K application**.

purge counter	When all the deleted instances of a **sync atoms** are purged from a schema database, the Data Manager updates this counter with the current database **sync clock** value. This counter tracks the relative time of the last purge and applies to the entire database.
record status flags	Flags in databases on the handheld that indicate whether an individual record has been modified since the most recent synchronization. Used for fast sync operations.
register	When referring to conduits for Windows, "to register" a conduit means to perform a one-time operation outside of the HotSync process that provides HotSync Manager with basic information about a conduit. This information enables HotSync Manager to uniquely identify and to find a conduit when it needs to run it or to call any of a its entry points. An installer can register a conduit in one of two ways: writing configuration entries or copying a conduit to the `Conduits` folder. **Conventional registration:** Writing configuration entries is the conventional method supported for C API-, COM-, and Java-based conduits by all versions of HotSync Manager. To register a conduit in the conventional manner, call the Conduit Manager API to write new information into the conduit configuration entries. **Folder registration:** Copying a conduit to the `Conduits` folder is a method supported only for C API-based conduits and only by HotSync Manager versions 6.0 and later. To register a conduit using this method, call the Conduit Manager API to locate the `Conduits` folder and then copy your conduit DLL to that folder. See **folder-registered conduit**. Developers can also use the Conduit Configuration utility to write conduit configuration entries. Be careful not to confuse registering with HotSync Manger and registering with the Sync Manager. C API-based conduits must register with the Sync Manager at run time.
remote transactions	Data transactions performed by a user on a handheld.

schema database	A type of Palm OS database that bears a strong resemblance to a relational database. Introduced in Palm OS Cobalt, schema databases organize data into tables, which consist of rows and columns. Schema databases use the concept of a schema to define the structure of a table row. Unlike relational databases, however, schema databases don't allow you to perform joins and other complex operations. There are three primary differences between schema and classic databases: schema database rows are structured and can exceed 64 KB in length (classic records are blobs that must be less than 64 KB); schema databases are uniquely identified by a combination of name and creator ID (classic databases are uniquely identified by name alone); and schema databases store data using the processor's native endianness (classic databases must store record data using big-endianness, for compatibility with the 68K-based Dragonball CPU used in the early Palm OS devices). See the two "non-schema" database types: **extended database** and **classic database**.
slow sync	A form of synchronization that is used when the most recent synchronizations for the handheld and desktop computer were not performed with each other. In slow sync operations, a comprehensive comparison of handheld and desktop computer records is performed. See **fast sync**.
sync atoms	A unit of information in a schema database for which the handheld Data Manager tracks changes between HotSync operations by incrementing a change counter. Palm OS Cobalt defines the following sync atoms: table definitions, category definitions, all row data, and each row's category membership. When a sync atom is modified, added, or deleted in a given schema database, the handheld updates the sync atom's **change counter** with the existing **sync clock** value. Then at the end of each successful HotSync operation, the handheld updates the database's sync clock.
sync clock	For each schema database, a sync clock indicates when it was last synchronized. At the time of its creation, a database has a sync clock value that starts at 1. At the end of each successful HotSync

	operation, the Data Manager on the handheld increments the database's sync clock by 1 and updates the change counter and purge counter with the sync clock value.
Sync Manager API	The collection of functions and data structures through which the desktop computer interacts with the handheld during synchronization operations. This lower-level API is called by the higher-level methods provided in the Generic Conduit Framework. The COM Sync software provides a bridge between the Sync Manager and a COM-based conduit.
synchronization conduit	Performs whatever tasks a conduit developer designs it to do, typically synchronizing databases on the handheld with their desktop counterparts. Contrast with "install conduit" and "backup conduit."
synchronization logic	Code that implements the synchronization actions you need your conduit to perform. The CDK provides the Generic Conduit Framework, which performs full, mirror-image synchronization. PalmSource, Inc. highly recommends that you use the Generic Conduit Framework, especially for any mirror-image synchronization conduit.
synchronization preference	The manner in which a conduit synchronizes during subsequent HotSync operations as set by the user. Conduits can implement a **Change HotSync Action** dialog box, as the default conduits do, which prompts the user to select **Synchronize the files**, **Desktop overwrites handheld**, **Handheld overwrites Desktop**, or **Do nothing**. At the start of every HotSync operation, HotSync Manager passes to each conduit the synchronization preference that the user set for that conduit. A conduit can support the preferences that the default conduits support, its own custom preferences, or none at all. Also see permanent synchronization preference and temporary synchronization preference.
table name	Name of a table in a schema database. Table names should be valid SQL identifiers.

temporary synchronization preference	The user's **synchronization preference** that HotSync Manager passes to a conduit at the start of *only the next* HotSync operations. During all HotSync operations thereafter, a HotSync Manager reverts to the user's **permanent synchronization preference**, unless the user makes a further change to make a conduit's temporary preference permanent.
transaction-based conduit	A conduit that performs further logic-based operations beyond copying data between the handheld and the desktop computer. For example, a conduit that sends different data to the handheld based on data, or requests, that are found on the handheld. Another example is an email conduit that performs more complex operations than simply creating a mirror image.
user data store	Information stored on the desktop about HotSync users created by the current Windows user. HotSync Manager stores user information here—for example, name, synchronization preferences, directory, and password information—but you can create and access data that you want to track per HotSync user. The user data store is divided into named sections containing named keys whose values you set. Use the User Data API to access this information programmatically. Developers can use the User Info utility to view or modify basic user data during development.
user ID	An identification number associated with a handheld. This is a pseudo-random number generated and assigned to the handheld by HotSync Manager on the desktop during the handheld's initial synchronization. The user ID value is independent of the user name—that is, the same user name can be associated with multiple user IDs, so the user ID is the only truly unique identifier. HotSync Manager assigns a user ID to a handheld when it synchronizes to either a handheld that has just been hard reset or to a handheld that has never been synchronized. When you synchronize such a handheld to a desktop computer that already has a user name and user ID defined, HotSync Manager displays a list of existing user names. After you select an existing name, HotSync Manager assigns that user name and its already associated user ID to the handheld. Any additional such handhelds synchronized with this desktop will undergo the same assignment process.

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

users Generally, this term refers to end users of Palm Powered handhelds. CDK documentation makes further distinctions between HotSync users and Windows (or desktop) users.

HotSync users refers to all users who have performed a HotSync operation between their device and a single desktop when logged in as the same Windows user.

Windows users refers to all users who have separate Windows logins. There can be multiple HotSync users using a single Windows user login.

vault A special secure database in which the Authorization Manager on the handheld stores all the relevant information to support secure databases—such as HEKs, rules, tokens, and so on. Vaults must be backed up at the end of every HotSync session *after* all other databases. After a hard reset, vaults must be restored to the handheld *before* all other databases.

Index