COM Sync Suite Reference

Palm OS® Conduit Development Kit for Windows, Version 6.0.1

Written by Brent Gossett.
Technical assistance from Ravikumar Duggaraju and Giovanni Marais.

COM Sync Suite Reference
Document Number 3022-006
May 14, 2004
For the latest version of this document, visit
http://www.palmos.com/dev/support/docs/.

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

Table of Contents

About This Document

The COM Sync Suite is a component of the Palm OS® Conduit Development Kit (CDK) for Windows from PalmSource, Inc. This suite provides COM objects/methods/properties, the IPDClientNotify interface, COM Sync software layer, documents, and utilities to help developers create COM-based conduits that run on Windows computers. Key to the success of the Palm OS platform, conduits are software objects that exchange and synchronize data between an application running on a desktop computer and a Palm Powered™ handheld.

The COM Sync Suite Reference describes the COM Sync Suite object hierarchy and provides a prototype of each object, method, and property. This document provides Visual Basic prototypes, parameters, and examples.

The sections in this introduction are:

- Related Documentation
- What this Document Contains
- Changes to This Document
- Conventions Used in this Document
- Additional Resources

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 major parts of this document and the chapters in each.

Chapter 1, "Introduction." Describes the COM Sync Suite's object model and error handling.

Chapter 2, "IPDClientNotify Interface." Specifies the IPDClientNotify interface.

Chapter 3, "Objects." Describes all COM Sync objects in alphabetical order.

Chapter 4, "Methods." Describes all COM Sync methods in alphabetical order.

Chapter 5, "Properties." Describes all COM Sync properties in alphabetical order.

Chapter 6, "Constants." Describes all COM Sync constants in alphabetical order.

Chapter 7, "Errors." Describes all COM Sync errors in alphabetical order.

Appendix A, "Revision History." Lists significant additions and changes in each release of the COM Sync Suite.

Appendix B, "Private Methods and Properties." Lists all of the private methods and properties in the COM Sync module.

Changes to This Document

This section describes significant changes made in each version of this document. For additions and changes to COM Sync Suite APIs in each version of the CDK, see Appendix A, "Revision History," on page 591.

- Document 3022-006 for CDK 6.0.1
- Document 3022-005 for CDK 6.0
- Document 3022-004 for CDK 6.0

Document 3022-006 for CDK 6.0.1

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

- Chapter 4, "Methods," on page 111.
  - Updated example implementation of `GetConduitInfo()` to show use of new `EGetConduitInfo` enum values to opt out of default behaviors.
  - Noted limitations on `CallRemoteModule()` and described how to use it with both Palm OS Cobalt and Palm OS Garnet handhelds.
  - Noted that `ExportDatabaseToFile()` and `ImportDatabaseFromFile()` work only with classic databases.
  - Expanded the description of `GenerateBackupFileName()` and `InstallDatabase()`.
- Chapter 5, "Properties," on page 441.
  - Expanded the description of `SyncType`.
- Appendix A, "Revision History," on page 591.

  Added section that lists all API additions to the COM Sync Suite in CDK 6.0.1. This section lists additions made to this document, which are not listed above.

Document 3022-005 for CDK 6.0

The significant changes are listed by chapter below:

- Chapter 4, "Methods," on page 111.
  - Added that the result of calling `CallRemoteModule()` is indeterminate in a rare case.
  - Noted that `ExportDatabaseToFile()` and `ImportDatabaseFromFile()` do not work with schema databases.
  - Noted that it is not necessary to call `RefreshConduitInfo()` or `RestartHotSyncMgr()` to make HotSync Manager versions 6.0 or later recognize a newly registered conduit.

Document 3022-004 for CDK 6.0

Most of the changes in this version describe the new objects that enable you to access extended and schema databases introduced in Sync Manager API version 2.4 (HotSync Manager 6.0). These and other changes are listed below in chapter order:

- Chapter 3, "Objects," on page 7.
  - Added `PDDateBookDbHHRecord2` and `PDDateBookDbHHRecordAdapter2` objects, which were added to COM Sync in an update released shortly after CDK 4.03. These objects support classic Date Book records that have exceptions to a repeating event.
  - Added objects for accessing **extended database**s. See the list of these objects in "Extended Database Objects" on page 593.
  - Added objects for accessing **schema database**s. See the list of these objects in "Schema Database Objects" on page 594.
- Chapter 4, "Methods," on page 111.
  - Added `GetExceptionDates()` and `SetExceptionDates()` methods to handle exception dates in the `PDDateBookDbHHRecord2` object.
  - Added methods for extended and schema database objects.

- Chapter 5, "Properties," on page 441.
  - Added `DisplayPhone` property, which had been erroneously left out of the previous version of this document.
  - Added properties for extended and schema database objects.

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.
`Sub, If, Case Else, Print, Long`	Words with initial letter capitalized indicate Visual Basic language-specific keywords.
setup	Words you're instructed to type appear in bold.
variable	In prototype and text, placeholders for information you supply.
`variable`	In prototype and text, method or property parameters. .
[expressionlist]	In prototype, items inside square brackets are optional.
`{While | Until}`	In prototype, braces and a vertical bar indicate a choice between two or more items. You must choose one of the items unless all of the items are optional and are enclosed in square brackets.
`Sub HelloButton_Click() Readout.Text = _ "Hello, world!" End Sub`	This font is used for code and prototypes.
`C:\Program Files\COMConduit.dll`	Paths and filenames are given in mixed case.
→	Parameter is passed into a function.
←	Parameter is passed out of a function.
↔	Parameter passed in and out of a function.

Programming Style in This Manual

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

- Keywords appear with initial letters capitalized:

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

- An apostrophe (`'`) introduces comments:

```
' These lines are comments
' Comments are ignored when the program
' is running.
```

- Control-flow blocks and statements in `Sub`, `Function`, and `Property` procedures are indented from the enclosing code:

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

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

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

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

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

1

Introduction

The COM Sync Suite is part of the Palm OS® Conduit Development Kit for Windows from PalmSource, Inc. This suite allows you to directly access information on Palm Powered™ handhelds from a desktop computer using any COM-enabled programming environment and language such as Visual Basic (VB), Visual C/C++, Borland C++ Builder, Borland Delphi, and Java.

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

The following chapters describe the COM Sync interface, objects, methods, properties, constants, and error codes, each in alphabetical order. The sections of this chapter show the following sample descriptions:

- Sample Object Description
- Sample Method Description
- Sample Property Description

For more information about the COM Sync Suite, see the *COM Sync Suite Companion*.

Sample Object Description

Purpose What the object represents in the COM Sync object model. In online versions of this document, use the object description as your primary way of navigating this document for information about this object. All the methods and properties are hyperlinked to their complete descriptions in those chapters.

Methods A list and brief descriptions of all the methods available in this object. If no methods are defined, this section reads "None."

`SampleMethod`
A brief description of each method.

Properties A list and brief descriptions of all the properties defined in this object, including whether the property is read-only (R), or read/write (R/W). If no properties are defined, this section reads "None."

`SampleProperty`
A brief description of each property.

Comments Details on using this object, its capabilities and limitations.

Example An example of how to create and use this object in Visual Basic.

See Also References to related objects, methods, and properties.

Sample Method Description

Purpose	What this method does.
Applies to	Objects that this method is available in.
Prototype	A Visual Basic prototype of this method as seen in the Object Browser.
Parameters	Description of each parameter shown in the prototype. ← `ParamOut` Description of a parameter that is passed back by this method → `ParamIn` Description of a parameter that the caller passes into this method.
Returns	Description of what this method returns. If does not return anything, this section reads “None.”
Errors	Descriptions of error codes that this method can cause. `eErrorCode` What caused this error.
Comments	Details on using this method, its capabilities and limitations.
Example	An example using this method in Visual Basic.
See Also	References to related methods or properties.

Sample Property Description

Purpose	What type of value this property holds.
Applies to	Objects that this property is available in.
Accessibility	Indicates whether this property is read-only or read/write.
Prototype	A Visual Basic prototype of this property as seen in the Object Browser.
Parameters	Description of each parameter shown in the prototype. ← *`ParamOut`* Description of a parameter that is passed back when this property is read. → *`ParamIn`* Description of a parameter that is passed in when this property is written.
Comments	Details on using this property, if any.
Example	An example using this property in Visual Basic.
See Also	References to related methods or properties.

2

IPDClientNotify Interface

This chapter describes the COM Sync Suite's `IPDClientNotify` interface.

IPDClientNotify

Purpose Notification interface implemented by a COM-based conduit client. Clients wishing to be integrated into the HotSync® process must implement this interface. You must expose an object with this interface through the standard COM registration techniques. The interface provides hooks into the HotSync process, configuration, and execution. This interface is to be implemented by only ActiveX clients for the methods below.

Methods

`BeginProcess()`
Sets up connection to begin the synchronization process.

`CfgConduit()`
Informs a conduit when the user selects it from HotSync Manager's **Custom** dialog box. Called only by HotSync Manager versions 3.0 and later (earlier versions call `ConfigureConduit()` instead).

`ConfigureConduit()`
Informs a conduit when the user selects it from HotSync Manager's **Custom** dialog box. HotSync Manager versions earlier than 3.0 call only this method, whereas versions 3.0 and later call the `CfgConduit()` method first and then call `ConfigureConduit` only if the call to `CfgConduit` is not successful.

`GetConduitInfo()`
Returns information about the conduit (including name and version) when requested by HotSync Manager.

Properties None.

Comments `IPDClientNotify` is exported as a part of COM Sync's COMDirect type library and it is mandatory for ActiveX clients to implement this interface. The COM Sync module calls these methods to communicate with the conduit.

3

Objects

This chapter describes the COM Sync objects in alphabetical order. For an overview of the COM Sync object model, see "COM Sync Object Model" on page 14 in the *COM Sync Suite Companion*.

IMPORTANT: Visual Basic .NET shows all interfaces and objects in the PDStandard and PDDirect libraries. *Do not use the interfaces in these libraries; use only the objects.* For example, the Object Browser lists both the `IPDDatabaseQuery` interface and the `PDDatabaseQuery` object, but you must use only the `PDDatabaseQuery` object in your code.

DmCategories

Purpose This utility object supports access to category information in extended databases.

Methods

`Refresh()`
Reinitializes this object from its source, discarding any changes in the cache.

`ResetDirtyFlags()`
Resets all the category `Dirty` flags to `False`.

`Save()`
Writes the category information into the application info block of this database and writes the application info block to the handheld.

Properties

`CategoryId`
(R/W) Category ID specified by category index.

`DbName`
(R) Name of this object's associated database on the handheld.

`Dirty`
(R/W) Category dirty flag specified by category index.

`LastId`
(R/W) Category ID of the last new category.

`Name`
(R/W) Category name specified by category index.

Comments Uses the first 275 bytes of the application info block and converts them to a categories structure. Provides methods and properties to manage the categories structure. When saved, merges them back into the application info block.

Each record in an extended database can have a category index. The definition of all categories in an extended database is stored within the database's application info block.

The `DmCategories` object provides a wrapper for this standard categories structure. Each category entry in the standard category structure contains three elements: `Name`, `CategoryId`, and `Dirty` flag. The `DmCategories` object caches the category information, and updates the extended database only when the `Save()` method is executed. For C++, the following programming structure describes the category structure.

```
typedef struct tagCategories {
   WORD renamedCategories;
   char categoryLabels [16] [16];
   BYTE categoryUniqIDs[16];
   BYTE lastUniqID;
} Categories;
```

See Also `DmRecordAdapter` object.

DmDatabaseInfo

Purpose Represents information about an **extended database** on the handheld.

Method `Refresh()`

Reinitializes this object from its source, discarding any changes in the cache.

Properties `AppInfoSize`

(R) Application info block size of this database.

`BackupDate`

(R) Date that this database was last backed up.

`CardNum`

(R) The number of the **memory card** on which the database is stored.

`CreateDate`

(R) Creation date of this database.

`Creator`

(R) The **creator ID** associated with the current conduit or database.

`DataBytes`

(R) Number of bytes of storage used by this database for data only, excluding overhead.

`DbFlags`

(R) Database flags that are set at creation time.

`DbName`

(R) Name of this object's associated database on the handheld.

`DbType`

(R) The **database type**.

`ExcludeFromSync`

(R) Determines whether this database is excluded from synchronization.

`IsRam`

(R) Determines whether a database is stored in RAM or ROM.

`MaxRecordSize`
(R) Size of the largest record in this database.

`ModCount`
(R) Database modification count.

`ModDate`
(R) Last modification date.

`RecordCount`
(R) Number of records in this database.

`SortInfoSize`
(R) Size of database SortInfo block in bytes.

`TotalBytes`
(R) Total number of bytes of storage used by this database, including overhead.

`Version`
(R) An application-specific version number of this database.

CommentsS The database properties represent the standard header information plus extended information calculated by the handheld. Be aware that `DataBytes`, `MaxRecordSize`, and `TotalBytes` may take some time to acquire.

See Also `ReadDatabaseInfoByNameCreator()` method.
`DmDatabaseInfo` property.

DmDatabaseQuery

Purpose Represents the collection of **extended databases** on the handheld.

Methods

`AddLogEntry()`
: Adds a text string to the HotSync log on either the desktop or the handheld.

`CreateRecordDatabase()`
: Creates a new extended record database on the handheld.

`OpenRecordDatabase()`
: Opens an extended record database on the handheld.

`ReadDatabaseInfoByNameCreator()`
: Returns a `DmDatabaseInfo` object for an extended database specified by name and creator ID.

`ReadDatabaseNameList()`
: Returns a list of **non-schema database** names that are either in RAM or ROM on the handheld.

`RemoveDatabase()`
: Deletes an extended database on the handheld.

Properties

`MaxAllowedRecordSize`
: (R) Size in bytes of the largest record allowed in an extended database on the handheld.

Comments This is the first object that you need to create for accessing *extended* databases on the handheld. You can open/create as many such databases as you need. Performance degrades, however, if you open more than one extended database at a time.

See Also `DmDatabaseInfo`, `DmRecordAdapter` objects.

DmRecordAdapter

Purpose Represents an open, **extended database**. Its methods can iterate through records in the database serially or access them randomly.

Methods

`AddLogEntry()`
Adds a text string to the HotSync log on either the desktop or the handheld.

`ChangeCategory()`
Changes all records of a particular category to a new category.

`ReadAppInfoBlock()`
Reads this database's application info block.

`ReadById()`
Reads a record using its unique ID.

`ReadByIndex()`
Reads a record using its index.

`ReadNext()`
Reads the next record.

`ReadNextInCategory()`
Reads the next record in a category.

`ReadNextModified()`
Reads the next modified record.

`ReadNextModifiedInCategory()`
Reads the next modified record in a category.

`ReadSortInfoBlock()`
Reads a record database's sort info block.

`ReadUniqueIdList()`
Creates a list of unique IDs in record index order.

`Remove()`
Deletes the specified record from an open extended record database on the handheld.

`RemoveSet()`
Deletes a set of records in an extended database.

`ResetAllModifiedFlags()`
Resets the modified (dirty) flag of all records in the open extended record database on the handheld.

`Write()`
Writes a record in an extended database.

`WriteAppInfoBlock()`
Writes an application info block to an open extended database on the handheld.

`WriteSortInfoBlock()`
Writes a sort information block to an open extended database on the handheld.

Properties

`AccessMode`
(R) Open database access mode.

`CloseOptions`
(R/W) Update database dates on close.

`DbName`
(R) Name of this object's associated database on the handheld.

`DmCategories`
(R) Returns a `DmCategories` object representing the categories in this extended database.

`DmDatabaseInfo`
(R) Returns a `DmDatabaseInfo` object representing information about this extended database.

`EOF`
(R) Database iterator is at the end of the database.

`InputBufferSize`
(R) Size of the buffer to allocate to read record database data.

`IterationIndex`
(R/W) Current starting index for the record data iteration methods.

`RecordCount`
(R) Number of records in this database.

Comments

From a `DmDatabaseQuery` object, you can create a `DmRecordAdapter` object, which represents the extended database that you opened or created.

Reading extended database records can be accomplished in one of two ways. Records can be accessed serially using the iterator methods, or randomly using the direct methods. The iterator methods are named `ReadNextXXX`, and require you to set the `IterationIndex` before using them. The iterator methods are:

- `ReadNext()`
- `ReadNextInCategory()`
- `ReadNextModified()`
- `ReadNextModifiedInCategory()`

Use the `EOF` property to determine when there are no more records in the iteration.

Direct methods to access records randomly are named `ReadByXXX()`, namely `ReadById()` and `ReadByIndex()`.

To close the database, in Visual Basic, set the object reference to the value `Nothing`. In C++, the last `Release` of a `DmRecordAdapter` object closes the database. If you want to update the database dates upon close, set the `CloseOptions` as you wish before closing the database.

IMPORTANT: This note applies to *Visual Studio .NET users only*. Though you can open more than one database at a time, you cannot do so with the same object. For example, if you had `DmRecordAdapter1` to open database "A," you cannot use `DmRecordAdapter1` again to open database "B." You must define a new `DmRecordAdapter` object to open database "B."

See Also

`DmDatabaseQuery`, `DmCategories`, `DmDatabaseInfo`, `PDDatabaseQuery`, `PDCategories`, `PDDatabaseInfo` objects.
`CreateRecordDatabase()`, `OpenRecordDatabase()` methods.
`ERemoveSetType` constants.

PDAddressDbHHRecord

Purpose Represents an Address Book record. Its properties represent the values of the standard Address Book fields.

Methods

`ReadFromByteStream`
Private.

`WriteToByteStream`
Private.

Properties

`Address`
(R/W) Content of the “Address” field in this record.

`CategoryId`
(R/W) Category ID specified by category index.

`City`
(R/W) Content of the “City” field in this record.

`Company`
(R/W) Content of the “Company” field in this record.

`Country`
(R/W) Content of the “Country” field in this record.

`Custom1`
(R/W) Content of the “Custom 1” field in this record.

`Custom2`
(R/W) Content of the “Custom 2” field in this record.

`Custom3`
(R/W) Content of the “Custom 3” field in this record.

`Custom4`
(R/W) Content of the “Custom 4” field in this record.

`DisplayPhone`
(R/W) Contact information to display in the Address Book list view.

`FirstName`
(R/W) Content of the “First name” field in this record.

`Index`
(R/W) Position of this record in its PIM database.

`IsArchived`
: (R/W) Indicates whether this record is marked to be archived.

`IsDeleted`
: (R/W) Indicates whether this record is marked to be deleted.

`IsDirty`
: (R/W) Indicates whether this record is has been modified since the last synchronization.

`IsPrivate`
: (R/W) Indicates whether this record is marked as private.

`LastName`
: (R/W) Content of the “Last name” field in this record.

`Notes`
: (R/W) Content of the note in this record.

`Phone1`
: (R/W) Content of the Phone 1 field in this record.

`Phone2`
: (R/W) Content of the Phone 2 field in this record.

`Phone3`
: (R/W) Content of the Phone 3 field in this record.

`Phone4`
: (R/W) Content of the Phone 4 field in this record.

`Phone5`
: (R/W) Content of the Phone 5 field in this record.

`PhoneLabel1`
: (R/W) Name of the Phone 1 field in this record.

`PhoneLabel2`
: (R/W) Name of the Phone 2 field in this record.

`PhoneLabel3`
: (R/W) Name of the Phone 3 field in this record.

`PhoneLabel4`
: (R/W) Name of the Phone 4 field in this record.

`PhoneLabel5`
: (R/W) Name of the Phone 5 field in this record.

`State`
(R/W) Content of the “State” field in this record.

`Title`
(R/W) Content of the “Title” field in this record.

`UniqueId`
(R/W) The record ID of this record.

`ZipCode`
(R/W) Content of the “Zip Code” field in this record.

Comments For a `PDAddressDbHHRecordAdapter` object, you can create a `PDAddressDbHHRecord` object, which represents the Address Book record that you read or write. Each of this object’s properties is one of the fields in an Address Book record.

Use this object with Address Book versions earlier than 6.0. It does not work with the the application provided in Palm OS® Cobalt.

Example See the example under “PDAddressDbHHRecordAdapter” on page 19.

See Also `PDAddressDbHHRecordAdapter` object.

PDAddressDbHHRecordAdapter

Purpose Represents an open Address Book record database. Its methods can iterate through records in a database serially or access them randomly.

Methods

`AddLogEntry()`
Adds a text string to the HotSync® log on either the desktop or the handheld.

`ChangeCategory()`
Changes all records of a particular category to a new category.

`ReadAppInfoBlock()`
Reads a record database's AppInfo block.

`ReadById()`
Reads a record using its unique ID.

`ReadByIndex()`
Reads a record using its index.

`ReadNext()`
Reads the next record.

`ReadNextInCategory()`
Reads the next record in a category.

`ReadNextModified()`
Reads the next modified record.

`ReadNextModifiedInCategory()`
Reads the next modified record in a category.

`ReadSortInfoBlock()`
Reads a record database's SortInfo block.

`ReadUniqueIdList()`
Creates a list of unique IDs in record index order.

`Remove()`
Deletes the specified record from an open record database on the handheld.

`RemoveSet()`
Deletes a set of database records.

`ResetAllModifiedFlags()`
: Resets the modified (dirty) flag of all records in the opened record database on the handheld.

`Write()`
: Writes a database record.

`WriteAppInfoBlock()`
: Writes an AppInfo block to an open record database on the handheld.

`WriteSortInfoBlock()`
: Writes a sort information block to an open database on the handheld.

Properties

`AccessMode`
: (R) Open database access mode.

`CloseOptions`
: (R/W) Update database dates on close.

`DbName`
: (R) Name of this object's associated database on the handheld.

`EOF`
: (R) Database iterator is at the end of the database.

`InputBufferSize`
: (R) Size of the buffer to allocate to read record database data.

`IterationIndex`
: (R/W) Current starting index for the record data iteration methods.

`PDCategories`
: (R) Returns a `PDCategories` object representing the categories in this database.

`PDDatabaseInfo`
: (R) Returns a `PDDatabaseInfo` object representing information about this database.

`RecordCount`
: (R) Number of records in this database.

Comments

From a `PDDatabaseQuery` object, you can create a `PDAddressDbHHRecordAdapter` object, which represents the Address Book record database that you opened or created. With this object, you can access Address Book records represented by `PDAddressDbHHRecord` objects.

Use this object with Address Book versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

When you *open* an Address Book database with this object, you must specify both the database's name and the full adapter name of this object:

```
OpenRecordDatabase("AddressDB", _
    "PDStandard.PDAddressDbHHRecordAdapter", _
    eRead Or eWrite Or eShowSecret)
```

When you *create* an Address Book database with this object, you must additionally specify the Address Book database's creator ID (`'addr'`) and type (`'DATA'`):

```
CreateRecordDatabase("AddressDB", _
    "PDStandard.PDAddressDbHHRecordAdapter", _
    "addr", "DATA", eRead Or eWrite, eBackupDb, _
    1, 0)
```

Reading database records can be accomplished in one of two ways. Records can be accessed serially using the iterator methods, or randomly using the direct methods. The iterator methods are named `ReadNextXXX`, and require you to set the `IterationIndex` before using them. The iterator methods are:

- `ReadNext()`
- `ReadNextInCategory()`
- `ReadNextModified()`
- `ReadNextModifiedInCategory()`

Use the `EOF` property to determine when there are no more records in the iteration.

Direct methods to access records randomly are named `ReadByXXX()`, namely `ReadById()` and `ReadByIndex()`.

To close the database, in Visual Basic, set the object reference to the value `Nothing`. In C++, the last `Release` of a `PDRecordAdapter` object closes the database. If you want to update the database dates upon close, set the `CloseOptions` as you wish before closing the database.

IMPORTANT: This note applies to *Visual Studio .NET users only*. Though you can open more than one database at a time, you cannot do so with the same object. For example, if you had `PDAddressDbHHRecordAdapter1` to open database "A," you cannot use `PDAddressDbHHRecordAdapter1` again to open database "B." You must define a new `PDAddressDbHHRecordAdapter` object to open database "B."

Example

```
Dim pDbQuery As New PDDatabaseQuery
' Declare the PDAddressDbHHRecordAdapter object.
Dim pAddr As PDAddressDbHHRecordAdapter

' Open the AddressDb database.
Set pAddr = pDbQuery.OpenRecordDatabase("AddressDB", _
   "PDStandard.PDAddressDbHHRecordAdapter", eRead Or eWrite _
   Or eShowSecret)

' Declare the record header and data.
Dim nIndex As Long
Dim vUniqueId As Variant
Dim nCategory As Long
Dim eAttributes As ERecordAttributes
Dim vData As Variant
' Declare the count of records containing the string
Dim nCount As Long
Dim nTest As Variant

' Declare the PDAddressDbHHRecord object, set its properties,
' and write it.
Dim pAddressRecord As New PDAddressDbHHRecord
pAddressRecord.City = "Sunnyvale"
pAddressRecord.Address = "1240 Crossman Ave."
pAddressRecord.Company = "PalmSource, Inc."
pAddressRecord.Country = "USA"
pAddressRecord.Custom1 = "custom1"
pAddressRecord.Custom2 = "custom2"
pAddressRecord.Custom3 = "custom3"
pAddressRecord.Custom4 = "custom4"
pAddressRecord.DisplayPhone = 1
pAddressRecord.FirstName = "Albert"
pAddressRecord.LastName = "Einstein"
pAddressRecord.Notes = "E = mc^2"
pAddressRecord.State = "CA"
pAddressRecord.Title = "Theoretical Physicist"
pAddressRecord.Phone1 = "ae@palmsource.com"
pAddressRecord.Phone2 = "408-123-4567"
pAddressRecord.Phone3 = "408-123-8901"
pAddressRecord.Phone4 = "408-123-2345"
pAddressRecord.Phone5 = "408-123-6789"
pAddressRecord.CategoryId = 2
pAddressRecord.PhoneLabel1 = PHONE_LABEL_EMAIL
pAddressRecord.PhoneLabel2 = PHONE_LABEL_MOBILE
pAddressRecord.PhoneLabel3 = PHONE_LABEL_FAX
pAddressRecord.PhoneLabel4 = PHONE_LABEL_MAIN
pAddressRecord.PhoneLabel5 = PHONE_LABEL_PAGER

vUniqueId = pAddr.Write(pAddressRecord)
```

See Also `PDDatabaseQuery`, `PDCategories`, `PDDatabaseInfo`, `PDAddressDbHHRecord` objects.
`CreateRecordDatabase()`, `OpenRecordDatabase()` methods.
`ERemoveSetType` constants.

PDCategories

Purpose This utility object supports access to category information in a classic database.

Methods

`Refresh()`
Reinitializes this object from its source, discarding any changes in the cache.

`ResetDirtyFlags()`
Resets all the category `Dirty` flags to `False`.

`Save()`
Writes the category information into the AppInfo block of this database and writes the AppInfo block to the handheld.

Properties

`CategoryId`
(R/W) Category ID specified by category index.

`DbName`
(R) Name of this object's associated database on the handheld.

`Dirty`
(R/W) Category dirty flag specified by category index.

`LastId`
(R/W) Category ID of the last new category.

`Name`
(R/W) Category name specified by category index.

Comments Uses the first 275 bytes of the **application info** block and converts them to a categories structure. Provides methods and properties to manage the categories structure. When saved, merges them back into the application info block.

Each record in a handheld database can have a category index. The definition of all categories in a handheld database is stored within the database's application info block.

The `PDCategories` object provides a wrapper for this standard categories structure. Each category entry in the standard category structure contains three elements: `Name`, `CategoryId`, and `Dirty` flag. The `PDCategories` object caches the category information, and updates the handheld database only when the `Save()` method is executed. For C++, the following programming structure describes the category structure.

```
typedef struct tagCategories {
   WORD renamedCategories;
   char categoryLabels [16] [16];
   BYTE categoryUniqIDs[16];
   BYTE lastUniqID;
} Categories;
```

See Also `PDRecordAdapter` object.

PDCondMgr

Purpose A collection of utility methods that register conduits and notifiers with HotSync Manager for the current Windows user. These methods also manage information about these conduits and notifiers.

Methods

`CreatorIDToString()`
Converts a `Long` conduit creator ID into a `String`.

`GetBackupConduit()`
Retrieves the name of HotSync Manager's **backup conduit** for the current Windows user.

`GetConduitCount()`
Returns the number of conduits registered with HotSync Manager for the current Windows user.

`GetConduitInfo()`
Returns complete information about a user-registered conduit in a `PDConduitInfo` object.

`GetConduitList()`
Returns a list of creator IDs of all the user-registered conduits.

`GetDWORDData()`
Retrieves a `DWORD` configuration entry value for the specified user-registered conduit.

`GetNotifierList()`
Returns a list of all the user-registered notifier filenames.

`GetStringData()`
Retrieves a `String` configuration entry value for the specified user-registered conduit.

`ModifyNotifier()`
Modifies the path or filename of a notifier already registered with HotSync Manager for the current Windows user.

`RegisterConduit()`
Registers a conduit for the current Windows user based on the information provided in a `PDConduitInfo` object.

`RegisterNotifier()`
Registers a notifier with HotSync Manager for the current Windows user.

`SetBackupConduit()`
Sets the filename of the HotSync Manager backup conduit for the current Windows user.

`SetDWORDData()`
Sets a `DWORD` configuration entry value for the specified user-registered conduit.

`SetStringData()`
Sets a `String` configuration entry value for the specified user-registered conduit.

`StringToCreatorID()`
Converts a `String` into a `DWORD` conduit creator ID.

`UnregisterConduit()`
Unregisters a user-registered conduit with HotSync Manager.

`UnregisterNotifier()`
Unregisters a user-registered notifier with HotSync Manager.

Properties None.

Comments The member methods of this object access the underlying Conduit Manager C API.

This object manages conduits and notifiers for the current Windows user. To manage system-registered conduits, use `PDSystemCondMgr`.

See Also `PDConduitInfo`, `PDSystemCondMgr` object.
"Registering Conduits and Notifiers with HotSync Manager" on page 73 in *Introduction to Conduit Development*.

PDConduitInfo

Purpose Represents all the conduit information to register a conduit with HotSync Manager.

Methods None. Use methods defined in `PDCondMgr` and `PDSystemCondMgr` to access `PDConduitInfo` properties.

Properties

`COMClassID`
(R/W) ProgID of this COM-based conduit.

`CreatorID`
(R/W) Creator ID of the application on the handheld that this conduit is responsible for synchronizing.

`DeskTopDataDirectory`
(R/W) Name of this conduit's data directory.

`DeskTopDataFile`
(R/W) Name of the desktop data file that your conduit synchronizes with the handheld database.

`DisplayName`
(R/W) User-visible name of this conduit.

`FileName`
(R/W) Filename of this conduit DLL.

`HandHeldDB`
(R/W) Name of the database on the handheld that this conduit accesses.

`JavaClassName`
(R/W) Full name of the Java-based conduit class (including package).

`JavaClassPath`
(R/W) Directory that contains all the classes used by this Java-based conduit.

`Priority`
(R/W) Execution priority for this conduit.

Comments The member properties of this object correspond to the conduit configuration entries created on the desktop when you register a conduit. They map closely to the entries you create when you use the Conduit Configuration (CondCfg) utility to register your conduit on your development and test machine (see Chapter 3, "Conduit Configuration Utility," on page 11 in the *Conduit Development Utilities Guide*).

See Also `PDCondMgr`, `PDSystemCondMgr` objects.

PDDatabaseInfo

Purpose Represents information about a **non-schema database**.

Method `Refresh()`

Reinitializes this object from its source, discarding any changes in the cache.

Properties `AppInfoSize`

(R) AppInfo block size of this database.

`BackupDate`

(R) Date that this database was last backed up.

`CardNum`

(R) Card number of the memory card on the handheld on which databases are stored.

`CreateDate`

(R) Creation date of this database.

`Creator`

(R) Creator ID of this database on the handheld.

`DataBytes`

(R) Number of bytes of storage used by this database for data only, excluding overhead.

`DbFlags`

(R) Database flags that are set at creation time.

`DbIndex`

(R) Database index in the total set of databases.

`DbName`

(R) Name of this object's associated database on the handheld.

`DbType`

(R) Database type.

`ExcludeFromSync`

(R) Determines whether this database is excluded from synchronization.

`IsRam`

(R) Determines whether a database is stored in RAM or ROM.

`MaxRecordSize`
: (R) Size of the largest record in this database.

`ModCount`
: (R) Database modification count.

`ModDate`
: (R) Last modification date.

`RecordCount`
: (R) Number of records in this database.

`SortInfoSize`
: (R) Size of database SortInfo block in bytes.

`TotalBytes`
: (R) Total number of bytes of storage used by this database, including overhead.

`Version`
: (R) An application-specific version number of this database.

CommentsS The database properties represent the standard header information plus extra information calculated by the handheld. Be aware that `DataBytes`, `MaxRecordSize`, and `TotalBytes` may take some time to acquire.

IMPORTANT: Use only the `PDDatabaseInfo` object available in the `PDStandard` library. Do not use the `PDDatabaseInfo` object available in the `PDirect` library.

See Also `ReadDbInfoByName()`, `ReadDbInfoByCreatorType()` methods.
`PDDatabaseInfo`, `DmDatabaseInfo` property.

PDDatabaseQuery

Purpose Represents the collection of **classic databases** on the handheld.

Methods

`AddLogEntry()`
: Adds a text string to the HotSync® log on either the desktop or the handheld.

`CreateRecordDatabase()`
: Creates a new record database on the handheld.

`CreateResourceDatabase()`
: Creates a new resource database on the handheld.

`OpenRecordDatabase()`
: Opens a record database on the handheld.

`OpenResourceDatabase()`
: Opens a resource database on the handheld.

`ReadDbInfoByName()`
: Returns a `PDDatabaseInfo` object for a named database.

`ReadDbInfoByCreatorType()`
: Returns a `PDDatabaseInfo` object for a creator/type pair.

`ReadDbNameList()`
: Returns a list of classic database names that are either in RAM or ROM on the handheld.

`RemoveDatabase()`
: Deletes a database on the handheld.

Properties

`MaxAllowedRecordSize`
: (R) Size of the largest record allowed on the handheld in bytes.

`RamDbCount`
: (R) Number of databases in primary storage RAM on the handheld.

`RomDbCount`
: (R) Number of databases in ROM on the handheld.

Comments This is the first object that you need to create for accessing classic databases on the handheld. You can open/create as many classic databases as you need. But because the underlying Sync Manager API permits only one classic database to be open at a time, performance degrades if you intermix record access from more than one classic database.

To work with extended databases, use `DmDatabaseQuery`. To work with schema databases, use `PSDDatabaseQuery`.

See Also `PDDatabaseInfo`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter`, `PDResourceAdapter` objects.

PDDateBookDbHHRecord

Purpose Represents a classic Date Book record that does not support exceptions to repeating events. Use the `PDDateBookDbHHRecord2` object instead of this object.

Methods

`ReadFromByteStream`
Private.

`WriteToByteStream`
Private.

Properties

`AlarmAdvanceTime`
(R/W) How long before an event to trigger the alarm.

`AlarmAdvanceUnits`
(R/W) Time units that the `AlarmAdvanceTime` property is specified in.

`DaysMaskForWeeklyRepeat`
(R/W) Mask indicating which days of the week on which a weekly repeating event occurs.

`Description`
(R/W) Text describing an event.

`EndTime`
(R/W) Time and date on which an event ends.

`Index`
(R/W) Position of this record in the Date Book database.

`IsAlarmSet`
(R/W) Indicates whether the alarm is set for this event.

`IsArchived`
(R/W) Indicates whether this record is marked to be archived.

`IsDeleted`
(R/W) Indicates whether this record is marked to be deleted.

`IsDirty`
(R/W) Indicates whether this record is has been modified since the last synchronization.

`IsEventNotTimed`
(R/W) Indicates whether a time is specified for this event.

`IsEventRepeatable`
(R/W) Indicates whether this event repeats in.

`IsPrivate`
(R/W) Indicates whether this record is marked as private.

`Notes`
(R/W) Content of the note in this record.

`RepeatDay`
(R/W) Day on which to repeat this event each month.

`RepeatEndDate`
(R/W) Date on which to end this repeating event.

`RepeatFrequency`
(R/W) How many cycles between instances of this repeating event.

`RepeatType`
(R/W) Cycle on which this event repeats in.

`StartTime`
(R/W) Time and date on which an event starts.

`UniqueId`
(R/W) The record ID of this record.

`WeekIndexForMonthlyRepeatByDay`
(R/W) Week on which to repeat this event if it repeats monthly by day.

Comments For a `PDDateBookDbHHRecordAdapter` object, you can create a `PDDateBookDbHHRecord` object, which represents the Date Book record that you read or write. Each of this object's properties is one of the fields in a Date Book record.

Use this object with Date Book versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

NOTE: To handle exception dates in Date Book records, you must use `PDDateBookDbHHRecordAdapter2` and `PDDateBookDbHHRecord2`. The `PDDateBookDbHHRecordAdapter` and `PDDateBookDbHHRecord` objects do not support exceptions dates in repeating events.

Example See the example under “PDDateBookDbHHRecordAdapter” on page 40.

See Also `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecordAdapter` objects.

PDDateBookDbHHRecord2

Purpose Represents a classic Date Book record that supports exceptions to repeating events. Use this object rather than `PDDateBookDbHHRecord`.

Methods

`GetExceptionDates()`
Retrieves a list of dates that are exceptions to a Date Book repeating event.

`ReadFromByteStream`
Private.

`SetExceptionDates()`
Sets the exception dates for a repeating event in Date Book.

`WriteToByteStream`
Private.

Properties

`AlarmAdvanceTime`
(R/W) How long before an event to trigger the alarm.

`AlarmAdvanceUnits`
(R/W) Time units that the `AlarmAdvanceTime` property is specified in.

`DaysMaskForWeeklyRepeat`
(R/W) Mask indicating which days of the week on which a weekly repeating event occurs.

`Description`
(R/W) Text describing an event.

`EndTime`
(R/W) Time and date on which an event ends.

`Index`
(R/W) Position of this record in the Date Book database.

`IsAlarmSet`
(R/W) Indicates whether the alarm is set for this event.

`IsArchived`
(R/W) Indicates whether this record is marked to be archived.

`IsDeleted`
(R/W) Indicates whether this record is marked to be deleted.

`IsDirty`
(R/W) Indicates whether this record is has been modified since the last synchronization.

`IsEventNotTimed`
(R/W) Indicates whether a time is specified for this event.

`IsEventRepeatable`
(R/W) Indicates whether this event repeats in.

`IsPrivate`
(R/W) Indicates whether this record is marked as private.

`Notes`
(R/W) Content of the note in this record.

`RepeatDay`
(R/W) Day on which to repeat this event each month.

`RepeatEndDate`
(R/W) Date on which to end this repeating event.

`RepeatFrequency`
(R/W) How many cycles between instances of this repeating event.

`RepeatType`
(R/W) Cycle on which this event repeats in.

`StartTime`
(R/W) Time and date on which an event starts.

`UniqueId`
(R/W) The record ID of this record.

`WeekIndexForMonthlyRepeatByDay`
(R/W) Week on which to repeat this event if it repeats monthly by day.

Comments For a `PDDateBookDbHHRecordAdapter2` object, you can create a `PDDateBookDbHHRecord2` object, which represents the Date Book record that you read or write. Each of this object's properties is one of the fields in a Date Book record.

Use this object with Date Book versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

NOTE: To handle exception dates in Date Book records, you must use `PDDateBookDbHHRecordAdapter2` and `PDDateBookDbHHRecord2`. The `PDDateBookDbHHRecordAdapter` and `PDDateBookDbHHRecord` objects do not support exceptions dates in repeating events.

Example See the example under "PDDateBookDbHHRecordAdapter2" on page 45.

See Also `PDDateBookDbHHRecordAdapter2` object.

PDDateBookDbHHRecordAdapter

Purpose Represents an open Date Book record database. Its methods can iterate through records in a database serially or access them randomly.

Methods

`AddLogEntry()`
Adds a text string to the HotSync log on either the desktop or the handheld.

`ChangeCategory()`
Changes all records of a particular category to a new category.

`ReadAppInfoBlock()`
Reads a record database's AppInfo block.

`ReadById()`
Reads a record using its unique ID.

`ReadByIndex()`
Reads a record using its index.

`ReadNext()`
Reads the next record.

`ReadNextInCategory()`
Reads the next record in a category.

`ReadNextModified()`
Reads the next modified record.

`ReadNextModifiedInCategory()`
Reads the next modified record in a category.

`ReadSortInfoBlock()`
Reads a record database's SortInfo block.

`ReadUniqueIdList()`
Creates a list of unique IDs in record index order.

`Remove()`
Deletes the specified record from an open record database on the handheld.

`RemoveSet()`
Deletes a set of database records.

`ResetAllModifiedFlags()`
Resets the modified (dirty) flag of all records in the opened record database on the handheld.

`Write()`
Writes a database record.

`WriteAppInfoBlock()`
Writes an AppInfo block to an open record database on the handheld.

`WriteSortInfoBlock()`
Writes a sort information block to an open database on the handheld.

Properties

`AccessMode`
(R) Open database access mode.

`CloseOptions`
(R/W) Update database dates on close.

`DbName`
(R) Name of this object's associated database on the handheld.

`EOF`
(R) Database iterator is at the end of the database.

`InputBufferSize`
(R) Size of the buffer to allocate to read record database data.

`IterationIndex`
(R/W) Current starting index for the record data iteration methods.

`PDCategories`
(R) Returns a `PDCategories` object representing the categories in this database.

`PDDatabaseInfo`
(R) Returns a `PDDatabaseInfo` object representing information about this database.

`RecordCount`
(R) Number of records in this database.

Comments

From a `PDDatabaseQuery` object, you can create a `PDDateBookDbHHRecordAdapter` object, which represents the Date Book record database that you opened or created. With this object, you can access Date Book records represented by `PDDateBookDbHHRecord` objects.

Use this object with Date Book versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

NOTE: To handle exception dates in Date Book records, you must use `PDDateBookDbHHRecordAdapter2` and `PDDateBookDbHHRecord2`. The `PDDateBookDbHHRecordAdapter` and `PDDateBookDbHHRecord` objects do not support exceptions dates in repeating events.

When you *open* a Date Book database with this object, you must specify both the database's name and the full adapter name of this object:

```
OpenRecordDatabase("DatebookDB", _
   "PDStandard.PDDateBookDbHHRecordAdapter", _
   eRead Or eWrite Or eShowSecret)
```

When you *create* a Date Book database with this object, you must additionally specify the Date Book database's creator ID (`'date'`) and type (`'DATA'`):

```
CreateRecordDatabase("DatebookDB", _
   "PDStandard.PDDateBookDbHHRecordAdapter", _
   "date", "DATA", eRead Or eWrite, eBackupDb, _
   1, 0)
```

Reading database records can be accomplished in one of two ways. Records can be accessed serially using the iterator methods, or randomly using the direct methods. The iterator methods are named `ReadNextXXX()`, and require you to set the `IterationIndex` before using them. The iterator methods are:

- `ReadNext()`
- `ReadNextInCategory()`
- `ReadNextModified()`

- ReadNextModifiedInCategory()

Use the EOF property to determine when there are no more records in the iteration.

Direct methods to access records randomly are named `ReadByXXX`, namely ReadById() and ReadByIndex().

To close the database, in Visual Basic, set the object reference to the value `Nothing`. In C++, the last `Release` of a `PDRecordAdapter` object closes the database. If you want to update the database dates upon close, set the CloseOptions as you wish before closing the database.

IMPORTANT: This note applies to *Visual Studio .NET users only*. Though you can open more than one database at a time, you cannot do so with the same object. For example, if you had `PDDateBookDbHHRecordAdapterA` to open database "A," you cannot use `PDDateBookDbHHRecordAdapterA` again to open database "B." You must define a new `PDDateBookDbHHRecordAdapter` object to open database "B."

Example

```
' Declare the PDDatabaseQuery object.
Dim pDbQuery As New PDDatabaseQuery
Dim PDateRecord As New PDDateBookDbHHRecord
   'Declare the PDDateBookDbRecordAdapter object
   Dim pDateAdapter As PDDateBookDbHHRecordAdapter

' First check whether a HotSync operation is in progress.
Dim PHotSyncManager As New PDHotSyncUtility

If Not PHotSyncManager.IsSyncInProgress Then
   MsgBox "COM Sync Suite objects are only active during a _
      HotSync operation. Refer to the COMSyncCompanion.pdf _
   manual for more information on how to create a debug _
   environment.", vbInformation, "Information"
   Exit Sub
End If

' Open the DateBookDB database.
Set pDateAdapter = pDbQuery.OpenRecordDatabase("DatebookDB",_
   "PDStandard.PDDatebookDbHHRecordAdapter", eRead Or eWrite_
   Or eShowSecret)
```

```
' Fill in record data.
PDateRecord.Description = "Test Record"
PDateRecord.StartTime = "07/19/2002 9:00:00 AM"
PDateRecord.EndTime = "07/19/2002 9:15:00 AM"

' Set alarm info.: trigger reminder 10 min before the event.
PDateRecord.IsAlarmSet = True
PDateRecord.AlarmAdvanceTime = 10
PDateRecord.AlarmAdvanceUnits = PD_AAU_MINUTES

PDateRecord.Notes = "My Notes"

' Set repeat info.: repeat every month on third Friday.
PDateRecord.IsEventRepeatable = True
PDateRecord.RepeatEndDate = "07/19/2003"
PDateRecord.RepeatFrequency = 1
PDateRecord.RepeatType = EPDMonthlyByDay
PDateRecord.WeekIndexForMonthlyRepeatByDay = EPDThird
PDateRecord.RepeatDay = EPDFriday

' Write the record.
Dim uniqueid As Variant
uniqueid = pDateAdapter.Write(PDateRecord)
   MsgBox "successfully created a datebook event which _
      repeats every month on third friday from July 2002 to _
      July 2003"
```

See Also PDDateBookDbHHRecordAdapter2, PDDatabaseQuery, PDCategories, PDDatabaseInfo, PDDateBookDbHHRecord objects.
CreateRecordDatabase(), OpenRecordDatabase() methods.
ERemoveSetType constants.

PDDateBookDbHHRecordAdapter2

Purpose Represents an open Date Book record database that supports exceptions to repeating events. Use this object rather than `PDDateBookDbHHRecordAdapter`. This object's methods can iterate through records in a database serially or access them randomly.

Methods

`AddLogEntry()`
Adds a text string to the HotSync log on either the desktop or the handheld.

`ChangeCategory()`
Changes all records of a particular category to a new category.

`ReadAppInfoBlock()`
Reads a record database's application info block.

`ReadById()`
Reads a record using its unique ID.

`ReadByIndex()`
Reads a record using its index.

`ReadNext()`
Reads the next record.

`ReadNextInCategory()`
Reads the next record in a category.

`ReadNextModified()`
Reads the next modified record.

`ReadNextModifiedInCategory()`
Reads the next modified record in a category.

`ReadSortInfoBlock()`
Reads a record database's SortInfo block.

`ReadUniqueIdList()`
Creates a list of unique IDs in record index order.

`Remove()`
Deletes the specified record from an open record database on the handheld.

`RemoveSet()`
Deletes a set of database records.

`ResetAllModifiedFlags()`
Resets the modified (dirty) flag of all records in the opened record database on the handheld.

`Write()`
Writes a database record.

`WriteAppInfoBlock()`
Writes an AppInfo block to an open record database on the handheld.

`WriteSortInfoBlock()`
Writes a sort information block to an open database on the handheld.

Properties

`AccessMode`
(R) Open database access mode.

`CloseOptions`
(R/W) Update database dates on close.

`DbName`
(R) Name of this object's associated database on the handheld.

`EOF`
(R) Database iterator is at the end of the database.

`InputBufferSize`
(R) Size of the buffer to allocate to read record database data.

`IterationIndex`
(R/W) Current starting index for the record data iteration methods.

`PDCategories`
(R) Returns a `PDCategories` object representing the categories in this database.

`PDDatabaseInfo`
(R) Returns a `PDDatabaseInfo` object representing information about this database.

`RecordCount`
(R) Number of records in this database.

Comments

From a `PDDatabaseQuery` object, you can create a `PDDateBookDbHHRecordAdapter2` object, which represents the Date Book record database that you opened or created. With this object, you can access Date Book records represented by `PDDateBookDbHHRecord2` objects.

Use this object with Date Book versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

NOTE: To handle exception dates in Date Book records, you must use `PDDateBookDbHHRecordAdapter2` and `PDDateBookDbHHRecord2`. The `PDDateBookDbHHRecordAdapter` and `PDDateBookDbHHRecord` objects do not support exceptions dates in repeating events.

When you *open* a Date Book database with this object, you must specify both the database's name and the full adapter name of this object:

```
OpenRecordDatabase("DatebookDB", _
    "PDStandard.PDDateBookDbHHRecordAdapter2", _
    eRead Or eWrite Or eShowSecret)
```

When you *create* a Date Book database with this object, you must additionally specify the Date Book database's creator ID (`'date'`) and type (`'DATA'`):

```
CreateRecordDatabase("DatebookDB", _
    "PDStandard.PDDateBookDbHHRecordAdapter2", _
    "date", "DATA", eRead Or eWrite, eBackupDb, _
    1, 0)
```

Reading database records can be accomplished in one of two ways. Records can be accessed serially using the iterator methods, or randomly using the direct methods. The iterator methods are named `ReadNextXXX()`, and require you to set the `IterationIndex` before using them. The iterator methods are:

- `ReadNext()`
- `ReadNextInCategory()`
- `ReadNextModified()`

- `ReadNextModifiedInCategory()`

Use the `EOF` property to determine when there are no more records in the iteration.

Direct methods to access records randomly are named `ReadByXXX()`, namely `ReadById()` and `ReadByIndex()`.

To close the database, in Visual Basic, set the object reference to the value `Nothing`. In C++, the last `Release` of a `PDRecordAdapter` object closes the database. If you want to update the database dates upon close, set the `CloseOptions` as you wish before closing the database.

IMPORTANT: This note applies to *Visual Studio .NET users only*. Though you can open more than one database at a time, you cannot do so with the same object. For example, if you had `PDDateBookDbHHRecordAdapter2A` to open database “A,” you cannot use `PDDateBookDbHHRecordAdapter2A` again to open database "B." You must define a new `PDDateBookDbHHRecordAdapter2` object to open database “B.”

Example

```
' Declare the PDDatabaseQuery object.
Dim pDbQuery As New PDDatabaseQuery
Dim PDateRecord As New PDDateBookDbHHRecord2
   'Declare the PDDateBookDbRecordAdapter object
   Dim pDateAdapter As PDDateBookDbHHRecordAdapter2

' First check whether a HotSync operation is in progress.
Dim PHotSyncManager As New PDHotSyncUtility

If Not PHotSyncManager.IsSyncInProgress Then
   MsgBox "COM Sync Suite objects are only active during a _
      HotSync operation. Refer to the COMSyncCompanion.pdf _
   manual for more information on how to create a debug _
   environment.", vbInformation, "Information"
   Exit Sub
End If

' Open the DateBookDB database.
Set pDateAdapter = pDbQuery.OpenRecordDatabase("DatebookDB",_
   "PDStandard.PDDatebookDbHHRecordAdapter2", eRead Or _
   eWrite Or eShowSecret)
```

```
' Fill in record data.
PDateRecord.Description = "Test Record"
PDateRecord.StartTime = "07/19/2002 9:00:00 AM"
PDateRecord.EndTime = "07/19/2002 9:15:00 AM"

' Set alarm info.: trigger reminder 10 min before the event.
PDateRecord.IsAlarmSet = True
PDateRecord.AlarmAdvanceTime = 10
PDateRecord.AlarmAdvanceUnits = PD_AAU_MINUTES

PDateRecord.Notes = "My Notes"

' Set repeat info.: repeat every month on third Friday.
PDateRecord.IsEventRepeatable = True
PDateRecord.RepeatEndDate = "07/19/2003"
PDateRecord.RepeatFrequency = 1
PDateRecord.RepeatType = EPDMonthlyByDay
PDateRecord.WeekIndexForMonthlyRepeatByDay = EPDThird
PDateRecord.RepeatDay = EPDFriday

' Write the record.
Dim uniqueid As Variant
uniqueid = pDateAdapter.Write(PDateRecord)
   MsgBox "successfully created a datebook event which _
      repeats every month on third friday from July 2002 to _
      July 2003"
```

See Also PDDatabaseQuery, PDCategories, PDDatabaseInfo, PDDateBookDbHHRecord2 objects.
CreateRecordDatabase(), OpenRecordDatabase() methods.
ERemoveSetType constants.

PDExpansionCardInfo

Purpose Represents expansion card information given a slot reference number.

Methods None.

Properties `CapabilityFlags`

Describes the capabilities of an expansion card, such as whether it has storage and whether it is read-only.

`DeviceClass`

Describes the name of the type of expansion card.

`DeviceUniqueId`

Unique identifier for an expansion card product.

`ManufacturerName`

Name of the manufacturer of the expansion card.

`MediaType`

Type of media supported by the expansion card.

`ProductName`

Name of the expansion card product.

Comments These read-only properties specify the characteristics of the expansion card loaded in the slot, including whether the card supports secondary storage or is strictly read-only.

See Also `PDExpansionManager` object.

PDExpansionManager

Purpose Represents the Expansion Manager on the desktop. Its methods can detect the presence of and get information about expansion slots on a handheld and the cards in them. It can get slot reference numbers, which you subsequently pass to its methods to gather card information.

Methods

`GetCardInfo()`
Retrieves information about an expansion card in a given slot.

`GetSlotInfo()`
Retrieves information about a specified expansion slot, including the reference number of a mounted volume.

`GetSlotReferenceNumbers()`
Retrieves a list of slot reference numbers on a handheld.

`IsExpansionSlotPresent()`
Verifies the presence of an expansion slot on the handheld.

Properties None.

Comments The Expansion Manager on the handheld is an optional system extension that adds support for hardware expansion cards on Palm Powered™ handhelds. The handheld Expansion Manager's primary function is to manage slots on the handheld and the drivers associated with those slots. Individual slot drivers on the handheld—which are provided by handheld manufacturers—provide support for various expansion card types including Secure Digital (SD), MultiMediaCard (MMC), CompactFlash, Sony's Memory Stick, and others.

The `PDExpansionManager` object provides conduits an interface to the Expansion Manager on the handheld during a HotSync operation. Through this interface, conduits can determine whether an expansion card is present in a slot and get information about that card.

See Also `PDExpansionCardInfo` object.

PDHotsyncInfo

Purpose Represents information about the current HotSync session.

Methods None.

Properties `CardNum`

(R) The number of the **memory card** on which the database is stored.

`ConnectionType`

(R) An `EConnectionType` value that indicates the type transfer medium of the current HotSync operation.

`Creator`

(R) The **creator ID** associated with the current conduit.

`DbType`

(R) The **database type**.

`FirstSync`

(R) An `EFirstSync` value that indicates whether the current HotSync operation is the first for the handheld, the first with the current desktop, or the first for neither.

`LocalName`

(R) The desktop file that the conduit synchronizes with. This value is set in the conduit's `File` configuration entry.

`NameList`

(R) List of the handheld databases that have the same creator ID as the current conduit. The number of items in the array is specified by the `RemoteNameCount` property.

`PathName`

(R) The conduit's directory name. This value is set in the conduit's `Directory` configuration entry.

`RegistryKey`

(R) The primary Windows registry key for the current conduit. Do not use this property; use the `PDConduitInfo` to access conduit configuration entries instead.

`RegistryPath`

(R) The full Windows registry path of the current conduit. Do not use this property; use the `PDConduitInfo` object to access conduit configuration entries instead.

`RemoteNameCount`
(R) The number of entries in the conduit's database `NameList` property.

`SyncType`
(R) Synchronization type, which is one of the `ESyncTypes` constants.

`UserName`
(R) The HotSync user name of the user who is currently performing a HotSync operation.

Comments The values of these properties are passed in to a conduit when HotSync Manager calls its `BeginProcess()` entry point.

See Also `PDHotsyncInfo` property.
`EConnectionType`, `EFirstSync`, `ESyncTypes` constants.

PDHotSyncUtility

Purpose A collection of utility methods for controlling the HotSync Manager application.

Methods

`DisplayLog()`
: Displays the **HotSync Log** dialog box of the HotSync Manager application.

`GetCommStatus()`
: Retrieves the status of the HotSync Manager application's communication types.

`IsSyncInProgress()`
: Determines whether the HotSync Manager application is currently busy synchronizing a handheld.

`LaunchCustomDlg()`
: Displays the **Custom** dialog box of the HotSync Manager application.

`LaunchFileLinkDlg()`
: *(Deprecated)* Displays the **File Link** wizard of the HotSync Manager application.

`LaunchSetupDlg()`
: Displays the **Setup** dialog box of the HotSync Manager application.

`RefreshConduitInfo()`
: Requests that HotSync Manager reload information about all registered conduits.

`ResetComm()`
: Resets the communication methods of the HotSync Manager application.

`RestartHotSyncMgr()`
: Restarts the HotSync Manager application.

`SetCommStatus()`
: Sets the status of the HotSync Manager application's communication types.

`StartHotSyncMgr()`
: Starts the HotSync Manager application.

`TerminateHotSyncMgr()`
: Closes the HotSync Manager application.

Properties	None.
Comments	The member methods of this object access the underlying HotSync Manager C API.
See Also	`PDHotsyncInfo` object.

PDInstall

Purpose A collection of 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.

Methods

`ChangeFileDestinationHHToSlot()`
Changes the destination of a file that is already queued to be installed in primary storage on a user's *handheld* instead to be installed in secondary storage in an expansion *slot*.

`ChangeFileDestinationSlotToHH()`
Changes the destination of a file that is already queued to be installed in secondary storage in an expansion *slot* instead to be installed in primary storage on a user's *handheld*.

`ChangeFileSlotDestination()`
Changes from one expansion *slot* to another the destination of a file that is already queued to be installed in secondary storage in an expansion *slot* of a user's handheld.

`GetAllQueuedHHFiles()`
Retrieves a list of all the files queued to be installed in the handheld's main memory for the specified user.

`GetAllQueuedHHFilesOfType()`
Retrieves a list of all the files of the specified *type* that are queued to be installed in the handheld's main memory for the specified user.

`GetAllQueuedSlotFiles()`
Retrieves a list of all the files queued to be installed to the handheld's specified expansion slot for a given user.

`GetHHFileSize()`
Retrieves the size of the specified file that is queued to be installed to the handheld's main memory for a given user.

`GetPath()`
Retrieves one of the stored desktop paths.

`GetSlotFileCount()`
Retrieves the number of files queued to install to the specified slot for a given user.

`GetSlotFileSize()`
Retrieves the size of the specified file queued to be installed to the handheld's specified expansion slot for a given user.

`InstallFileToHH()`
Queues a file to be installed in primary storage on a user's handheld.

`InstallFileToSlot()`
Queues a file to be installed in secondary storage in an expansion slot of a user's handheld.

`RemoveFileFromHHQueue()`
Removes a file from the queue of files that are to be installed in primary storage on a user's *handheld*.

`RemoveFileFromSlotQueue()`
Removes a file from the queue of files that are to be installed in secondary storage in an expansion *slot* of a user's handheld.

`SetPath()`
Sets the value of one of the stored path variables.

Properties None.

Comments The member methods of this object access the underlying Install Aide C API.

PDInstallConduit

Purpose A collection of utility methods for registering an install conduit with HotSync Manager and managing install conduit information.

Methods

`GetDWORDData()`
Retrieves a `DWORD` configuration entry value for the specified conduit.

`GetStringData()`
Retrieves a `String` configuration entry value for the specified conduit.

`RegisterIC()`
Registers an **install conduit** based on the information provided in a `PDInstallConduitInfo` object.

`SetDWORDData()`
Sets a `DWORD` configuration entry value for the specified conduit.

`SetStringData()`
Sets a `String` configuration entry value for the specified conduit.

`UnregisterIC()`
Unregisters an install conduit with HotSync Manager.

Properties None.

Comments The member methods of this object access the underlying Install Conduit Manager C API.

See Also `PDInstallConduitInfo` object.

PDInstallConduitInfo

Purpose Represents all the information to register an install conduit with HotSync Manager.

Methods None. Use methods defined in `PDInstallConduit` to access `PDInstallConduitInfo`.

Properties

`Directory`
(R/W) Name of the install directory associated with this install conduit.

`Extension`
(R/W) The file type extensions of the files that this install conduit can install.

`Mask`
(R/W) A unique bit mask value associated with this install conduit.

`Module`
(R/W) Filename of this install conduit.

`Name`
(R/W) User-visible name of this install conduit.

`UniqueId`
(R/W) Unique ID associated with this install conduit.

Comments The member properties of this object correspond to the conduit configuration entries created on the desktop when you register an install conduit.

See Also `PDInstallConduit` object.

PDMemoDbHHRecord

Purpose Represents a Memo Pad record. Its properties represent the values of the standard Memo Pad fields.

Methods

`ReadFromByteStream`
Private.

`WriteToByteStream`
Private.

Properties

`CategoryId`
(R/W) Category ID specified by category index.

`Index`
(R/W) Position of this record in the Memo Pad database.

`IsArchived`
(R/W) Indicates whether this record is marked to be archived.

`IsDeleted`
(R/W) Indicates whether this record is marked to be deleted.

`IsDirty`
(R/W) Indicates whether this record is has been modified since the last synchronization.

`IsPrivate`
(R/W) Indicates whether this record is marked as private.

`Memo`
(R/W) Content of a this Memo Pad record.

`UniqueId`
(R/W) The record ID of this record.

Comments For a `PDMemoDbHHRecordAdapter` object, you can create a `PDMemoDbHHRecord` object, which represents the Memo Pad record that you read or write. Each of this object's properties is one of the fields in a Memo Pad record.

Use this object with Memo Pad versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

Example See the example under "PDMemoDbHHRecordAdapter" on page 61.

See Also `PDMemoDbHHRecordAdapter` object.

PDMemoDbHHRecordAdapter

Purpose Represents an open Memo Pad record database. Its methods can iterate through records in a database serially or access them randomly.

Methods

`AddLogEntry()`
Adds a text string to the HotSync log on either the desktop or the handheld.

`ChangeCategory()`
Changes all records of a particular category to a new category.

`ReadAppInfoBlock()`
Reads a record database's AppInfo block.

`ReadById()`
Reads a record using its unique ID.

`ReadByIndex()`
Reads a record using its index.

`ReadNext()`
Reads the next record.

`ReadNextInCategory()`
Reads the next record in a category.

`ReadNextModified()`
Reads the next modified record.

`ReadNextModifiedInCategory()`
Reads the next modified record in a category.

`ReadSortInfoBlock()`
Reads a record database's SortInfo block.

`ReadUniqueIdList()`
Creates a list of unique IDs in record index order.

`Remove()`
Deletes the specified record from an open record database on the handheld.

`RemoveSet()`
Deletes a set of database records.

`ResetAllModifiedFlags()`
Resets the modified (dirty) flag of all records in the opened record database on the handheld.

`Write()`
Writes a database record.

`WriteAppInfoBlock()`
Writes an application info block to an open record database on the handheld.

`WriteSortInfoBlock()`
Writes a sort information block to an open database on the handheld.

Properties

`AccessMode`
(R) Open database access mode.

`CloseOptions`
(R/W) Update database dates on close.

`DbName`
(R) Name of this object's associated database on the handheld.

`EOF`
(R) Database iterator is at the end of the database.

`InputBufferSize`
(R) Size of the buffer to allocate to read record database data.

`IterationIndex`
(R/W) Current starting index for the record data iteration methods.

`PDCategories`
(R) Returns a `PDCategories` object representing the categories in this database.

`PDDatabaseInfo`
(R) Returns a `PDDatabaseInfo` object representing information about this database.

`RecordCount`
(R) Number of records in this database.

Comments

From a `PDDatabaseQuery` object, you can create a `PDMemoDbHHRecordAdapter` object, which represents the Memo Pad record database that you opened or created. With this object, you can access Memo Pad records represented by `PDMemoDbHHRecord` objects.

Use this object with Memo Pad versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

When you *open* a Memo Pad database with this object, you must specify both the database's name and the full adapter name of this object:

```
OpenRecordDatabase("MemoDB", _
    "PDStandard.PDMemoDbHHRecordAdapter", _
    eRead Or eWrite Or eShowSecret)
```

When you *create* a Memo Pad database with this object, you must additionally specify the Memo Pad database's creator ID (`'memo'`) and type (`'DATA'`):

```
CreateRecordDatabase("MemoDB", _
    "PDStandard.PDMemoDbHHRecordAdapter", _
    "memo", "DATA", eRead Or eWrite, eBackupDb, _
    1, 0)
```

Reading database records can be accomplished in one of two ways. Records can be accessed serially using the iterator methods, or randomly using the direct methods. The iterator methods are named `ReadNextXXX()`, and require you to set the `IterationIndex` before using them. The iterator methods are:

- `ReadNext()`
- `ReadNextInCategory()`
- `ReadNextModified()`
- `ReadNextModifiedInCategory()`

Use the `EOF` property to determine when there are no more records in the iteration.

Direct methods to access records randomly are named `ReadByXXX`, namely `ReadById()` and `ReadByIndex()`.

To close the database, in Visual Basic, set the object reference to the value `Nothing`. In C++, the last `Release` of a `PDRecordAdapter` object closes the database. If you want to update the database dates upon close, set the `CloseOptions` as you wish before closing the database.

IMPORTANT: This note applies to *Visual Studio .NET users only*. Though you can open more than one database at a time, you cannot do so with the same object. For example, if you had `PDAddressDbHHRecordAdapter1` to open database "A," you cannot use `PDAddressDbHHRecordAdapter1` again to open database "B." You must define a new `PDAddressDbHHRecordAdapter` object to open database "B."

Example

```
' Declare the PDDatabaseQuery object.
Dim pDbQuery As New PDDatabaseQuery
Dim PMemoRecord As New PDMemoDbHHRecord
   'Declare the PDMemoDbRecordAdapter object
   Dim pMemoAdapter As PDMemoDbHHRecordAdapter

' First check whether a HotSync operation is in progress.
Dim PHotSyncManager As New PDHotSyncUtility

If Not PHotSyncManager.IsSyncInProgress Then
   MsgBox "COM Sync Suite objects are only active during a _
      HotSync operation. Refer to the COMSyncCompanion.pdf _
   manual for more information on how to create a debug _
   environment.", vbInformation, "Information"
   Exit Sub
End If

' Open the MemoDB database.
Set pMemoAdapter = pDbQuery.OpenRecordDatabase("MemoDB", _
   "PDStandard.PDMemoDbHHRecordAdapter", eRead Or eWrite Or _
   eShowSecret)

' Fill in record data.
PMemoRecord.Memo = "Text of memo."

' Write the record.
Dim uniqueid As Variant
uniqueid = pMemoAdapter.Write(PMemoRecord)
   MsgBox "Successfully created a memo."
```

See Also

`PDDatabaseQuery`, `PDCategories`, `PDDatabaseInfo`, `PDMemoDbHHRecord` objects.
`CreateRecordDatabase()`, `OpenRecordDatabase()` methods.
`ERemoveSetType` constants.

PDMemoryCardInfo

Purpose Represents information about the handheld's primary storage (called a "memory card").

Methods None.

Properties

`CardName`
(R) Memory card name.

`CardNum`
(R) Card number of the memory card on the handheld on which databases are stored.

`CardVersion`
(R) Memory card version.

`CreationDate`
(R) Memory card creation date.

`FreeRamSize`
(R) Amount of available RAM on the card in bytes.

`ManufName`
(R) Memory card manufacturer's name.

`RamDbCount`
(R) Number of databases in primary storage RAM on the handheld.

`RamSize`
(R) Total amount of RAM on the memory card in bytes.

`RomDbCount`
(R) Number of databases in ROM on the handheld.

`RomSize`
(R) Total amount of ROM on the memory card in bytes.

See Also `PDMemoryCardInfo` property.

PDRecordAdapter

Purpose Represents an open **classic database**. Its methods can iterate through records in a database serially or access them randomly.

Methods

`AddLogEntry()`
Adds a text string to the HotSync log on either the desktop or the handheld.

`ChangeCategory()`
Changes all records of a particular category to a new category.

`ReadAppInfoBlock()`
Reads a record database's application info block.

`ReadById()`
Reads a record using its unique ID.

`ReadByIndex()`
Reads a record using its index.

`ReadNext()`
Reads the next record.

`ReadNextInCategory()`
Reads the next record in a category.

`ReadNextModified()`
Reads the next modified record.

`ReadNextModifiedInCategory()`
Reads the next modified record in a category.

`ReadSortInfoBlock()`
Reads a record database's sort info block.

`ReadUniqueIdList()`
Creates a list of unique IDs in record index order.

`Remove()`
Deletes the specified record from an open record database on the handheld.

`RemoveSet()`
Deletes a set of database records.

`ResetAllModifiedFlags()`
Resets the modified (dirty) flag of all records in the opened record database on the handheld.

`Write()`
Writes a database record.

`WriteAppInfoBlock()`
Writes an application info block to an open record database on the handheld.

`WriteSortInfoBlock()`
Writes a sort information block to an open database on the handheld.

Properties

`AccessMode`
(R) Open database access mode.

`CloseOptions`
(R/W) Update database dates on close.

`DbName`
(R) Name of this object's associated database on the handheld.

`EOF`
(R) Database iterator is at the end of the database.

`InputBufferSize`
(R) Size of the buffer to allocate to read record database data.

`IterationIndex`
(R/W) Current starting index for the record data iteration methods.

`PDCategories`
(R) Returns a `PDCategories` object representing the categories in this database.

`PDDatabaseInfo`
(R) Returns a `PDDatabaseInfo` object representing information about this database.

`RecordCount`
(R) Number of records in this database.

Comments

From a `PDDatabaseQuery` object, you can create a `PDRecordAdapter` object, which represents the classic record database that you opened or created.

Reading classic database records can be accomplished in one of two ways. Records can be accessed serially using the iterator methods, or randomly using the direct methods. The iterator methods are named `ReadNextXXX()`, and require you to set the `IterationIndex` before using them. The iterator methods are:

- `ReadNext()`
- `ReadNextInCategory()`
- `ReadNextModified()`
- `ReadNextModifiedInCategory()`

Use the `EOF` property to determine when there are no more records in the iteration.

Direct methods to access records randomly are named `ReadByXXX`, namely `ReadById()` and `ReadByIndex()`.

To close the database, in Visual Basic, set the object reference to the value `Nothing`. In C++, the last `Release` of a `PDRecordAdapter` object closes the database. If you want to update the database dates upon close, set the `CloseOptions` as you wish before closing the database.

IMPORTANT: This note applies to *Visual Studio .NET users only*. Though you can open more than one database at a time, you cannot do so with the same object. For example, if you had `PDRecordAdapter1` to open database “A,” you cannot use `PDRecordAdapter1` again to open database "B." You must define a new `PDRecordAdapter` object to open database “B.”

See Also

`PDDatabaseQuery`, `PDCategories`, `PDDatabaseInfo` objects.
`CreateRecordDatabase()`, `OpenRecordDatabase()` methods.
`ERemoveSetType` constants.

PDResourceAdapter

Purpose Represents an open **classic database** created to contain resources rather than records. Its methods can iterate through resources in a database serially or access them randomly.

Methods

`AddLogEntry()`
Adds a text string to the HotSync® log on either the desktop or the handheld.

`ReadNextResource()`
Reads the next record in a resource database.

`ReadResource()`
Reads a resource record by index.

`RemoveAllResources()`
Deletes all resources from an open resource database on the handheld.

`RemoveResource()`
Deletes a resource from an open resource database on the handheld.

`WriteResource()`
Writes a resource to an open resource database on the handheld.

Properties

`AccessMode`
(R) Open database access mode.

`CloseOptions`
(R/W) Update database dates on close.

`DbName`
(R) Name of this object's associated database on the handheld.

`EOF`
(R) Database iterator is at the end of the database.

`InputBufferSize`
(R) Size of the buffer to allocate to read record database data.

`IterationIndex`
(R/W) Current starting index for the record data iteration methods.

`PDDatabaseInfo`
(R) Returns a `PDDatabaseInfo` object representing information about this database.

`RecordCount`
(R) Number of records in this database.

Comments From a `PDDatabaseQuery` object, you can create a `PDResourceAdapter` object, which represents the resource database that you opened or created.

Reading database records can be accomplished in one of two ways. Records can be accessed serially using the iterator methods, or randomly using the direct methods. The iterator methods are named `ReadNextXXX()`, and require you to set the `IterationIndex` before using them. The iterator function is named `ReadNextResource()`. Use the `EOF` property to determine when there are no more records in the iteration.

The direct method to access records randomly is named `ReadResource()`.

To close the database, in Visual Basic, set the object reference to the value `Nothing`. In C++, the last `Release` of a `PDResourceAdapter` object closes the database. If you want to update the database dates upon close, set the `CloseOptions` as you wish before closing the database.

IMPORTANT: This note applies to *Visual Studio .NET users only*. Though you can open more than one database at a time, you cannot do so with the same object. For example, if you had `PDResourceAdapter1` to open database “A,” you cannot use `PDResourceAdapter1` again to open database "B." You must define a new `PDResourceAdapter` object to open database “B.”

See Also `PDDatabaseInfo` object.
`CreateResourceDatabase()`, `OpenResourceDatabase()` methods.

PDSystemAdapter

Purpose Handheld system functions.

Methods

`AddLogEntry()`
Adds a text string to the HotSync® log on either the desktop or the handheld.

`CallRemoteModule()`
Calls a module (an application, panel, or other executable) on the handheld and returns data and status information to your conduit from that module.

`HHOsVersion()`
Returns the Palm OS® software version.

`ReadAppPreference()`
Reads an application's preference block.

`ReadFeature()`
Returns a feature.

`RebootSystem()`
Sends a request to soft-reset the handheld at the end of the HotSync operation.

`SyncMgrAPIVersion()`
Retrieves the version of the Sync Manager API that is installed on the desktop computer.

`WriteAppPreference()`
Writes an application's preference block.

Properties

`DateTime`
(R/W) Current date and time on the handheld.

`LocalizationId`
(R) Localization ID, currently unused.

`PDHotsyncInfo`
(R) Returns a `PDHotsyncInfo` object representing information about the current HotSync session.

`PDMemoryCardInfo`
(R) Returns a `PDMemoryCardInfo` object representing information about the handheld's primary storage (called a "memory card").

`PDUserInfo`
(R) Returns a `PDUserInfo` object representing information about the current handheld user.

`ProductId`
(R) Handheld product ID.

`RomSoftwareVersion`
(R) Palm OS® software version on the handheld.

See Also `PDHotsyncInfo`, `PDMemoryCardInfo`, `PDUserInfo`, `PDRecordAdapter` objects.

PDSystemCondMgr

Purpose A collection of utility methods that register conduits with HotSync Manager for the system. These methods also manage information about these conduits.

Methods

`CreatorIDToString()`
: Converts a `Long` conduit creator ID into a `String`.

`GetBackupConduit()`
: Retrieves the name of HotSync Manager's **backup conduit** for the system.

`GetConduitCount()`
: Returns the number of conduits registered with HotSync Manager for the system.

`GetConduitInfo()`
: Returns complete information about a system-registered conduit in a `PDConduitInfo` object.

`GetConduitList()`
: Returns a list of creator IDs of all the system-registered conduits.

`GetDWORDData()`
: Retrieves a `DWORD` configuration entry value for the specified system-registered conduit.

`GetStringData()`
: Retrieves a `String` configuration entry value for the specified system-registered conduit.

`RegisterConduit()`
: Registers a conduit for the system based on the information provided in a `PDConduitInfo` object.

`SetBackupConduit()`
: Sets the filename of the HotSync Manager backup conduit for the system.

`SetDWORDData()`
: Sets a `DWORD` configuration entry value for the specified system-registered conduit.

`SetStringData()`
: Sets a `String` configuration entry value for the specified system-registered conduit.

`StringToCreatorID()`
: Converts a `String` into a `DWORD` conduit creator ID.

`UnregisterConduit()`
: Unregisters a system-registered conduit with HotSync Manager.

Properties None.

Comments The member methods of this object access the underlying Conduit Manager C API.

This object manages conduits for the system. To manage user-registered conduits and notifiers, use `PDCondMgr`.

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

See Also `PDConduitInfo`, `PDCondMgr` objects.
"Registering Conduits and Notifiers with HotSync Manager" on page 73 in *Introduction to Conduit Development*.

PDTodoDbHHRecord

Purpose Represents a To Do List record. Its properties represent the values of the standard To Do List fields.

Methods

`ReadFromByteStream`
Private.

`WriteToByteStream`
Private.

Properties

`CategoryId`
(R/W) Category ID specified by category index.

`Description`
(R/W) Text describing this To Do List item.

`DueDate`
(R/W) Due date of this item.

`Index`
(R/W) Position of this record in the To Do List database.

`IsArchived`
(R/W) Indicates whether this record is marked to be archived.

`IsCompleted`
(R/W)

`IsDeleted`
(R/W) Indicates whether this record is marked to be deleted.

`IsDirty`
(R/W) Indicates whether this record is has been modified since the last synchronization.

`IsPrivate`
(R/W) Indicates whether this record is marked as private.

`Notes`
(R/W) Content of the note in this record.

`Priority`
(R/W) Priority of this To Do List item.

`UniqueId`
(R/W) The record ID of this record.

Comments For a `PDTodoDbHHRecordAdapter` object, you can create a `PDTodoDbHHRecord` object, which represents the To Do List record that you read or write. Each of this object's properties is one of the fields in a To Do List record.

Use this object with To Do List versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

Example See the example under "PDTodoDbHHRecordAdapter" on page 77.

See Also `PDTodoDbHHRecordAdapter` object.

PDTodoDbHHRecordAdapter

Purpose Represents an open To Do List record database. Its methods can iterate through records in a database serially or access them randomly.

Methods

`AddLogEntry()`
Adds a text string to the HotSync log on either the desktop or the handheld.

`ChangeCategory()`
Changes all records of a particular category to a new category.

`ReadAppInfoBlock()`
Reads a record database's application info block.

`ReadById()`
Reads a record using its unique ID.

`ReadByIndex()`
Reads a record using its index.

`ReadNext()`
Reads the next record.

`ReadNextInCategory()`
Reads the next record in a category.

`ReadNextModified()`
Reads the next modified record.

`ReadNextModifiedInCategory()`
Reads the next modified record in a category.

`ReadSortInfoBlock()`
Reads a record database's sort info block.

`ReadUniqueIdList()`
Creates a list of unique IDs in record index order.

`Remove()`
Deletes the specified record from an open record database on the handheld.

`RemoveSet()`
Deletes a set of database records.

`ResetAllModifiedFlags()`
Resets the modified (dirty) flag of all records in the opened record database on the handheld.

`Write()`
Writes a database record.

`WriteAppInfoBlock()`
Writes an application info block to an open record database on the handheld.

`WriteSortInfoBlock()`
Writes a sort information block to an open database on the handheld.

Properties

`AccessMode`
(R) Open database access mode.

`CloseOptions`
(R/W) Update database dates on close.

`DbName`
(R) Name of this object's associated database on the handheld.

`EOF`
(R) Database iterator is at the end of the database.

`InputBufferSize`
(R) Size of the buffer to allocate to read record database data.

`IterationIndex`
(R/W) Current starting index for the record data iteration methods.

`PDCategories`
(R) Returns a `PDCategories` object representing the categories in this database.

`PDDatabaseInfo`
(R) Returns a `PDDatabaseInfo` object representing information about this database.

`RecordCount`
(R) Number of records in this database.

Comments From a PDDatabaseQuery object, you can create a `PDTodoDbHHRecordAdapter` object, which represents the To Do List record database that you opened or created. With this object, you can access To Do List records represented by PDTodoDbHHRecord objects.

Use this object with To Do List versions earlier than 6.0. It does not work with the the application provided in Palm OS Cobalt.

When you *open* a To Do List database with this object, you must specify both the database's name and the full adapter name of this object:

```
OpenRecordDatabase("ToDoDB", _
   "PDStandard.PDTodoDbHHRecordAdapter", _
   eRead Or eWrite Or eShowSecret)
```

When you *create* a To Do List database with this object, you must additionally specify the To Do List database's creator ID (`'todo'`) and type (`'DATA'`):

```
CreateRecordDatabase("ToDoDB", _
   "PDStandard.PDTodoDbHHRecordAdapter", _
   "todo", "DATA", eRead Or eWrite, eBackupDb, _
   1, 0)
```

Reading database records can be accomplished in one of two ways. Records can be accessed serially using the iterator methods, or randomly using the direct methods. The iterator methods are named `ReadNextXXX()`, and require you to set the `IterationIndex` before using them. The iterator methods are:

- `ReadNext()`
- `ReadNextInCategory()`
- `ReadNextModified()`
- `ReadNextModifiedInCategory()`

Use the `EOF` property to determine when there are no more records in the iteration.

Direct methods to access records randomly are named `ReadByXXX`, namely `ReadById()` and `ReadByIndex()`.

To close the database, in Visual Basic, set the object reference to the value `Nothing`. In C++, the last `Release` of a `PDRecordAdapter` object closes the database. If you want to update the database dates upon close, set the `CloseOptions` as you wish before closing the database.

IMPORTANT: This note applies to *Visual Studio .NET users only*. Though you can open more than one database at a time, you cannot do so with the same object. For example, if you had `PDTodoDbHHRecordAdapter1` to open database “A,” you cannot use `PDTodoDbHHRecordAdapter1` again to open database "B." You must define a new `PDTodoDbHHRecordAdapter` object to open database “B.”

Example

```
' Declare the PDDatabaseQuery object.
Dim pDbQuery As New PDDatabaseQuery
Dim PTodoRecord As New PDTodoDbHHRecord
   'Declare the PDTodoDbRecordAdapter object
   Dim pTodoAdapter As PDTodoDbHHRecordAdapter

' First check whether a HotSync operation is in progress.
Dim PHotSyncManager As New PDHotSyncUtility

If Not PHotSyncManager.IsSyncInProgress Then
   MsgBox "COM Sync Suite objects are only active during a _
      HotSync operation. Refer to the COMSyncCompanion.pdf _
   manual for more information on how to create a debug _
   environment.", vbInformation, "Information"
   Exit Sub
End If

' Open the ToDoDB database.
Set pTodoAdapter = pDbQuery.OpenRecordDatabase("ToDoDB", _
   "PDStandard.PDAddressDbHHRecordAdapter", eRead Or eWrite _
   Or eShowSecret)

' Fill in record data.
PTodoRecord.Description = "Buy a Palm Powered™ handheld."
PTodoRecord.DueDate = Date
PTodoRecord.Priority = 1
PTodoRecord.Notes = “This has been a subliminal message _
   brought to you by PalmSource, Inc.”
```

```
' Write the record.
Dim uniqueid As Variant
uniqueid = pTodoAdapter.Write(PTodoRecord)
   MsgBox "Successfully created a To Do List item."
```

See Also `PDDatabaseQuery`, `PDCategories`, `PDDatabaseInfo`, `PDTodoDbHHRecord` objects.
`CreateRecordDatabase()`, `OpenRecordDatabase()` methods.
`ERemoveSetType` constants.

PDUserData

Purpose A collection of utility methods for accessing the **users data store** on the desktop computer.

Methods

`AddNewUser()`
Adds a user to the users data store.

`DeleteKey()`
Deletes a key or an entire section from the specified user's area of the users data store.

`DeleteUser()`
Deletes a user from the users data store.

`DeleteUserPermanentSyncPreferences()`
Deletes the permanent synchronization preferences for *all* of the specified user's conduits.

`DeleteUserTemporarySyncPreferences()`
Deletes the temporary synchronization preferences for *all* of the specified user's conduits.

`GetIDFromName()`
Retrieves a unique **user ID** given the user's name.

`GetIDFromPath()`
Retrieves a user ID given the user directory.

`GetIntegerValue()`
Retrieves an integer value from a key in the specified user's area of the users data store.

`GetRootDirectory()`
Retrieves the path of all user directories on the desktop computer (as stored in the `Core\Path` HotSync Manager configuration entry).

`GetSlotCount()`
Retrieves the number of expansion slots on the handheld for the specified user.

`GetSlotDisplayName()`
Retrieves the display name for the given slot on the specified user's handheld.

`GetSlotInstallDirectory()`
Retrieves the slot-install directory name (not the full path) for the specified user and handheld slot.

`GetSlotList()`
: Retrieves a list of all the slot IDs for each of the expansion slots present on the specified user's handheld.

`GetSlotMediaType()`
: Retrieves the media type of the given slot on the specified user's handheld.

`GetStringValue()`
: Retrieves a string value from a key in the specified user's area of the users data store.

`GetUserCount()`
: Returns the number of users in the users data store.

`GetUserDirectory()`
: Retrieves the user directory's name for the specified user ID.

`GetUserList()`
: Retrieves a list of user IDs.

`GetUserNameFromID()`
: Retrieves a user name in the users data store given a user ID.

`GetUserPassword()`
: Retrieves the encrypted user password for the specified user ID.

`GetUserPermanentSyncPreferences()`
: Retrieves a conduit's permanent synchronization preferences for the specified user ID.

`GetUserTemporarySyncPreferences()`
: Retrieves a conduit's temporary synchronization preferences for the specified user ID.

`IsProfileUser()`
: Determines whether an account is a **user profile**.

`RemoveUserTemporarySyncPreferences()`
: Removes the specified conduit's temporary synchronization preferences for the specified user ID.

`SetIntegerValue()`
: Sets an integer value to a key in the specified user's area of the users data store.

`SetStringValue()`
: Sets a string value to a key in the specified user's area of the users data store.

`SetUserDirectory()`
: Sets the directory name of the specified user ID.

`SetUserName()`
: Sets the user name of the specified user ID.

`SetUserPermanentSyncPreferences()`
: Sets a conduit's permanent synchronization preferences for the specified user ID.

`SetUserTemporarySyncPreferences()`
: Sets a conduit's temporary synchronization preferences for the specified user ID.

Properties None.

Comments The member methods of this object access the underlying User Data C API.

PDUserInfo

Purpose Current handheld user information.

Methods None.

Properties `UserId`

(R) User ID, which specifies the user to reference in the users data file.

`LastSyncDate`

(R) Last synchronization date.

`LastSyncPC`

(R) ID assigned by HotSync Manager of the last PC that was synchronized with this handheld.

`Password`

(R) Encrypted handheld password.

`UserName`

(R) Name of the handheld user in the user data store to synchronize with.

`ViewerId`

(R) ID of the handheld. Not currently used.

See Also `PDUserInfo` property.

PDUtility

Purpose A collection of utility methods for manipulating strings, byte arrays, and integers.

Methods

`AllBytesToBSTR()`
Creates a `String` from all the bytes in a string `Byte` array.

`BSTRToByteArray()`
Inserts a `String` into a `Byte` array.

`BSTRToDWORD()`
Converts a four-character string to an unsigned `Long`—for example, to convert creator IDs and database types.

`ByteArrayToBSTR()`
Extracts a `String` from a `Byte` array.

`ByteArrayToDWORD()`
Extracts an unsigned `Long` from a `Byte` array (for example, a record ID).

`ByteArrayToHexBSTR()`
Converts an input `Byte` array to a formatted hex display `String`.

`ByteArrayToRecordId()`
Converts a `Byte` array to a record ID.

`ByteArrayToWORD()`
Extracts an unsigned `Integer` from a `Byte` array.

`DWORDToBSTR()`
Converts an unsigned `Long` to a four-character `String`. Used for creator IDs, database type, and others.

`DWORDToByteArray()`
Inserts an unsigned `Long` into a `Byte` array.

`RecordIdToByteArray()`
Converts a record ID to a `Byte` array.

`RecordIdToString()`
Converts record ID to a readable `String`.

`StringToRecordId()`
Converts a string (`BSTR`) to record ID.

`SwapDWORD()`
Swaps the bytes of an unsigned `Long`.

`SwapWORD()`
Swaps the bytes of an unsigned `Integer`.

`WORDToByteArray()`
Inserts an unsigned `Integer` into a `Byte` array.

Properties None.

PDVFSFileManager

Purpose Represents a file or directory created or opened by a `PDVFSVolumeManager` object. Its methods can read and write open files or directories.

Methods

`Close()`
Closes this open file on an expansion card.

`Read()`
Reads data from a file on an expansion card into the specified buffer.

`Seek()`
Sets the position from which to read or write within an open file on an expansion card.

`Tell()`
Gets the current position of the file pointer within an open file on an expansion card.

`Write()`
Writes data to an open file on an expansion card.

Properties

`Attributes`
Attributes of a file or directory on an expansion card, such as whether it is read-only.

`CreationDate`
Creation date for a file or directory.

`EOF`
The file pointer has reached the end of the file.

`LastAccessedDate`
Last accessed date of a file or a directory on an expansion card.

`LastModificationDate`
Last modification date of a file or a directory on an expansion card.

`Size`
Size of a file on an expansion card or what to resize a file to.

Comments Call the `PDVFSVolumeManager.Open()` method to create a `PDVFSFileManager` object.

See Also `PDVFSVolumeManager` object.
`Open()`, `GetFileList()`, `GetSubDirectoryList()`, `ImportDatabaseFromFile()` methods.
`EPDFileOrigin` constants.

PDVFSManager

Purpose Represents the Virtual File System (VFS) Manager on the desktop. Its methods can detect the presence of and get information about file system volumes available on a given expansion card in a handheld. It can get volume reference numbers for all mounted volumes, which you subsequently use to create a `PDVFSVolumeManager` object.

Methods

`GetVolumeCount()`
Retrieves the total number of mounted volumes on cards in all expansion slots.

`GetVolumeManager()`
Creates a `PDVFSVolumeManager` object to access a given volume.

`GetVolumeReferenceList()`
Retrieves a list of the volume reference numbers of all mounted volumes.

`IsVolumeAvailable()`
Determines whether there is a volume available on the handheld.

Properties None.

Comments The VFS Manager is a layer of software that allows conduits to access all installed file systems on handheld expansion cards. It provides a unified interface to conduit developers while allowing them to seamlessly access many different types of file systems—such as VFAT, HFS, and NFS—on many different types of media, including Secure Digital (SD), MultiMediaCard (MMC), CompactFlash, Sony's Memory Stick, and others.

See Also `PDVFSVolumeManager` object.
`SlotReferenceNumber` property.

PDVFSVolumeManager

Purpose Represents a volume on an expansion card. Its methods can create files and directories, copy files to and from the desktop, and import/export files and Palm OS® databases between primary storage memory and an expansion card.

Methods

`CopyFileFromDeskTop()`
Copies a file from the desktop to a volume on a handheld expansion card.

`CopyFileToDeskTop()`
Copies a file from a volume on a handheld expansion card to the desktop.

`CreateDirectory()`
Creates a directory on this volume on a handheld expansion card.

`CreateFile()`
Creates a file on this volume on a handheld expansion card.

`Delete()`
Deletes a closed file or empty directory on this volume on a handheld expansion card.

`ExportDatabaseToFile()`
Flattens and exports the specified database on the handheld to the specified PDB or PRC file on an expansion card. Works only with classic databases.

`Format()`
Formats and mounts this volume.

`GetDefaultDirectory()`
Retrieves the default directory on this volume on an expansion card for files of the specified type.

`GetFileList()`
Retrieves the names of all the files in a given directory.

`GetSubDirectoryList()`
Retrieves the names of all the subdirectories in a given directory.

`ImportDatabaseFromFile()`
Creates a database from the specified PDB or PRC file on an expansion card. Works only with classic databases.

`Open()`
: Opens a file or directory on an expansion card and returns a `PDVFSFileManager` object.

`Rename()`
: Renames a closed file or directory on an expansion card.

Properties

`Attributes`
: Attributes of a volume on an expansion card, such as whether it is read-only.

`FileSystemType`
: Type of file system on this volume on an expansion card.

`Label`
: Label of this volume on an expansion card.

`MediaType`
: Type of media supported by the expansion card.

`mountClass`
: Mount class of the file system driver that mounted this volume on an expansion card.

`SlotLibRefNumber`
: Reference number for the slot driver shared library on the handheld that is allocated to the slot number on which this volume is mounted.

`SlotReferenceNumber`
: Reference number for the expansion slot that holds this volume.

`TotalCapacity`
: Total capacity, in bytes, of this volume on an expansion card.

`UsedSpace`
: Amount of space, in bytes, already in use on this volume on an expansion card.

Comments Call `PDVFSManager`.`GetVolumeManager()` to create this object. Then use `PDVFSVolumeManager` methods to access a particular volume.

See Also `PDVFSManager` object.
`GetVolumeManager()`, `Close()`, `Read()`, `Write()`, `Tell()`, `Seek()` methods.
VFS Volume Mount Class Constants, `EPDVFSFileSystemType` constants.

PSDCategoryAdapter

Purpose Represents the collection of categories in a schema database.

Methods `AddCategory()`

Adds a new category to a schema database and returns its category ID.

`GetCount()`

Retrieves the number of categories in a schema database.

`GetIDList()`

Retrieves a list of the category IDs in a schema database.

`GetModifiedIDList()`

Retrieves a list of the IDs of modified categories in a schema database.

`GetNameList()`

Retrieves the names of all of the categories in a schema database.

`IsDirty()`

Determines whether a category has been modified since the last HotSync operation.

`RemoveCategory()`

Removes a category from a schema database.

`RenameCategory()`

Changes the name of a category in a schema database.

Properties `Name`

(R) Category name specified by category ID in a schema database.

PSDColumnInfo

Purpose Represents information about a column in a schema database.

Methods None.

Properties

`DataType`
(R/W) Type of data stored in a column in a schema database.

`Dynamic`
(R/W) Flag that indicates whether a column in a schema is dynamic.

`ID`
(R/W) Column ID of a column in a schema.

`MaxSize`
(R/W) Maximum size of a column in a schema.

`Name`
(R/W) Name of a column in a schema.

`NonSyncable`
(R/W) Flag that indicates whether the data in a column is to be synchronized.

`WritableExceptionInReadOnlyRows`
(R/W) Flag that indicates whether the data in a column is writable.

PSDDatabaseAdapter

Purpose Represents an open **schema database**. Its methods manage rows, groups of rows with the same schema (tables), columns, and each row's category membership.

Methods

`AddTable()`
Adds a new table to a schema database.

`DeleteRowsInCategory()`
Deletes rows whose category IDs match those on the specified list according to the specified match mode.

`GetCategoryAdapter()`
Returns a category adapter object for a schema database.

`GetColumnCustomProperty()`
Retrieves the value of a custom column property in a table.

`GetDatabaseInfo()`
Retrieves information about this schema database.

`GetDatabaseHandle()`
Returns the handle of this open schema database.

`GetModifiedTableNames()`
Retrieves the names of tables that have been modified since the last HotSync operation.

`GetRowAdapter()`
Returns a row adapter object for this schema database.

`GetSyncTypeInfo()`
Retrieves the synchronization mode of a sync atom for this schema database in the current HotSync operation.

`GetTableCount()`
Returns the total number of tables in this schema database.

`GetTableInfo()`
Returns information about a table in this schema database.

`GetTableNames()`
Retrieves the names of all of the tables in this schema database.

`MoveRowsToCategory()`
Moves all of the rows that belong to a specified set of categories into another category.

`RemoveCategoryFromAllRows()`
: Removes all matching rows from a specified list of categories in this schema database.

`RemoveColumnCustomProperty()`
: Removes a custom property from a table column in this schema database.

`RemoveTable()`
: Removes a table from this schema database.

`SetColumnCustomProperty()`
: Sets the value of a custom column property in a table.

Properties None.

PSDDatabaseInfo

Purpose Represents information about a schema database.

Methods None.

Properties

`Attributes`
(R) Flags that indicate the attributes of this schema database.

`BackupDate`
(R) Date that this database was last backed up.

`CreationDate`
(R) Date on which this schema database was created.

`CreatorID`
(R) Creator ID of this schema database.

`DataBytes`
(R) Number of bytes of storage used by this schema database for data only, excluding overhead.

`DisplayName`
(R) Display name of this schema database.

`Encoding`
(R) Type of character encoding of text data in a schema database only.

`Flags`
(R) Flags that indicate whether this schema database is excluded from HotSync operations and whether it is in RAM on the handheld.

`IsReadOnlyDatabase`
(R) Flag that indicates whether this schema database is read-only.

`IsSecureDatabase`
(R) Flag that indicates whether this schema database is secure.

`ModifyDate`
(R) Date on which this schema database was most recently modified.

`ModifyNumber`
: (R) The database modification number, which is incremented every time a row in this schema database is added, modified, or deleted on the handheld.

`Name`
: (R) Internal name of this schema database.

`RowCount`
: (R) Number of rows in this schema database.

`TableCount`
: (R) Number of tables in this schema database.

`TotalBytes`
: (R) Total number of bytes of storage used by this database, including overhead.

`Type`
: (R) Database type of this schema database.

`Version`
: (R) An application-specific version number of this database.

See Also `BackupDatabase()`, `GetDatabaseInfo()`, `InstallAndBackupDatabase()`, `IsDatabaseBackupNeeded()`, `ReadDatabaseInfoByName()`, `ReadBackupImageInfo()` methods.

PSDDatabaseQuery

Purpose Represents the collection of databases on the handheld. Some of this objects's methods are specific to schema databases and are indicated as such below.

Methods

`AddLogEntry()`
Adds a text string to the HotSync log on either the desktop or the handheld.

`BackupSecurityData()`
Backs up vault databases from the handheld to a directory on the desktop.

`CloseDatabase()`
(Schema databases only) Closes an open schema database.

`CreateDatabase()`
(Schema databases only) Creates a schema database.

`DeleteDatabase()`
(Schema databases only) Deletes a schema database and all of its data.

`GenerateBackupFileName()`
Generates the unique backup filename of a database specified by its name, creator, type, and attributes.

`GetChangeContext()`
Retrieves the **change context** for a schema database from the handheld.

`GetDeskTopTrustStatus()`
Determines whether the HotSync operation in progress is with a trusted desktop.

`InstallDatabase()`
Installs a database image file on the desktop to primary storage on a handheld.

`OpenDatabase()`
(Schema databases only) Opens a schema database.

`ReadBackupImageInfo()`
Reads the database header information from a backup image file on the desktop.

`ReadDatabaseInfoByName()`
Retrieves information about a database given its name, creator ID, and type.

`ReadDatabaseNameList()`
Returns the names of all databases on the handheld that match the specified creator ID and type.

`RestoreSecurityData()`
Restores **vault** databases from the desktop to the handheld.

Properties None.

Comments This object provides some methods that operate on all all types of database: classic, extended, and schema databases. However, its methods to create/open/close/delete operate only on schema databases, as indicated above. To perform these actions on classic databases, use `PDDatabaseQuery`; on extended databases, use `DmDatabaseQuery`.

This is the first object that you need to create if you want to use these methods to access databases on the handheld. You can open/create as many schema databases on the handheld as you need. But because the underlying Sync Manager API permits only one open database at a time, performance degrades if you intermix record access from more than one database.

PSDDatabaseUtilities

Purpose A collection of utility methods for backing up and installing databases during a HotSync operation, determing desktop trust status, and managing security data.

Methods

`BackupDatabase()`
Backs up a handheld database to a directory or file on the desktop.

`BackupSecurityData()`
Backs up vault databases from the handheld to a directory on the desktop.

`CallDeviceApplication()`
Calls an application on a Palm OS Cobalt handheld and returns data and status information to your conduit from that application.

`GenerateBackupFileName()`
Generates the unique backup filename of a database specified by its name, creator, type, and attributes.

`GetDeskTopTrustStatus()`
Determines whether the HotSync operation in progress is with a trusted desktop.

`InstallAndBackupDatabase()`
Installs a database on the handheld from an image file on the desktop and then backs up the same database.

`InstallDatabase()`
Installs a database image file on the desktop to primary storage on a handheld.

`IsDatabaseBackupNeeded()`
Determines whether the desktop backup file for a database on the handheld is out-of-date.

`ReadBackupImageInfo()`
Reads the database header information from a backup image file on the desktop.

`RestoreSecurityData()`
Restores vault databases from the desktop to the handheld.

Properties None.

Comments

This object provides some methods that operate on all all types of database: classic, extended, and schema databases. However, its methods to create/open/close/delete operate only on schema databases, as indicated above. To perform these actions on classic databases, use `PDDatabaseQuery`; on extended databases, use `DmDatabaseQuery`.

This is the first object that you need to create if you want to use these methods to access databases on the handheld. You can open/ create as many schema databases on the handheld as you need. But because the underlying Sync Manager API permits only one open database at a time, performance degrades if you intermix record access from more than one database.

PSDRowAdapter

Purpose Represents an open **schema database**. Its methods can manipulate rows in a table.

Methods

`AddCategoryMembership()`
Adds a row to all of the categories in the specified list in a schema database.

`AddRow()`
Adds a new row to a table in a schema database.

`DeleteAllRowsInTable()`
Marks all rows as deleted in a table in a schema database.

`DeleteRow()`
Marks a row as deleted in a schema database.

`GetCategoryMembership()`
Retrieves a row's category memberships in a schema database.

`GetRowCountInTable()`
Retrieves the number of rows in a table in a schema database.

`IsRowInCategory()`
Determines whether a row belongs to a set of categories.

`ModifyRow()`
Writes an entire row—attributes, category memberships, and column values—to a schema database on the handheld.

`PurgeAllRowsInTable()`
Removes all the rows that are marked as deleted in a table in a schema database.

`ReadColumnValue()`
Reads the specified bytes of a column value from a row in a schema database.

`ReadColumnValues()`
Reads the specified column values from a row in a schema database.

`ReadIDList()`
Retrieves the row IDs of all the rows in a table that are in a set of categories.

`ReadModifiedIDList()`
: Retrieves the row IDs of all the modified rows in a table that are in a set of categories.

`ReadModifiedRows()`
: Reads the modified rows in a table that match the specified criteria.

`ReadRow()`
: Reads an entire row—attributes, category memberships, and column values—from a schema database on the handheld.

`ReadRowInfo()`
: Retrieves information about a row, but no column values, from a schema database on the handheld.

`ReadRows()`
: Reads entire rows that match the given criteria from a schema database on the handheld.

`ReadRowsByIDList()`
: Reads entire rows that are on the specified row ID list from a schema database on the handheld.

`RemoveAllSecretRowsInTable()`
: Removes all of the secret rows in a table in a schema database.

`RemoveCategoryMembership()`
: Removes a row from all of the categories on a list.

`RemoveRow()`
: Removes a row from a schema database.

`SetCategoryMembership()`
: Adds a row to all the categories on a list.

`WriteColumnValue()`
: Writes the specified bytes of a single column value to a row in a schema database.

`WriteColumnValues()`
: Writes a set of column values to a row in a schema database.

Properties None.

Comments To create a `PSDRowAdapter` object, call `PSDDatabaseAdapter`.`GetRowAdapter()`.

PSDRowData

Purpose Represents the data in a table row in a schema database.

Methods

`AttachToTable()`
Attaches this object to a table's schema.

`GetCategoryCount()`
Retrieves the number of categories to which this row belongs.

`GetColumnsWithData()`
Retrieves a list of names of the columns in this row that contain data.

`GetDataSize()`
Retrieves the size of a column value in this row.

`GetDataType()`
Retrieves the data type of a column in this row.

`IsArchived()`
Determines whether this row is marked for archiving.

`IsDataModified()`
Determines whether this row contains column data that is marked as modified since the last HotSync operation.

`IsDeleted()`
Determines whether this row has been marked as deleted.

`IsMembershipModified()`
Determines whether this row's category memberships have been modified.

Properties

`CategoryIDList`
(R) List of categories to which this row belongs.

`ColumnIDFromName`
(R) Column ID specified by column name.

`ColumnNameFromID`
(R) Column name specified by column ID.

`IsDataPresent`
(R) Flag that indicates whether a column in this row contains valid data.

`IsPrivate`
(R/W) Flag that indicates whether this row is marked private.

`IsReadOnly`
: (R/W) Flag that indicates whether this row is marked read-only.

`RowID`
: (R) The unique row ID of this row.

`TableName`
: (R) The name of the table that this row is in.

`Value`
: (R/W) The value of a column in this row that is specified by column name.

`ValueByID`
: (R/W) The value of a column in this row that is specified by column ID.

Comments All `PSDRowData` methods use column names rather than column IDs, except where necessary.

PSDRowSet

Purpose Represents a set of schema database rows that is returned by `PSDRowAdapter` methods that can read more than one row.

Methods

`GetCurrentRowID()`
Retrieves the current row ID in this set of rows.

`GetRowCount()`
Retrieves the number of rows in this row set.

`MoveFirst()`
Moves the cursor to the first row in this set and returns an object representing the first row.

`MoveLast()`
Moves the cursor to the last row in this set and returns an object representing the last row.

`MoveNext()`
Moves the cursor to the next row in this set and returns an object representing this row.

`MovePrevious()`
Moves the cursor to the previous row in this set and returns an object representing this row.

`MoveTo()`
Moves the cursor to the specified row in this set and returns an object representing this row.

Properties

`BOF`
(R/W) The cursor has reached the beginning of this row set.

`EOF`
(R) The cursor is at the end of the row set.

Comments The methods of this object allow you to iterate through all the rows in the `PSDRowSet` object returned by the `PSDRowAdapter` methods listed in “See Also” below.

Example See the `RowSet` sample provided in the COM Sync Suite of the CDK.

See Also `PSDRowAdapter` object.
`ReadRows()`, `ReadModifiedRows()`, `ReadRowsByIDList()`, `ReadModifiedIDList()` methods.

PSDTable

Purpose Represents the schema of a table in a schema database.

Methods

`AddColumn()`
Adds a column definition to the schema of this table.

`GetColumnCount()`
Retrieves the number of columns in this table.

`GetColumnIDList()`
Retrieves the column IDs of all columns in this table.

`GetColumnInfoByID()`
Retrieves a column definition from this table given a column ID.

`GetColumnInfoByName()`
Retrieves a column definition from this table given a column name.

`GetColumnNames()`
Retrieves a list of all the column names in this table.

`RemoveColumns()`
Removes column definitions from this table given a list of column IDs.

Properties

`Name`
(R/W) Name of this table in a schema database.

4 Methods

This chapter describes the COM Sync methods in alphabetical order.

AddCategory

Purpose Adds a new category to a schema database and returns its category ID.

Applies to `PSDCategoryAdapter` object.

Prototype
```
Function AddCategory(ByVal CategoryName As String)
   As Long
```

Parameters → *`CategoryName`*
Text string to add to the desktop or handheld HotSync log.

Returns The category ID of the newly created category.

AddCategoryMembership

Purpose Adds a row to all of the categories in the specified list in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Sub AddCategoryMembership(ByVal vRowID,
    CategoryIDList)
```

Parameters → *vRowID*

The row ID of the row to add to the specified categories.

→ *CategoryIDList*

An array of category IDs. This method adds the row to all of these categories.

Returns None.

Comments

AddColumn

Purpose Adds a column definition to the schema of this table.

Applies to `PSDTable` object.

Prototype `Sub AddColumn(PSDColumnInfo As IPSDColumnInfo)`

Parameters → *`PSDColumnInfo`*
A `PSDColumnInfo` object that defines a column to add.

Returns None.

AddLogEntry

Purpose Adds a text string to the HotSync® log on either the desktop or the handheld.

Applies to `DmDatabaseQuery`, `DmRecordAdapter`, `PDSystemAdapter`, `PDDatabaseQuery`, `PDResourceAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype

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

→ *`pLogText`*
Text string to add to the desktop or handheld HotSync log.

→ *`eActivity`*
Log activity constant, see `ELogActivity` constants.

→ *`bTimeStamp`*
Boolean that selects whether a timestamp is added to the desktop log entry. Applies only to the desktop log. When `True`, add a time stamp. When `False`, don’t add a time stamp.

→ *`bPalmLog`*
Boolean that selects whether to add to the handheld or desktop HotSync log. When `True`, add comments to the handheld log. When `False`, add comments to the desktop log.

Returns None.

Example

```
Dim pSystem as New PDSystemAdapter
Dim pLogText as String
' Add to the desktop log
pLogText = "Add this string to the desktop log."
pSystem.AddLogEntry(pLogText, eText)
' Add to the HotSync log
pLogText = "Add this string to the HotSync log."
pSystem.AddLogEntry(pLogText, , , True)
```

See Also `ELogActivity` constants.

AddNewUser

Purpose Adds a user to the users data store.

Applies to `PDUserData` object.

Prototype

```
Sub AddNewUser(UserName As String, bIsProfileUser
    As Boolean)
```

Parameters

→ *`UserName`*
A string containing the name of the user to add. It must be no more than 20 characters long.

→ *`bIsProfileUser`*
If `True`, this method creates a new user profile named `UserName`. If `False`, it creates a regular user account.

Returns None.

Errors

`eInvalidUser`
`UserName` is not an acceptable string.

`eNoCorePath`
No path to find the users data store was found.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eSaveErr`
Saving changes was not successfully completed.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

`eUserExists`
The user name has already been created.

Comments If the users data store does not exist, this method creates it and adds the new user to it.

See Also `DeleteUser()`, `GetIDFromName()` methods

AddRow

Purpose Adds a new row to a table in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Function AddRow(PSDRowData As IPSDRowData) As
    Variant
```

Parameters → *`PSDRowData`*

A `PSDRowData` object that specifies the new row's data.

Returns The row ID of this new row.

Comments

AddTable

Purpose	Adds a new table to a schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Sub` **`AddTable`**`(`*`PSDTable`*` As PSDTable)`
Parameters	→ *`PSDTable`* Specifies the `PSDTable` object to add to this schema database.
Returns	None.
Comments	This method adds the table information specified in *`PSDTable`* to this schema database.

AllBytesToBSTR

Purpose Creates a `String` from all the bytes in a string `Byte` array.

Applies to `PDUtility` object.

Prototype

```
Function AllBytesToBSTR(vData as Variant, nOffset
    as Long, pSubstitute as Byte, pString as
    String) as Long
```

Parameters

→ *vData*
: `Byte` array containing the data to convert.

→ *nOffset*
: Zero-based index into array `vData` from which to extract the string.

→ *pSubstitute*
: Substitute character for nondisplay values.

← *pString*
: Output string.

Returns The next offset in the `Byte` array.

Comments Use this method to convert mixed ASCII/non-ASCII arrays to displayable strings. This method substitutes a character for all nondisplay characters.

Example

```
Sub DisplayRecord(Record As Variant)
    Dim Utility As New PDUtility
    Dim Display As String
    Dim NextOffset As Long
    ' Convert input byte array to string
    NextOffset = Utility.AllBytesToBSTR(Record, _
        0, Asc("."), Display)
    txtValue.Text = Display
End Sub
```

AttachToTable

Purpose Attaches this object to a table's schema.

Applies to `PSDRowData` object.

Prototype

```
Sub AttachToTable(pAdapter As IPSDDatabaseAdapter,
    ByVal TableName As String)
```

Parameters

→ *`pAdapter`*
A `PSDDatabaseAdapter` object returned by `CreateDatabase()` or `OpenDatabase()`, which specifies the schema database that this row data is in.

→ *`TableName`*
The name of the table to attach this object to.

Returns None.

BackupDatabase

Purpose Backs up a handheld database to a directory or file on the desktop.

Applies to `PSDDatabaseUtilities` object.

Prototype

```
Sub BackupDatabase(ByVal Name As String, ByVal
    vCreatorID, ByVal vDBType, ByVal Attribute As
    EPSDDBAttribute, ByVal FilePath As String,
    ByVal bAlwaysBackup As Boolean,
    PSDDatabaseInfo As IPSDDatabaseInfo)
```

Parameters

→ *`Name`*
The **database name** as a null-terminated string. Do not pass in a null value.

→ *`vCreatorID`*
Creator ID of the database as a `Variant`—for example, `'adrs'`. See `PSDDatabaseInfo`.`CreatorID`.

→ *`vDBType`*
The database type as a `Variant`—for example, `'DATA'`. See `PSDDatabaseInfo`.`Type`. The type is only used as a cross-check and may be set to zero if you don't care what its value is.

→ *`Attribute`*
A `EPSDDBAttribute` enum value that specifies whether the database is a schema, extended, or classic database.

→ *`FilePath`*
The destination path or filename of the backup file, as a null-terminated string. If the caller specifies a directory path, then the Sync Manager generates the filename automatically, appends it to the specified directory path, and passes it back upon return; in this case *`FilePath`* must be large enough to contain the entire path (directory + filename). Do not pass in a null value.

→ *`bAlwaysBackup`*
If `True`, always back up the database. If `False`, backup the database only if necessary; for a description of how Sync Manager determines whether a database needs to be backed up, see `IsDatabaseBackupNeeded()`.

← *`PSDDatabaseInfo`*

A `PSDDatabaseInfo` object that receives values for the following properties of the database that this method backed up: `Attributes`, `BackupDate`, `CreationDate`, `CreatorID`, `Flags`, `IsReadOnlyDatabase`, `ModifyDate`, `ModifyNumber`, `Name`, `Type`, and `Version`. Note that this method uses the `Type` and `Attributes` passed *back* in this object to generate the backup filename, if *`FilePath`* is a directory.

Returns None.

Comments Based on the handheld database that you specify, this method creates or updates an image file of it on the desktop. This method enables a conduit to transfer an entire database to the desktop in a single call.

The caller specifies the database by its name, creator ID, type, and attributes (whether it is a classic, extended, or schema database).

NOTE: This function works only with handhelds running Palm OS Cobalt.

If *`FilePath`* is a directory, this function automatically generates the backup filename from the database creator, type, name, and attributes so that you do not have to call `GenerateBackupFileName()` first.

This function backs up the database only if *all* of the following statements are true:

- The file or path (*`FilePath`*) is writable.
- If the database is secure, the desktop must be trusted.
- *Either* of the following is true:
  - The corresponding call to `IsDatabaseBackupNeeded()` returns `True`. This method tests backup dates, backup bit, whether the database and file exist and whether their names, creator IDs, and types match.
  - The *`bAlwaysBackup`* parameter is `True`.

NOTE: If the database's backup bit is not set, this function does not back it up, unless you set `bAlwaysBackup` to `True`. But note that doing so causes this method to back up the database even if the specified image file on the desktop is already up to date.

Compatibility Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also `IsDatabaseBackupNeeded()`, `InstallAndBackupDatabase()` methods.

BackupSecurityData

Purpose Backs up vault databases from the handheld to a directory on the desktop.

Applies to `PSDDatabaseQuery`, `PSDDatabaseUtilities` object.

Prototype `Sub BackupSecurityData(ByVal Path As String)`

Parameters → *`Path`*
The path of the destination directory of the vault files, as a null-terminated string. The Sync Manager generates the filenames automatically. Do not pass in a null value.

Returns None.

Comments The Authorization Manager on the handheld stores all the relevant information to support secure databases—such as HEKs, rules, tokens, and so on—in one or more special secure databases called **vault**s. A backup conduit must back up vaults at the end of every HotSync session *after* all other databases. This function ensures that vaults are backed up in the order mandated by the Authorization Manager.

After a handheld is reset, vaults must be restored to the handheld *before* all other databases so that the Authorization Manager allows other secure databases to be restored afterwards. See `RestoreSecurityData()`.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

BeginProcess

Purpose Sets up connection to begin the synchronization process.

Applies to `IPDClientNotify` interface

Prototype `Function` **`BeginProcess`**`() as Boolean`

Returns A `Boolean` to indicate to the server whether to enable or disable connection. The client should return `True` to indicate that it is continuing to process. The client should return `False` to indicate that it is finished and that the server should shut down the connection.

Comments Clients use this method to process databases during the HotSync process. Clients can return either `True` or `False` depending on what they plan to do. Object requests processing continues until either a `False` is returned or the client releases all objects.

Example

```
Private Function IPDClientNotify_BeginProcess() As Boolean
' Procedure Main contains conduit code
Call Main
' Done processing. Return False to allow HotSync Manager to
' continue running other conduits.
IPDClientNotify_BeginProcess = False
End Function
```

BSTRToByteArray

Purpose Inserts a `String` into a `Byte` array.

Applies to `PDUtility` object.

Prototype

```
Function BSTRToByteArray (vData as Variant,
    nOffset as Long, pString as String) as Long
```

Parameters

← *vData*
`Byte` array to be manipulated.

→ *nOffset*
Zero-based index where insertion begins.

→ *pString*
String to insert.

Returns The next offset in the `Byte` array.

Comments Inserts `pString` plus a `Null` terminator in the `Byte` array at the position indexed by the integer value `nOffset`. These strings contain ASCII values, not Unicode.

Example

```
Sub InsertString(Record As Variant, Value As String)
    Dim Utility As New PDUtility
    Dim NextOffset As Long
   ' Insert the string in the array
    NextOffset = Utility.BSTRToByteArray(Record, 0, Value)
End Sub
```

BSTRToDWORD

Purpose Converts a four-character string to an unsigned `Long`—for example, to convert creator IDs and database types.

Applies to `PDUtility` object.

Prototype

```
Function BSTRToDWORD(pBstr As String, [bSwap As
    Boolean = True]) As Long
```

Parameters

→ *`pBstr`*
String to convert.

→ *`bSwap`*
If `True`, this method swaps the bytes in the returned `Long` value.

Returns The converted unsigned `Long`.

Example

```
Dim Utility As New PDUtility
Dim Value as String
Dim dwVal as Long
' Convert the string
Value = "Abcd"
dwVal = Utility.BSTRToDWORD(Value, True)
```

See Also `DWORDToBSTR()` method.

ByteArrayToBSTR

Purpose Extracts a `String` from a `Byte` array.

Applies to `PDUtility` object.

Prototype

```
Function ByteArrayToBSTR (vData as Variant,
    nOffset as Long, nLength as Long, pString as
    String) as Long
```

Parameters

→ *`vData`*
: `Byte` array containing the data to extract.

→ *`nOffset`*
: Zero-based index into array `vData` from which to extract the string (zero based).

→ *`nLength`*
: String length to extract.

← *`pString`*
: Output string.

Returns The next offset in the `Byte` array following the string just extracted.

Comments Extracts a string including the terminating `Null`. The `nLength` parameter permits you to specify a maximum string length for fields which may or may not be `Null`-terminated. For normal strings, set this to a large value (a number greater than the array length.)

Example

```
Sub GetString(Record As Variant, Value As String)
    Dim Utility As New PDUtility
    Dim nextOffset As Long
    ' Extract the string from the array
    nextOffset = Utility.ByteArrayToBSTR(Record, 0,_
        32767, Value)
End Sub
```

ByteArrayToDWORD

Purpose Extracts an unsigned `Long` from a `Byte` array (for example, a record ID).

Applies to `PDUtility` object.

Prototype
```
Function ByteArrayToDWORD (vData as Variant,
    nOffset as Long, bSwap as Boolean, nDWordVal
    as Long) as Long
```

Parameters

→ *`vData`*
Input `Byte` array.

→ *`nOffset`*
Zero-based index into array `vData` from which to extract the unsigned long value.

→ *`bSwap`*
If `True`, this method swaps the bytes before extracting.

← *`nDWordVal`*
Unsigned `Long` extracted by this method.

Returns The next offset in the `Byte` array.

Example
```
Sub ExtractDWORD(Record as Variant, Value as Long)
    Dim Utility As New PDUtility
    Dim NextOffset As Long
    ' Extract the value
    NextOffset = Utility.ByteArrayToDWORD(Record, 0, True, _
      Value)
End Sub
```

ByteArrayToHexBSTR

Purpose Converts an input `Byte` array to a formatted hex display `String`.

Applies to `PDUtility` object.

Prototype

```
Function ByteArrayToHexBSTR (vData as Variant,
    nOffset as Long, nCount as Long, pString as
    String) as Long
```

Parameters The `ByteArrayToHexBSTR` method syntax has these parts:

→ *vData*
Input `Byte` array.

→ *nOffset*
Zero-based index into array `vData` where conversion begins.

→ *nCount*
Number of bytes to convert.

← *pString*
Output hex display string.

Returns The next offset in the `Byte` array.

Comments The input `Byte` array is formatted into a standard hexadecimal display, format including CRLF pairs at the end of each sequence.

Input array:
This is a test. Use it in the text box.

```
Output string:
       000000 54 68 69 73 20 69 73 20 61 20 74 65 73 74 2E 20
This is a test.
       000010 55 73 65 20 69 74 20 69 6E 20 74 68 65 20 74 65
Use it in the text box.
       000020 78 74 20 62 6F 78 2E 2E
```

Example

```
Sub DisplayHex(Record As Variant)
    Dim Utility As New PDUtility
    Dim NextOffset As Long
    Dim Value as String
    Dim Count As Integer
    ' Convert the string
    Count = UBound(Record) - LBound(Record) + 1
    NextOffset =_
      Utility.ByteArrayToHexBSTR(Record, 0, Count, Value)
    ' Display the string
    txtValue.Text = Value
End Sub
```

ByteArrayToRecordId

Purpose Converts a `Byte` array to a record ID.

Applies to `PDUtility` object.

Prototype
```
Function ByteArrayToRecordId (vData as Variant,
    nOffset as Long, bSwap as Boolean, vRecordId
    as Variant) as Long
```

Parameters

→ *`vData`*
Input Byte array.

→ *`nOffset`*
Zero-based index into array `vData` from which to extract the record ID.

→ *`bSwap`*
If `True`, this method swaps the bytes before extracting.

← *`vRecordId`*
Record ID returned as a `Variant`.

Returns The next offset in the `Byte` array.

Comments This method is provided to convert a `Byte` array, usually read from the binary file that contains your record data, into a record ID. You can use the returned `vRecordId` in methods like `ReadById()`, `Write()`, and so on. Currently record IDs are long integers, but this may change in the future. PalmSource strongly recommends that you use `PDUtility` methods such as these to convert record IDs between `Byte` array and `String` formats.

Example
```
Sub ExtractRecordId(bArray as Variant, vRecordId as Variant)
Dim Utility As New PDUtility
Dim NextOffset As Long
' Extract the value
NextOffset = Utility.ByteArrayToRecordId(bArray, 0, True, _
   vRecordId)
End Sub
```

ByteArrayToWORD

Purpose Extracts an unsigned `Integer` from a `Byte` array.

Applies to `PDUtility` object.

Prototype

```
Function ByteArrayToWORD (vData as Variant,
   nOffset as Long, bSwap as Boolean, nWordVal as
   Integer) as Long
```

Parameters

→ *vData*
Input Byte array.

→ *nOffset*
Zero-based index into array `vData` where unsigned `Integer` value is extracted.

→ *bSwap*
If `True`, this method swaps the bytes before extracting.

← *nWordVal*
Unsigned `Integer` extracted by this method.

Returns The next offset in the `Byte` array.

Example

```
Sub InsertWORD(Record as Variant, Value as Integer)
   Dim Utility As New PDUtility
   Dim NextOffset As Long
   ' Extract the value
   NextOffset = Utility.ByteArrayToWORD(Record, 0, _
      False, Value)
End Sub
```

CallDeviceApplication

Purpose Calls an application on a Palm OS Cobalt handheld and returns data and status information to your conduit from that application.

Applies to `PSDDatabaseUtilities` object.

Prototype

```
Function CallDeviceApplication(ByVal DatabaseName
    As String, ByVal vCreatorID, ByVal Attribute
    As EPSDDBAttribute, ByVal ActionCode As Long,
    ByVal TypeID As Long, vInputData, vResultData)
    As Long
```

Parameters

→ *`DatabaseName`*
A null-terminated string that specifies the **database name** of the target application. Do not pass in a null value.

→ *`vCreatorID`*
The **creator ID** of the target application.

→ *`Attribute`*
A `EPSDDBAttribute` enum value that specifies whether the database is a schema, extended, or classic database.

→ *`ActionCode`*
The application-specific code that specifies the action to perform.

→ *`TypeID`*
Specifies the **database type** ID of the target application. Type is used only as a cross-check and may be set to zero if you don't care what the database type is.

→ *`vInputData`*
Input parameter array to the function called on the handheld.

← *`vResultData`*
A `Byte` array containing the results passed back to the conduit from the handheld application.

Returns The result code returned by the handheld application.

Comments This method works only with handhelds running Palm OS Cobalt, version 6.0.1 or later. For these handhelds, use this function rather than `CallRemoteModule()`. `CallDeviceApplication()` allows you to uniquely identify the target application by creator ID, database name, and database attributes (classic, extended, or schema).

Most conduits can accomplish their jobs without using this method. PalmSource recommends not using this method unless absolutely essential. An example of an unavoidable use of this method is when you need to call your application to create a secure database. For details, see "Creating Secure Databases" on page 134 in *Introduction to Conduit Development*.

This method enables the caller to send arbitrary data in the *`vInputData`* parameter to an application on the handheld during a HotSync operation. The application can pass back variable-sized information in *`vResultData`* as a byte array, which the caller can examine when this method returns.

Note that the format of the data and the action codes are completely application-specific. The handheld application that you call must have the same structure as a Palm OS application; however, the application can have a proprietary type ID so that it does not show up in the Launcher.

When a **Palm OS Protein application** is the target, you can set *`Attribute`* to `(ePSDSchemaDBType Or ePSDExtendedDBType)` to indicate that the application may resided in either an extended or schema database. When a **68K application** is the target, set *`Attribute`* to zero.

When you call `CallDeviceApplication()` from a conduit, the application on the handheld launches with a `sysAppLaunchCmdHandleSyncCallApp` launch code. To handle this launch code, the handheld application must cast the command parameter block passed to `PilotMain()` to a `SysAppLaunchCmdHandleSyncCallAppType` pointer. The `SysAppLaunchCmdHandleSyncCallAppType` structure contains all the information that the caller passed into `CallDeviceApplication()` on the desktop as well as the necessary fields to pass the result back to the desktop.

After the handheld application processes the `sysAppLaunchCmdHandleSyncCallApp` launch code, it must send a `DlkCallAppReplyParamType` reply back to the desktop using the DLServer's (`DLServer.h`) `DlkControl()` function.

Table 4.1 and Table 4.2 are some important mappings from this method's paramters to the `SysAppLaunchCmdHandleSyncCallAppType` and `DlkCallAppReplyParamType` structures on the handheld.

Table 4.1 Mapping from desktop `CallDeviceApplication()` parameters to handheld `SysAppLaunchCmdHandleSyncCallAppType`

`CallDeviceApplication()` parameters	**`SysAppLaunchCmdHandleSyncCallAppType` structure**
`ActionCode`	`action`
Size of *`vInputData`*	`dwParamSize`
`vInputData`	`paramP`

Table 4.2 Mapping from desktop desktop `CallDeviceApplication()` parameters to handheld `DlkCallAppReplyParamType`

`CallDeviceApplication()` parameters	**`DlkCallAppReplyParamType` structure**
Size of *`vResultData`*	`dwResultSize`
`vResultData`	`resultP`
`CallDeviceApplication()` return value	`dwResultCode`

For more information and example handheld application code, see *Exploring Palm OS: System Management*.

Example The sample below calls the HotSync client application with an action code that simply tests this method.

```
Dim pUtils As New PSDDatabaseUtilities
Dim ConvertCreator As New PDCondMgr
Dim vInputData As Variant
Dim vResultData As Variant
Dim nResultCode As Long
Dim i As Integer
Dim strData As String

nResultCode = pUtils.CallDeviceApplication("HotSync", _
   "sync", ePSDExtendedDBType, 1, _
   ConvertCreator.StringToCreatorID("appl"), vInputData, _
   vResultData)

If nResultCode = 0 Then
   For i = 0 To UBound(vResultData)
      If Chr(vResultData(i)) <> vbNullChar Then
         strData = strData + Chr(vResultData(i))
      End If
   Next i
   ' strData should contain the word "SUCCESS".
End If
```

Compatibility Palm OS version: Palm OS Cobalt, version 6.0.1 or later.

CallRemoteModule

Purpose Calls a module (an application, panel, or other executable) on the handheld and returns data and status information to your conduit from that module.

Applies to `PDSystemAdapter` object.

Prototype

```
Function CallRemoteModule (vCreator as Variant,
    vDbType as Variant, nAction as Integer,
    vParams as Variant, nResultSize as Long,
    nResultCode as Long) as Variant
```

Parameters

→ *`vCreator`*
Module creator ID. The unique ID associated with the application on the handheld.

→ *`vDbType`*
Database Type. Four characters that can be in either Long (VT_I4) or Little Endian form.

→ *`nAction`*
Action code specific to the module being called.

→ *`vParams`*
Input parameter array to the remote method.

↔ *`nResultSize`*
Before the call, the requested size (in bytes) of the return values block. After the call, the actual size (in bytes) of the return values block.

← *`nResultCode`*
Return code from the module.

Returns A `Byte` array containing the return values block.

Comments

IMPORTANT: This method works only with classic databases, which is how **68K application**s are stored. It works on both Palm OS Cobalt and Palm OS Garnet or earlier handhelds, but on Palm OS Cobalt handhelds it can call only a classic database. To call a **Palm OS Protein application** on a Palm OS Cobalt handheld, you must use `CallDeviceApplication()` instead.

Most conduits can accomplish their jobs without using this method. PalmSource recommends not using this function unless absolutely essential.

This method enables the caller to send arbitrary data in the *`vParams`* parameter to a module on the handheld during a HotSync operation. The module can pass back variable-sized information in this method's return value, which the caller can examine when this method returns.

Note that the format of the data and the action codes are completely module-specific. The handheld module that you call must have the same structure as a Palm OS application; however, the module can have a proprietary type ID so that it does not show up in the Launcher.

When you call `CallRemoteModule()` from a conduit, the module on the handheld launches with a `sysAppLaunchCmdHandleSyncCallApp` launch code. To handle this launch code, the handheld module must cast the command parameter block passed to `PilotMain()` to a `SysAppLaunchCmdHandleSyncCallAppType` pointer. The `SysAppLaunchCmdHandleSyncCallAppType` structure contains all the information that the caller passed into `CallRemoteModule()` on the desktop as well as the necessary fields to pass the result back to the desktop.

After the handheld module processes the `sysAppLaunchCmdHandleSyncCallApp` launch code, it must send a `DlkCallAppReplyParamType` reply back to the desktop using the DLServer's (`DLServer.h`) `DlkControl()` function.

Table 4.3 and Table 4.4 are some important mappings from this method's paramters to the `SysAppLaunchCmdHandleSyncCallAppType` and `DlkCallAppReplyParamType` structures on the handheld.

Table 4.3 Mapping from desktop `CallRemoteModule()` parameters to handheld `SysAppLaunchCmdHandleSyncCallAppType`

`CallRemoteModule()` parameters	**`SysAppLaunchCmdHandleSyncCallAppType` structure**
`nAction`	`action`
Size of *`vParams`*	`dwParamSize`
`vParams`	`paramP`

Table 4.4 Mapping from desktop `CallRemoteModule()` parameters to handheld `DlkCallAppReplyParamType`

`CallRemoteModule()` parameters	**`DlkCallAppReplyParamType` structure**
`nResultSize`	`dwResultSize`
`CallRemoteModule()` return value	`resultP`
`nResultCode`	`dwResultCode`

For more information and example handheld application code, see *Exploring Palm OS: System Management*.

For Palm OS Cobalt Handhelds

When *`vDbType`* is zero, `CallRemoteModule()` launches the first application returned by `DmFindDatabaseByTypeCreator(dmFindClassicDB)` with type `sysFileTApplication`. If there is no such application, DLServer launches the first application returned by `DmFindDatabaseByTypeCreator(dmFindClassicDB)` with type `sysFileTPanel`. Otherwise, DLServer returns an error. So `CallRemoteModule()` really treats setting *`vDbType`* to zero as a substitute for first calling with a type ID of `sysFileTApplication` then a type ID of `sysFileTPanel`.

As noted above, this method calls only classic databases; it cannot be used to call a **Palm OS Protein application** on a Palm OS Cobalt handheld.

Example

```
Dim pSystem as New PDSystemAdapter
Dim vParams as Variant, vResult as Variant
Dim nResultSize as Long, nResultCode as Long
nResultSize = 100
vResult = pSystem.CallRemoteModule("Abcd", "xxxx", 1, _
        vParams, nResultSize, nResultCode)
```

Compatibility

Palm OS version: Palm OS 2.0 or later. See "Comments."

On a handheld running Palm OS Cobalt, version 6.0 and a desktop running Sync Manager 2.4 or earlier, if both an ARM-native application (extended resource database) and a 68K application (classic resource database) with the same name and creator ID are present on the handheld, you cannot specify which application that this function calls. Therefore the application that this method calls is indeterminate in this situation. This problem has been fixed in Sync Manager 2.5 so that this method calls only classic databases.

See Also

`CallDeviceApplication()`

CfgConduit

Purpose Informs a conduit when the user selects it from HotSync Manager's **Custom** dialog box. Called only by HotSync Manager versions 3.0 and later (earlier versions call `ConfigureConduit()` instead).

Applies to `IPDClientNotify` interface

Prototype

```
Sub CfgConduit (nCreatorId as Long, nUserId as
    Long, BSTR bstrUserName, BSTR bstrPathName,
    nSyncPerm as Long, nSyncTemp as Long, nSyncNew
    as Long, nSyncPref as Long)
```

Parameters

→ *`nCreatorId`*
Creator ID of the conduit

→ *`nUserId`*
ID of the current user.

→ *`bstrUserName`*
Name of the current user.

→ *`bstrPathName`*
The fully-qualified path to the conduit.

↔ *`nSyncPerm`*
The kind of synchronization to perform on a permanent basis. This must be one of the `ESyncTypes` constants.

↔ *`nSyncTemp`*
The kind of synchronization to perform on a one-time (temporary) basis. This must be one of the `ESyncTypes` constants.

↔ *`nSyncNew`*
The kind of synchronization to perform for a new handheld. This must be one of the `ESyncTypes`.

↔ *`nSyncPref`*
Temporary or permanent change, see `ESyncPref` constants.

Returns None.

Comments HotSync Manager calls the `CfgConduit()` method when a user decides to configure your conduit by clicking **HotSync Manager** > **Custom**. Usually a conduit responds by displaying a “Change HotSync Action” dialog box for the user to configure how the conduit performs during the next and all subsequent HotSync operations.

`CfgConduit()` is an updated version of the `ConfigureConduit()` method, which is used for the same purpose. This method receives different information from what `ConfigureConduit()` does. Versions 3.0 and later of HotSync Manager will attempt to call `CfgConduit()` before falling back to calling `ConfigureConduit()`. If your conduit must support HotSync Manager versions earlier than 3.0, then you must implement `ConfigureConduit()` also.

Example

```
Private Sub IPDClientNotify_CfgConduit(ByVal nCreatorId _
        As Long, ByVal nUserId As Long, ByVal bstrUserName_
        As String,ByVal bstrPathName As String, nSyncPerm As_
        Long, nSyncTemp As Long, nSyncNew As Long, _
        nSyncPref As Long)
On Error GoTo ErrorHandler
' Read the current SyncType for the Configuration form
gConfigSyncType = nSyncPerm
' Show the Configuration form and save the user selected_
        HotSync state back to gConfigSyncType frmSetup._
        Show vbModal
'Return the new SyncType to Hotsync from our gConfigSyncType_
        saved variable
nSyncNew = gConfigSyncType
' Set this new HotSync type as the permament HotSync type.
nSyncPref = ePermanentPreference
Exit Sub

ErrorHandler:
    ' Do error handling here.
End Sub
```

See Also `ConfigureConduit()` method.
`ESyncPref`, `ESyncTypes` constants.

ChangeCategory

Purpose Changes all records of a particular category to a new category.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype

```
Sub ChangeCategory(ByVal nOldCategory As Long,
    ByVal nNewCategory As Long)
```

Parameters

→ *`nOldCategory`*
Original category ID.

→ *`nNewCategory`*
New category ID.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenGet Conduit _
    InfoRecordDatabase("Memo", “PDDirect PDRecordAdapter”’ _
    eRead Or eWrite)
Adapter.ChangeCategory(1, 2)
```

ChangeFileDestinationHHToSlot

Purpose Changes the destination of a file that is already queued to be installed in primary storage on a user's *handheld* instead to be installed in secondary storage in an expansion *slot*.

Applies to `PDInstall` object.

Prototype

```
Sub ChangeFileDestinationHHToSlot(UserID As Long,
   SlotID As Long, FileName As String)
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`SlotID`*
The ID of the slot to install the file to. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

→ *`FileName`*
The name of the file to move (include no path).

Returns None.

Errors

`eInvalidPath`
The path of the slot-install directory is longer than 256 characters and cannot be retrieved.

`eMoveFailed`
This method failed to move the specified install file because, for example, the file does not exist.

`eParamError`
Parameters were not passed correctly.

Comments This method moves a file on the desktop computer from the associated handheld-install directory—that is, the directory associated with the install conduit registered to handle files of the type to be moved—to the specified slot-install directory. This method accepts only file types that are registered with an install conduit—for example, `.prc`, `.pdb`, and `.pqa` file types are registered with the Install conduit that ships with HotSync Manager, so this method can move such files to a user's slot-install directory.

Example

```
Dim PInstall As New PDInstall
Dim UserData As New PDUserData
Dim UserId As Long

' Retrieve the user ID from the HotSync Manager user name.
UserId = UserData.GetIDFromName("Palm OS Emulator")

Call PInstall.ChangeFileDestinationHHToSlot(UserId, 0, _
   "c:\temp\MyApp.prc")
```

See Also `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `ChangeFileDestinationSlotToHH()`, `ChangeFileSlotDestination()` methods

ChangeFileDestinationSlotToHH

Purpose Changes the destination of a file that is already queued to be installed in secondary storage in an expansion *slot* instead to be installed in primary storage on a user's *handheld*.

Applies to `PDInstall` object.

Prototype

```
Sub ChangeFileDestinationSlotToHH(UserID As Long,
    SlotID As Long, FileName As String)
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`SlotID`*
The ID of the *source* slot (the one *from* which to move the file). The file currently exists in the slot-install directory of this slot. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

→ *`FileName`*
The name of the file to move (include no path).

Returns None.

Errors

`eInvalidPath`
The path of the slot-install directory is longer than 256 characters and cannot be retrieved.

`eMoveFailed`
This method failed to move the specified install file because, for example, the file does not exist.

`eParamError`
Parameters were not passed correctly.

Comments This method moves a file on the desktop computer from the specified slot-install directory to the associated handheld-install directory—that is, to the directory associated with the install conduit registered to handle files of the specified type. This method accepts only file types that are registered with an install conduit—for example, `.prc`, `.pdb`, and `.pqa` file types are registered with the Install conduit that ships with HotSync Manager, so this method can move such files to a user's handheld-install directory.

Example

```
Dim PInstall As New PDInstall
Dim UserData As New PDUserData
Dim UserId As Long

' Retrieve the user ID from the HotSync Manager user name.
UserId = UserData.GetIDFromName("Palm OS Emulator")

Call PInstall.ChangeFileDestinationSlotToHH(UserId, 0, _
   "c:\temp\MyApp.prc")
```

See Also

`GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `ChangeFileDestinationHHToSlot()`, `ChangeFileSlotDestination()` methods

ChangeFileSlotDestination

Purpose Changes from one expansion *slot* to another the destination of a file that is already queued to be installed in secondary storage in an expansion *slot* of a user's handheld.

Applies to `PDInstall` object.

Prototype

```
Sub ChangeFileSlotDestination(UserID As Long,
    SourceSlotID As Long, TargetSlotID As Long,
    File As String)
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`SourceSlotID`*
The ID of the *source* slot (the one *from* which to move the file). The file currently exists in the install directory of this slot. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

→ *`TargetSlotID`*
The ID of the *target* slot (the one *to* which to move the file). The file is to be moved to the install directory of this slot. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

→ *`File`*
The name of the file to move (include no path).

Returns None.

Errors

`eInvalidPath`
The path of the slot-install directory is longer than 256 characters cannot be retrieved.

`eMoveFailed`
This method failed to move the specified install file because, for example, the file does not exist.

`eParamError`
Parameters were not passed correctly.

Comments This method moves a file on the desktop computer from one slot-install directory to another slot-install directory. It is useful for users whose handhelds have multiple slots. This method accepts all file types.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `ChangeFileDestinationHHToSlot()`, `ChangeFileDestinationSlotToHH()` methods

Close

Purpose Closes this open file on an expansion card.

Applies to `PDVFSFileManager` object.

Prototype `Sub` **`Close`**`()`

Parameters None.

Returns None.

Errors

`eVFSFileBadRef`
The file reference number is invalid: it has been closed or was not obtained from `Open()`.

`eVFSInvalidOperation`
A file system is not present.

`eVFSNotOpen`
The file system library on the handheld necessary for this call has not been installed or has not been opened.

Comments Use `Close` to close a file or directory that has been opened with `PDVFSVolumeManager.Open()`.

See Also `PDVFSVolumeManager` object.
`Open()` method.

CloseDatabase

Purpose Closes an open schema database.

Applies to `PSDDatabaseQuery` object.

Prototype

```
Sub CloseDatabase(PSDDatabaseAdapter As
    IPSDDatabaseAdapter, [ByVal options As
    EPSDCloseOptions = ePSDNone], [ByVal
    bSeenAllChanges As Boolean = False])
```

Parameters

→ *`PSDDatabaseAdapter`*
A `PSDDatabaseAdapter` object representing the schema database to close.

→ *`options`*
Option flags to indicate whether this method updates the backup and modification dates and whether the desktop has seen all changes. You can specify a combination of the `EPSDCloseOptions` values.

→ *`bSeenAllChanges`*
If `True`, this method marks the database as successfully synchronized by this conduit and updates the change context. Note that this change tracking mechanism is specific to a conduit.

Returns None.

Comments You must call this method to close a schema database before your conduit finishes. The COM Sync module does this automatically for non-schema databases, but not for schema databases.

ConfigureConduit

Purpose Informs a conduit when the user selects it from HotSync Manager's **Custom** dialog box. HotSync Manager versions earlier than 3.0 call only this method, whereas versions 3.0 and later call the `CfgConduit()` method first and then call `ConfigureConduit()` only if the call to `CfgConduit()` is not successful.

Applies to `IPDClientNotify` interface

Prototype

```
Sub ConfigureConduit(ByVal pPathName as String,
    ByVal pRegistry as String, nSyncPref as Long,
    nSyncType as Long)
```

Parameters The `Configuration` method syntax has these parts:

→ *`pPathName`*
Path to desktop data filename.

→ *`pRegistry`*
Full path to your conduit configuration entries.

↔ *`nSyncPref`*
Temporary or permanent change, see `ESyncPref` constants.

↔ *`nSyncType`*
Synchronization type, see `ESyncTypes` constants.

Returns None.

Comments As with `CfgConduit()`, HotSync Manager calls the `ConfigureConduit()` method when a user decides to configure your conduit by clicking **HotSync Manager** > **Custom**. Usually a conduit responds by displaying a "Change HotSync Action" dialog box for the user to configure how the conduit performs during the next and all subsequent HotSync operations.

However, the `ConfigureConduit()` method is an older version of the `CfgConduit()` method, which is used for the same purpose but provides different information. Versions of HotSync Manager earlier than 3.0 call the `ConfigureConduit()` method only; newer versions first attempt to call the `CfgConduit()` method, and fall back to calling this method if `CfgConduit()` is not available.

Example

```
Private Sub IPDClientNotify_ConfigureConduit(ByVal pPathName_
        As String, ByVal pRegistry As String, nSyncPref As_
        Long, nSyncType As Long)

   On Error GoTo ErrorHandler
   ' For older Hotsync versions
   ' Set the SyncType for the configuration form

   gConfigSyncType = nSyncPref
   frmSetup.Show vbModal

   'Set the SyncType for Hotsync
   nSyncPref = gConfigSyncType

ErrorHandler:
   ' Do error handling here
End Sub
```

See Also

`CfgConduit()` method.
`ESyncPref`, `ESyncTypes` constants.

CopyFileFromDeskTop

Purpose Copies a file from the desktop to a volume on a handheld expansion card.

Applies to `PDVFSVolumeManager` object.

Prototype

```
Sub CopyFileFromDeskTop(DeskTopFileName As String,
    DeviceFileName As String)
```

Parameters

→ *`DeskTopFileName`*
The full path and filename of the source file on the desktop to copy.

→ *`DeviceFileName`*
Full path and filename of the destination file on the handheld. All parts of the path, except the file, must exist. Can also be set to `Null` (see “Comments” below).

Returns None.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSBadName`
Invalid filename or path.

`eVFSDirectoryNotFound`
The path, excluding filename, does not exist or no default directory is registered for this file type.

`eVFSDiskFileAccess`
Failed to create or open the disk file on the desktop.

`eVFSFileAccessOther`
Could not access or map the desktop file—for example, because of insufficient memory on the desktop.

`eVFSFileAlreadyExists`
A directory with this name exists in this location already.

`eVFSFileNotFound`
The file was not found in the specified path.

`eVFSFilePermissionDenied`
: Permission denied to perform requested operation—for example, an attempt to write to a read-only file or to read a file already opened in the `eVFSModeExclusive` mode.

`eVFSInvalidOperation`
: A file system is not present.

`eVFSNoFileSystem`
: None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
: The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
: The volume reference number is invalid.

`eVFSVolumeFull`
: There is insufficient space left on the volume.

Comments The behavior of this method depends on whether a destination path is specified:

- If `DeviceFileName` is specified, all parts of the path, except the filename, must already exist.
  - If the full path exists, this method copies the file to specified location.
  - If the full path does *not* exist, this method fails and returns an error.
- If `DeviceFileName` is `Null`:
  - If a default directory is registered for this file type, this method ensures that the entire path exists—creating the directories leading up to the default directory, if necessary—and puts the file in the default directory.
  - If no default directory is registered for this file type, this method returns `eVFSDirectoryNotFound`.

If the path exists in either of the above cases, this method copies the file specified by `DeskTopFileName` to the destination on the expansion card. If the file already exists at the destination, this method overwrites it with the one specified by `DeskTopFileName`.

See Also `GetDefaultDirectory()`, `CopyFileToDeskTop()` methods.

CopyFileToDeskTop

Purpose Copies a file from a volume on a handheld expansion card to the desktop.

Applies to `PDVFSVolumeManager` object.

Prototype

```
Sub CopyFileToDeskTop(DeviceFileName As String,
    DeskTopFileName As String)
```

Parameters

→ *`DeviceFileName`*
The full path and filename of the source file on the handheld volume to copy.

→ *`DeskTopFileName`*
The full path and filename for the file to be created on the desktop. All parts of the path, except the file, must already exist. If the file does not exist, then this method creates it. If the file exists, then it overwrites the file.

Returns None.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSDiskFileAccess`
Failed to create or open the disk file on the desktop.

`eVFSDiskFull`
Not enough space on the desktop's disk.

`eVFSFileAccessOther`
Could not access or map the desktop file—for example, because of insufficient memory on the desktop.

`eVFSFileAccessOther`
Could not access or map the desktop file—for example, because of insufficient memory on the desktop.

`eVFSFileBadRef`
The file reference number is invalid: it has been closed or was not obtained from `Open()`.

`eVFSFilePermissionDenied`
: Permission denied to perform requested operation—for example, an attempt to write to a read-only file or to read a file already opened in the `eVFSModeExclusive` mode.

`eVFSInvalidOperation`
: A file system is not present.

`eVFSIsADirectory`
: This operation can be performed only on a regular file, not a directory.

`eVFSNoFileSystem`
: None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
: The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
: The volume reference number is invalid.

Comments Using this method to copy a file to the desktop is easier than opening the file on the expansion card and reading it into a buffer on the desktop.

See Also `Open()`, `CopyFileFromDeskTop()` methods.

CreateDatabase

Purpose Creates a schema database.

Applies to `PSDDatabaseQuery` object.

Prototype

```
Function CreateDatabase(ByVal DatabaseName As
    String, ByVal vCreatorID, ByVal vType, ByVal
    Version As Integer, PSDTable As IPSDTable,
    [ByVal Flags As EPSDDatabaseFlags =
    ePSDSchema]) As IPSDDatabaseAdapter
```

Parameters

→ *`DatabaseName`*
The **database name** as a null-terminated string. Do not pass in a null value.

→ *`vCreatorID`*
Creator ID of the database as a `Variant`—for example, `'adrs'`. See `PSDDatabaseInfo`.`CreatorID`.

→ *`vType`*
The database type as a `Variant`—for example, `'DATA'`. See `PSDDatabaseInfo`.`Type`.

→ *`Version`*
The database version number. See `PSDDatabaseInfo`.`Version`.

→ *`PSDTable`*
A `PSDTable` object, which defines a schema. This method creates only one table in the new database. You can add more by calling `AddTable()`.

→ *`Flags`*
A combination of one or more `EPSDDatabaseFlags` values.

Returns A `PSDDatabaseAdapter` object that represents the newly created schema database. This method also opens this database for read-write access in exclusive sharing mode with private records shown.

CreateDirectory

Purpose Creates a directory on this volume on a handheld expansion card.

Applies to `PDVFSVolumeManager` object.

Prototype `Sub **CreateDirectory**(*Directory* As String)`

Parameters

→ *`Directory`*
The full path of the directory to create.

Returns None.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSBadName`
Invalid filename or path.

`eVFSDirectoryNotFound`
The full path, not including the new directory name, does not exist.

`eVFSFileAlreadyExists`
A directory with this name exists in this location already.

`eVFSInvalidOperation`
A file system is not present.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
The volume reference number is invalid because, for example, the volume has not been mounted.

`eVFSVolumeFull`
There is insufficient space left on the volume.

Comments All parts of the path except the last component must already exist.

See Also `PDVFSFileManager` object.
`Open()`, `Delete()`, `Rename()`, `GetFileList()`, `GetSubDirectoryList()`, `CreateFile()` methods.

CreateFile

Purpose Creates a file on this volume on a handheld expansion card.

Applies to `PDVFSVolumeManager` object.

Prototype `Sub **CreateFile**(*FileName* As String)`

Parameters

→ *`FileName`*
The full path and filename of the file to create. All parts of the path, excluding the filename, must already exist.

Returns None.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSBadName`
Invalid filename or path.

`eVFSFileAlreadyExists`
A file with this name exists in this location already.

`eVFSInvalidOperation`
A file system is not present.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
The volume reference number is invalid because, for example, the volume has not been mounted.

`eVFSVolumeFull`
There is insufficient space left on the volume.

Comments This method does not open the file. All parts of the path, except the last part must already exist. Any read/write operations that you want to perform on this file require a `PDVFSFileManager` object, which you create by calling the `Open()` method.

It is the responsibility of the file system library on the handheld to ensure that all filenames are translated into a format that is compatible with the native format of the file system, such as the 8.3 convention for a FAT file system without long filename support. See "Directory Paths" on page 100 in the *COM Sync Suite Companion* for a description of how to construct a valid path.

See Also PDVFSFileManager object.
Open(), Delete(), Rename(), GetFileList(), CreateDirectory() methods.

CreateRecordDatabase

Purpose Creates a new classic or extended record database on the handheld.

Applies to `DmDatabaseQuery`, `PDDatabaseQuery` object.

Prototype

```
Function CreateRecordDatabase(ByVal pDbName As
    String, ByVal pAdapterName As String, ByVal
    vCreator, ByVal vDbType, [ByVal eAccessMode As
    EAccessModes], [ByVal EDbFlags As EDbFlags =
    eRecord], [ByVal nVersion As Long = 1], [ByVal
    nCardNum As Long]) As IUnknown
```

Parameters

→ *pDbName*
Name of database to open (case sensitive, 1-31 characters).

→ *pAdapterName*
Full name of the COM Sync database adapter to use. Names are of this form:

```
LibraryName.AdapterName
```

Do not include “Lib” in the name—for example, use `PDDirect.PDRecordAdapter`, not `PDDirectLib.PDRecordAdapter`.

→ *vCreator*
Creator ID. The unique ID associated with each database and application on the device. Each conduit is associated with a specific creator ID. It is four characters that can be in either `Long` (`VT_I4`) or Little Endian form.

→ *vDbType*
Database type. It is four characters that can be in either `Long` (`VT_I4`) or Little Endian form. If a `BSTR` (`VT_BSTR`), only the first four characters are used.

→ *eAccessMode*
Access mode from `EAccessModes` constants.

→ *EDbFlags*
Database flags from the `EDbFlags` constants.

→ *nVersion*
Version.

→ *nCardNum*
Card number.

Returns A database adapter object of the type you specify in the `pAdapterName` parameter. Possible returned objects include `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, and `PDTodoDbHHRecordAdapter` objects.

Comments The *vCreator* and *vDbType* parameters are `Variant`. This permits you to enter a `String` or an unsigned `Long` of the same value.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
' Create a new database
Set Adapter = DbQuery.CreateRecordDatabase_
   ("New Database", "PDDirect.PDRecordAdapter", "Abcd",_
   "DATA", eRead Or eWrite, eBackupDb, 1, 0)
```

See Also `DmDatabaseQuery`, `DmRecordAdapter`, `PDRecordAdapter`, `PDRecordAdapter` objects.
`OpenRecordDatabase()` method.
`EDbFlags`, `EAccessModes` constants.

CreateResourceDatabase

Purpose Creates a new resource database on the handheld.

Applies to `PDDatabaseQuery` object.

Prototype

```
Function CreateResourceDatabase(ByVal pDbName As
    String, ByVal pAdapterName As String, ByVal
    vCreator, ByVal vDbType, [ByVal eAccessMode As
    EAccessModes], [ByVal EDbFlags As EDbFlags =
    eResource], [ByVal nVersion As Long = 1],
    [ByVal nCardNum As Long]) as PDResourceAdapter
```

Parameters

→ *pDbName*
Database name (1-31 characters).

→ *pAdapterName*
ProgID of the database adapter to use.

→ *vCreator*
Creator ID. The unique ID associated with each database and application on the device. Each conduit is associated with a specific creator ID. It is four characters that can be in either `Long` (`VT_I4`) or Little Endian form.

→ *vDbType*
Database type. It is four characters that can be in either `Long` (`VT_I4`) or Little Endian form. If a `BSTR` (`VT_BSTR`), only the first four characters are used.

→ *eAccessMode*
Access mode from `EAccessModes` constants.

→ *EDbFlags*
Database flags from the `EDbFlags` constants.

→ *nVersion*
Database version.

→ *nCardNum*
Card number where you are creating the resource database.

Returns A `PDResourceAdapter` object that represents the resource database.

Comments The *vCreator* and *vDbType* parameters are `Variant`. This permits you to enter a `String` or an unsigned `Long` of the same value.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDResourceAdapter
' Create a new database
Set Adapter = _
       DbQuery.CreateResourceDatabase("New Application", _
       "PDDirect.PDResourceAdapter", "Abcd", "appl", _
        eRead Or eWrite, eBackupDb, 1, 0)
```

See Also

`PDResourceAdapter` object
`EDbFlags`, `EAccessModes` constants

CreatorIDToString

Purpose Converts a `Long` conduit creator ID into a `String`.

Applies to `PDCondMgr`, `PDSystemCondMgr` objects.

Prototype `Function` **`CreatorIDToString`**`(`*`CreatorID`*` As Long) As String`

Parameters → *`CreatorID`*
The `Long` creator ID that you want to convert.

Returns A creator ID as a `String`.

Errors `eInvalidID`
The specified conduit creator ID is not valid.

`eParamError`
Parameters were not passed correctly.

Example

```
Dim CreatorID As Long
Dim strResult As String
Const strCreator = "memo"
Dim PDcond As New PDCondMgr

' Converted the string value to a Long and back again
CreatorID = PDcond.StringToCreatorID(strCreator)
strResult = PDcond.CreatorIDToString(CreatorID)
```

Also see the example under `RegisterConduit()`.

See Also `StringToCreatorID()` method

Delete

Purpose Deletes a closed file or empty directory on this volume on a handheld expansion card.

Applies to `PDVFSVolumeManager` object.

Prototype `Sub **Delete**(*Name* As String)`

Parameters → *`Name`*
The full path of the file or directory to delete.

Returns None.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSBadName`
Invalid filename or path.

`eVFSDirNotEmpty`
The directory is not empty and therefore cannot be deleted.

`eVFSFileNotFound`
The file was not found in the specified path.

`eVFSFilePermissionDenied`
Permission denied to perform requested operation—for example, an attempt to write to a read-only file or to read a file already opened in the `eVFSModeExclusive` mode.

`eVFSFileStillOpen`
The file is still open—for example, trying to delete an open file.

`eVFSInvalidOperation`
A file system is not present.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
: The volume reference number is invalid because, for example, the volume has not been mounted.

Comments A directory must be empty before this method can delete it. A file must be closed before this method can delete it.

See Also `PDVFSFileManager` object.
`Close()`, `GetFileList()`, `GetSubDirectoryList()`, `Rename()`, `CreateFile()`, `CreateDirectory()` methods.

DeleteAllRowsInTable

Purpose Marks all rows as deleted in a table in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Sub DeleteAllRowsInTable(ByVal TableName As
    String)
```

Parameters → *`TableName`*
The name of the table from which to delete rows.

Returns None.

Comments This method does not remove the rows’ data; it only sets their Deleted flags.

See Also `RemoveRow()`

DeleteDatabase

Purpose Deletes a schema database and all of its data.

Applies to `PSDDatabaseQuery` object.

Prototype

```
Sub DeleteDatabase(ByVal DatabaseName As String,
  ByVal vCreatorID)
```

Parameters

→ *`DatabaseName`*
The **database name** as a null-terminated string. Do not pass in a null value.

→ *`vCreatorID`*
Creator ID of the database as a `Variant`—for example, `'adrs'`. See `PSDDatabaseInfo`.`CreatorID`.

Returns None.

DeleteKey

Purpose Deletes a key or an entire section from the specified user's area of the users data store.

Applies to `PDUserData` object.

Prototype

```
Sub DeleteKey(UserID As Long, Section As String,
    Key as String)
```

Parameters

→ *`UserID`*
A unique ID to specify the user.

→ *`Section`*
The section name in the specified user's area of the users data store.

→ *`Key`*
The key of the integer to delete. If this is `Null`, then the entire section is deleted.

Returns None.

Errors

`eInvalidUser`
`UserID` is an invalid number.

`eNoCorePath`
No path to find the users data store was found.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eSaveErr`
Saving changes was not successfully completed.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `AddNewUser()` methods

DeleteRow

Purpose Marks a row as deleted in a schema database.

Applies to `PSDRowAdapter` object.

Prototype `Sub DeleteRow(ByVal vRowID)`

Parameters → *`vRowID`*
The row ID of the row to delete.

Returns None.

Comments This method does not destroy the row's data; it only sets the row's "deleted" flag. Contrast this method with `RemoveRow()`.

DeleteRowsInCategory

Purpose Deletes rows whose category IDs match those on the specified list according to the specified match mode.

Applies to `PSDDatabaseAdapter` object.

Prototype
```
Sub DeleteRowsInCategory(vCategoryIDList, ByVal
    MatchMode As EPSDMatchMode)
```

Parameters

→ *`vCategoryIDList`*
A `Variant` array of category IDs.

→ *`MatchMode`*
The category match mode that this method uses to match the specified category ID list against rows' category memberships. Specify one of the `EPSDMatchMode` values.

Returns None.

Comments

DeleteUser

Purpose Deletes a user from the users data store.

Applies to `PDUserData` object.

Prototype `Sub DeleteUser(dwUserId As Long)`

Parameters → *`dwUserId`*

A unique ID to specify the user to delete.

Returns None.

Errors

`eInvalidUser`
: `dwUserId` is an invalid number.

`eNoCorePath`
: No path to find the users data store was found.

`eNoUsers`
: The users data store exists, but contains no information.

`eOtherUDErr`
: No users data store was found or another method or program is accessing the user data store.

`eSaveErr`
: Saving changes was not successfully completed.

`eUDSemaphoreError`
: Another method or program is accessing the user data store.

`eUDUnableToCreate`
: Creating a new users data store failed because of a file error.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `AddNewUser()` methods

DeleteUserPermanentSyncPreferences

Purpose Deletes the permanent synchronization preferences for *all* of the specified user's conduits.

Applies to `PDUserData` object.

Prototype `Sub` **`DeleteUserPermanantSyncPreferences`**`(`*`dwUserId`*` As Long)`

Parameters → *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

Returns None.

Errors

`eInvalidUser`
`dwUserId` is an invalid number.

`eNoCorePath`
No path to find the users data store was found.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eSaveErr`
Saving changes was not successfully completed.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

Comments This method clears the permanent synchronization preferences for *all* conduits. The result is the same as if the user has never clicked HotSync Manager's **Custom** > **Change** option and altered any permanent synchronization preferences.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `DeleteUserTemporarySyncPreferences()`, `RemoveUserTemporarySyncPreferences()`, `GetUserTemporarySyncPreferences()`, `SetUserTemporarySyncPreferences()`, `GetUserPermanentSyncPreferences()`, `SetUserPermanentSyncPreferences()` methods

DeleteUserTemporarySyncPreferences

Purpose Deletes the temporary synchronization preferences for *all* of the specified user's conduits.

Applies to `PDUserData` object.

Prototype `Sub` **`DeleteUserTemporarySyncPreferences`**`(`*`dwUserId`*` As Long)`

Parameters → *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

Returns None.

Errors

`eInvalidUser`
`dwUserId` is an invalid number.

`eNoCorePath`
No path to find the users data store was found.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eSaveErr`
Saving changes was not successfully completed.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

Comments This method clears the temporary synchronization preferences for *all* conduits so that the actions set in the conduits' permanent synchronization preferences will be taken during the next HotSync operation. The result is the same as if the user has never clicked HotSync Manager's **Custom** > **Change** option and altered any temporary synchronization preferences.

NOTE: This method clears *all* conduits' temporary synchronization preferences. Contrast it with `RemoveUserTemporarySyncPreferences()`, which clears the temporary preferences for only *one* conduit.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `RemoveUserTemporarySyncPreferences()`, `GetUserTemporarySyncPreferences()`, `SetUserTemporarySyncPreferences()`, `DeleteUserPermanentSyncPreferences()`, `GetUserPermanentSyncPreferences()`, `SetUserPermanentSyncPreferences()` methods

DisplayLog

Purpose Displays the **HotSync Log** dialog box of the HotSync Manager application.

Applies to `PDHotSyncUtility` object.

Prototype `Sub DisplayLog(dwUserId As Long)`

Parameters → *`dwUserId`*
A unique ID to specify the user whose log you want to display. If this value is 0, the current user's log is displayed.

Returns None.

Errors `eHotSyncNotFound`
HotSync Manager is not running.

Comments This method displays the HotSync Log dialog box, which enables the user to view log entries written to it during previous HotSync operations.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `AddLogEntry()` methods

DWORDToBSTR

Purpose Converts an unsigned `Long` to a four-character `String`. Used for creator IDs, database type, and others.

Applies to `PDUtility` object.

Prototype

```
Function DWORDToBSTR(nDWordVal As Long, [bSwap As
    Boolean = True]) As String
```

Parameters

→ *`nDWordVal`*
: Unsigned `Long` to convert.

→ *`bSwap`*
: If `True`, this method swaps the bytes in *`nDWordVal`* before returning the `String` value.

Returns The unsigned `Long` value expressed as a four-character `String`.

Example

```
Dim Utility As New PDUtility
Dim Value as String
Dim dwVal as Long
' Convert the string
dwVal = &H6d656d6f        'memo creator ID
Value = Utility.DWORDToBSTR(dwVal, True)
```

See Also `BSTRToDWORD()` method.

DWORDToByteArray

Purpose Inserts an unsigned `Long` into a `Byte` array.

Applies to `PDUtility` object.

Prototype
```
Function DWORDToByteArray(pvData as Variant,
    nOffset As Long, bSwap As Boolean, nDWordVal
    As Long) As Long
```

Parameters

↔ *pvData*
: `Byte` array used for insertion.

→ *nOffset*
: Offset location where the unsigned `Long` is inserted.

→ *bSwap*
: If `True`, the method swaps the bytes before inserting them.

→ *nDWordVal*
: Unsigned `Long` to insert.

Returns The next offset in the `Byte` array.

Example
```
Dim barray(7) As Byte
   Dim Vdata As Variant
   Dim ToPos As Long
   Dim Backpos As Long
   Dim Result As Long
   Dim putil As New PDUtility
   Dim i As Integer
   Dim strDisplay As String

   ' Init the variant
   Vdata = barray

   ' DWord so 4 bytes
   barray(0) = Asc("H")
   barray(1) = Asc("e")
   barray(2) = Asc("l")
   barray(3) = Asc("l")

   barray(4) = Asc("o")
   barray(5) = Asc(" ")
   barray(6) = Asc("!")
   barray(7) = Asc("!")
```

```
While ToPos < UBound(barray) - 1
   ToPos = putil.ByteArrayToDWORD(barray, ToPos, False, _
      Result)
   ' Convert Back
   Backpos = putil.DWORDToByteArray(Vdata, Backpos, _
      False, Result)
Wend

For i = 0 To UBound(barray)
   strDisplay = strDisplay & Chr(Vdata(i))
Next i

MsgBox strDisplay

MsgBox "StrConv --> " & StrConv(Vdata, vbUnicode)
```

ExportDatabaseToFile

Purpose Flattens and exports the specified database on the handheld to the specified PDB or PRC file on an expansion card. Works only with classic databases.

Applies to `PDVFSVolumeManager` object.

Prototype

```
Sub ExportDatabaseToFile(PathName As String,
    CardNumber As Long, DBName As String)
```

Parameters

→ *`PathName`*
The full path and filename of the destination file to create. All parts of the path, excluding the filename, must already exist.

→ *`CardNumber`*
The RAM card number in the handheld on which the database exists. Note that this does not refer to the expansion card and is therefore not related to the slot reference number. The card number for the first RAM memory card on the handheld is 0, which is the only one that most handhelds have.

→ *`DBName`*
The name of the source database to export to a file on the expansion card.

Returns None.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSBadName`
Invalid filename or path.

`eVFSFileAlreadyExists`
A file with this name exists in this location already.

`eVFSInvalidOperation`
A file system is not present.

Comments This utility method exports a database from primary storage memory on a handheld to a PDB or PRC file on an expansion card. This method is the opposite of `ImportDatabaseFromFile()`. Use this method, for example, to copy applications from primary storage to an expansion card.

IMPORTANT: This method works only with classic databases. It cannot export schema or extended databases.

See Also `Write()`, `ImportDatabaseFromFile()` methods.

Format

Purpose Formats and mounts this volume.

Applies to `PDVFSVolumeManager` object.

Prototype `Sub` **`Format`**`(`*`mountClass`*` As Long)`

Parameters

→ *`mountClass`*

This parameter is used when mounting the volume after it has been formatted. For possible values to use, see "VFS Volume Mount Class Constants" on page 577. You can pass in the same value as returned by this volume's `mountClass` property.

Returns None.

Errors

`eVFSInvalidOperation`

A file system is not present.

`eVFSNoFileSystem`

None of the file systems installed on the handheld support this operation.

`eVFSNotEnoughPower`

Insufficient battery power on the handheld to perform the operation.

`eVFSNotOpen`

The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeStillMounted`

The volume is still mounted, but this method formats it anyway.

Comments This method can only reformat an already formatted expansion card; it cannot format an unformatted card. This method attempts to find a compatible file system library on the handheld to format and then to remount the volume if the format succeeds. (The handheld slot driver provided by PalmSource, Inc. currently supports only one volume per slot.) In the process, the VFS Manager assigns a new volume reference number to this volume. Despite the change in its volume reference number, you can continue to use the same `PDVFSVolumeManager` object from which you formatted the volume.

See Also `GetVolumeManager()` method.

GenerateBackupFileName

Purpose Generates the unique backup filename of a database specified by its name, creator, type, and attributes.

Applies to `PSDDatabaseQuery`, `PSDDatabaseUtilities` objects.

Prototype

```
Function GenerateBackupFileName(ByVal vCreatorID,
    ByVal DatabaseName As String, ByVal Attributes
    As Long, ByVal vType) As String
```

Parameters

→ *`vCreatorID`*
Creator ID of the database. See `PSDDatabaseInfo`.`CreatorID`.

→ *`DatabaseName`*
The **database name** as a null-terminated string. Do not pass in a null value. See `PSDDatabaseInfo`.`Name`.

→ *`Attributes`*
The attributes of this database. Specify a combination of one or more of the `EPSDDatabaseFlags` and `EDbFlags` values. See `PSDDatabaseInfo`.`Attributes`.

→ *`vType`*
The database type. See `PSDDatabaseInfo`.`Type`.

Returns The generated backup filename.

Comments Versions of Palm OS *earlier than Palm OS Cobalt* uniquely identify databases by name only. Therefore, in versions of HotSync Manager *earlier than 6.0,* the default Backup conduit creates backup filenames that consist of only the database name with an extension that depends on the database attributes (PRC for classic resource databases and PDB for classic record databases). The case of database names is incorrectly ignored in backup filenames.

However, Palm OS Cobalt uniquely identifies schema and extended databases by *name and creator ID.* Therefore the Backup conduit that ships with HotSync Manager 6.0 and later generates backup filenames for all databases based on both their name and creator ID. This method generates such filenames.

This method generates standard backup filenames that a conduit can pass into the `BackupDatabase()` to create an image file on the desktop of a database on the handheld—though, you do not have to call this method before `BackupDatabase()`, because it can generate the filename itself. Another use for this method is to determine whether a database has a backup image on the desktop already—that is, whether it has been backed up.

This method generates filenames with extensions based on the database attributes and type as shown in Table 4.5.

Table 4.5 Backup filename extensions

Extension	Database description
PRC	Classic resource databases
PDB	Classic or extended record databases
SDB	Schema databases (nonsecure)
SSD	Secure schema databases
VLT	Security vault databases

The Sync Manager uses a name-mangling scheme to prevent collisions between Palm OS database names, which are case-sensitive, and Windows filenames, which are case-insensitive.

GetAllQueuedHHFiles

Purpose Retrieves a list of all the files queued to be installed in the handheld's main memory for the specified user.

Applies to `PDInstall` object.

Prototype

```
Function GetAllQueuedHHFiles(UserID As Long) As
    Variant
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

Returns A list of filenames as an array of string values.

Errors

`eInvalidUser`
`UserID` is an invalid number.

`eParamError`
Parameters were not passed correctly.

Comments This method builds a list of all the files in the specified user's handheld-install directory. See "Installing Files on the Handheld with PDInstall" on page 58 in the *COM Sync Suite Companion*.

Example

```
Dim PInstall As New PDInstall
Dim UserData As New PDUserData
Dim UserId As Long
Dim HHQueueList As Variant

' Retrieve the user ID from the HotSync Manager user name.
UserId = UserData.GetIDFromName("Palm OS Emulator")

Call PInstall.InstallFileToHH(UserId, "c:\temp\MyApp.prc")
HHQueueList = PInstall.GetAllQueuedHHFiles(UserId)
```

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetAllQueuedHHFilesOfType()`, `GetAllQueuedSlotFiles()`, `InstallFileToHH()`, `InstallFileToSlot()` methods

GetAllQueuedHHFilesOfType

Purpose Retrieves a list of all the files of the specified *type* that are queued to be installed in the handheld's main memory for the specified user.

Applies to `PDInstall` object.

Prototype

```
Function GetAllQueuedHHFilesOfType(UserID As Long,
    Extension As String) As Variant
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`Extension`*
A string ("*`*.extension`*") specifying the filename extension to search for—for example, `"*.prc"`, `"*.pdb"`, or `"*.pnc"`.

Returns A list of filenames as an array of `String` values.

Errors

`eInvalidUser`
`UserID` is an invalid number.

`eParamError`
Parameters were not passed correctly.

Comments This method builds a list of all the files in the specified user's handheld-install directory. See "Installing Files on the Handheld with PDInstall" on page 58 in the *COM Sync Suite Companion*.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetAllQueuedHHFiles()`, `GetAllQueuedSlotFiles()`, `InstallFileToHH()`, `InstallFileToSlot()` methods

GetAllQueuedSlotFiles

Purpose Retrieves a list of all the files queued to be installed to the handheld's specified expansion slot for a given user.

Applies to `PDInstall` object.

Prototype

```
Function GetAllQueuedSlotFiles(UserID As Long,
    SlotID As Long) As Variant
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`SlotID`*
The ID of the slot for which to get a list of queued files. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

Returns A list of filenames as an array of `String` values.

Errors

`eInvalidUser`
`UserID` is an invalid number.

`eParamError`
Parameters were not passed correctly.

Comments This method builds a list of all the files in the specified user's slot-install directory. This directory holds all the files that will be installed on the specified expansion slot on the handheld during the next HotSync operation. See "Installing Files on the Handheld with PDInstall" on page 58 in the *COM Sync Suite Companion*.

Example

```
Dim PInstall As New PDInstall
Dim UserData As New PDUserData
Dim UserId As Long
Dim SlotQueueList As Variant

' Retrieve the user ID from the HotSync Manager user name.
UserId = UserData.GetIDFromName("Palm OS Emulator")

Call PInstall.InstallFileToSlot(UserId, 0, _
   "c:\temp\MyApp.prc")
SlotQueueList = PInstall.GetAllQueuedSlotFiles(UserId, 0)
```

See Also

`GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `GetAllQueuedHHFilesOfType()`, `GetAllQueuedHHFiles()`, `InstallFileToHH()`, `InstallFileToSlot()` methods

GetBackupConduit

Purpose	Retrieves the name of HotSync Manager's **backup conduit**.
Applies to	`PDCondMgr`, `PDSystemCondMgr` objects.
Prototype	`Function` **`GetBackupConduit`**`() As String`
Parameters	None.
Returns	The filename of the backup conduit as a `String`.

Errors

`eParamError`
: Parameters were not passed correctly.

`eRegistryFailure`
: Unable to access the conduit configuration entries.

Comments This method retrieves the value of the `HotSync Manager\BackupConduit` configuration entry used by HotSync Manager for the current Windows user or for the system, depending on whether this method is called for a `PDCondMgr` or a `PDSystemCondMgr` object.

Example

```
Dim PDcond As New PDCondMgr
Dim strBackupConduit As String
strBackupConduit = PDcond.GetBackupConduit
```

See Also `SetBackupConduit()` method

GetCardInfo

Purpose Retrieves information about an expansion card in a given slot.

Applies to `PDExpansionManager` object.

Prototype `Function **GetCardInfo**(*slotNum* As Long) As Unknown`

Parameters → *`slotNum`*

The slot reference number of the slot to retrieve information about. Call `GetSlotReferenceNumbers()` to get these values.

Returns A `PDExpansionCardInfo` object.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eVFSCardNotPresent`
No card is present in the given slot.

`eVFSInvalidSlotNumber`
The slot reference number is not valid.

`eVFSNoSectorReadWrite`
The card does not support the slot driver block read/write API.

`eVFSSlotDeallocated`
The slot reference number is within the valid range, but the Expansion Manager has unloaded the slot driver on the handheld.

`eVFSUnsupportedOperation`
Either virtual file systems are not supported on the handheld or the handheld does not have an expansion slot.

Comments This method returns information about an expansion card, including whether the card supports secondary storage or is strictly read-only, by filling in the properties of a `PDExpansionCardInfo` object.

See Also `PDExpansionCardInfo` object.
`GetSlotReferenceNumbers()`, `IsExpansionSlotPresent()` methods.

GetCategoryAdapter

Purpose	Returns a category adapter object for a schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Function` **`GetCategoryAdapter`**`() As IPSDCategoryAdapter`
Parameters	None.
Returns	A `PSDCategoryAdapter` object.

GetCategoryCount

Purpose	Retrieves the number of categories to which this row belongs.
Applies to	`PSDRowData` object.
Prototype	`Function` **`GetCategoryCount`**`() As Long`
Parameters	None.
Returns	A count of this row's category memberships.

GetCategoryMembership

Purpose Retrieves a row's category memberships in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Function GetCategoryMembership(ByVal vRowID) As
    Variant
```

Parameters → *`vRowID`*
The row ID of the row to get category memberships of.

Returns A `Variant` array of category IDs that represent the categories that this row is a member of.

Comments

GetChangeContext

Purpose Retrieves the change context for a schema database from the handheld.

Applies to `PSDDatabaseQuery` object.

Prototype

```
Function GetChangeContext(ByVal DatabaseName As
    String, ByVal vCreatorID, ByVal vType) As
    Variant
```

Parameters

→ *`DatabaseName`*
The **database name** as a null-terminated string. Do not pass in a null value. See `PSDDatabaseInfo`.`Name`.

→ *`vCreatorID`*
Creator ID of the database. See `PSDDatabaseInfo`.`CreatorID`.

→ *`vType`*
The database type. See `PSDDatabaseInfo`.`Type`.

Returns A `Variant` array of bytes that represents the change context for the specified schema database.

Comments The Sync Manager stores the **change context** per user, conduit, and schema database and retrieves it automatically to determine the type of synchronization operation. Therefore most conduits do not need to store the change context themselves, but instead can rely on the Sync Manager to handle it. However, in some special cases a conduit might need to store the change context itself. For example, if the data source that the conduit synchronizes with resides in a central location, and the user synchronizes from multiple desktops with the same central data source using the same conduit, the conduit can store this change context in the same central location. Then during subsequent HotSync operations from different desktops, the conduit can retrieve the change context and pass it to the Sync Manager via `GetSyncTypeInfo()` to determine the synchronization mode that the conduit should use. After comparing the change contexts, the Sync Manager might indicate a fast sync is possible when it would not have been possible otherwise.

NOTE: A conduit must call this method *after* closing the database, but only if it needs to store a database's change context itself. Most conduits do not.

Compatibility Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also `GetSyncTypeInfo()` method

GetColumnCount

Purpose	Retrieves the number of columns in this table.
Applies to	`PSDTable` object.
Prototype	`Function` **`GetColumnCount`**`() As Long`
Parameters	None.
Returns	A count of this table's columns.

GetColumnCustomProperty

Purpose Retrieves the value of a custom column property in a table.

Applies to `PSDDatabaseAdapter` object.

Prototype

```
Function GetColumnCustomProperty(ByVal TableName
    As String, ByVal ColumnID As Long, ByVal
    PropertyID As Integer) As Variant
```

Parameters

→ *`TableName`*
The name of the table.

→ *`ColumnID`*
The column ID of the column.

→ *`PropertyID`*
The property ID of the custom column property. Valid values range from 0x05 to 0x0A.

Returns A byte array containing the value of the specified custom column property.

GetColumnIDList

Purpose	Retrieves the column IDs of all columns in this table.
Applies to	`PSDTable` object.
Prototype	`Function` **`GetColumnIDList`**`() As Variant`
Parameters	None.
Returns	A `Variant` array of column IDs.

GetColumnInfoByID

Purpose Retrieves a column definition from this table given a column ID.

Applies to `PSDTable` object.

Prototype

```
Function GetColumnInfoByID(ByVal ColumnID As Long)
    As IPSDColumnInfo
```

Parameters → *`ColumnID`*

A column ID to retreive the column definition of.

Returns A `PSDColumnInfo` object that represents the specified column definition.

GetColumnInfoByName

Purpose Retrieves a column definition from this table given a column name.

Applies to `PSDTable` object.

Prototype

```
Function GetColumnInfoByName(ByVal ColumnName As
    String) As IPSDColumnInfo
```

Parameters → *`ColumnName`*
The name of a column to retreive the column definition of.

Returns A `PSDColumnInfo` object that represents the specified column definition.

GetColumnNames

Purpose	Retrieves a list of all the column names in this table.
Applies to	`PSDTable` object.
Prototype	`Function` **`GetColumnNames`**`(`*`vNames`*`) As Long`
Parameters	← *`vNames`* A `Variant` array of column names.
Returns	The number of column names passed back in *`Names`*.

GetColumnsWithData

Purpose Retrieves a list of names of the columns in this row that contain data.

Applies to `PSDRowData` object.

Prototype `Function GetColumnsWithData(ColumnNames) As Long`

Parameters ← *`ColumnNames`*

A `Variant` array of column names. These are the columns that contain data.

Returns The number of column names passed back in *`ColumnNames`*.

GetCommStatus

Purpose Retrieves the status of the HotSync Manager application's communication types.

Applies to `PDHotSyncUtility` object.

Prototype

```
Function GetCommStatus(type As
    EPDHSConnectionType) As EPDHSConnectionStatus
```

Parameters → *`type`*

The communication type of which to retrieve the status. Use one of the values defined by the `EPDHSConnectionType` constant.

Returns The status of the specified communication type as a value of the `EPDHSConnectionStatus` constant.

Errors `eInvalidConnType`

The specified HotSync Manager connection type is not one defined by the `EPDHSConnectionType` constant.

See Also `SetCommStatus()` method.
`EPDHSConnectionType` constant.

GetConduitCount

Purpose Returns the number of conduits that are registered with HotSync Manager for the current Windows user or for the system.

Applies to `PDCondMgr`, `PDSystemCondMgr` objects.

Prototype `Function` **`GetConduitCount`** `() As Long`

Parameters None.

Returns The total number of conduits.

Errors `eParamError`

Parameters were not passed correctly.

Comments The returned count includes all conduits that are registered for either the current Windows user or the system, depending on whether it is called for a `PDCondMgr` or a `PDSystemCondMgr` object. This count includes conduits registered by placement in the user's `Conduits` folder (folder-registered, C API-based conduits) and those registered by configuration entries (conventionally registered). This count does not include backup or install conduits, nor does it include system-registered conduits.

Example

```
Dim ConduitCount As Long
Dim PCondMgr As New PDCondMgr
ConduitCount = PCondMgr.GetConduitCount()
```

See Also `GetConduitList()` method

GetConduitInfo

Purpose Returns complete information about a conduit in a `PDConduitInfo` object.

Applies to `PDCondMgr`, `PDSystemCondMgr` objects.

Prototype
```
Function GetConduitInfo(CreatorID As Long) As
    Unknown
```

Parameters

→ *`CreatorID`*
The creator ID of the conduit you want information about.

Returns A `PDConduitInfo` object.

Errors

`eInvalidID`
The specified conduit creator ID is not valid.

`eLocalMemory`
Not enough memory on the desktop to perform the requested operation.

`eNoSuchConduit`
The specified conduit does not exist.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments If this method succeeds in finding a registered conduit with a matching creator ID, it returns a `PDConduitInfo` object.

Example See the example under `RegisterConduit()`.

See Also `PDConduitInfo` object.
`StringToCreatorID()`, `RegisterConduit()`, `GetConduitList()` methods.

GetConduitInfo

Purpose Returns information about the conduit (including name and version) when requested by HotSync Manager.

Applies to `IPDClientNotify` interface

Prototype

```
Function GetConduitInfo(infoType As
    EGetConduitInfo, dwCreatorId As Long, dwUserId
    As Long, bstrUserName As String)
```

Parameters

→ *`infoType`*
Describes the type of information being requested as one of the `EGetConduitInfo` values.

→ *`dwCreatorId`*
Specifies the creator ID.

→ *`dwUserId`*
Specifies the user ID.

→ *`bstrUsername`*
Specifies the handheld user name.

Returns The data specified by the *`infoType`* parameter.

Comments HotSync Manager calls `GetConduitInfo()` to retrieve information about your conduit. Your implementation of this entry point must respond differently for each `EGetConduitInfo` value passed in the *`infoType`* parameter. Table 4.6 lists these values and how your conduit should respond.

Table 4.6 GetConduitInfo() requests and conduit responses

If the `EGetConduitInfo` value passed in ***infoType*** is...	Then return...
`Constant eGetConduitName = 0`	the display name of your conduit.
`Constant eGetMfcVersion = 1`	an `EMfcVersion` enum value.
`Constant eGetDefaultAction = 2`	an `ESyncTypes` enum value.
`Constant eGetConduitVersion = 3`	the version number of your conduit.
`Constant ePDDoNotDisplayInConduitListForUser = 4`	0, if your conduit should be displayed in the **Custom** dialog box. Return a nonzero value if it should not be displayed.
`Constant ePDRunAlways = 5`	`EPDRunOptions.ePDRunOnlyWhenAppExists`, if your conduit should be run only if a matching application is on the handheld. Return a value of `EPDRunOptions.ePDRunConduitAlways` if your conduit should always be run.
`Constant ePDDoNotDisplayProgress = 6`	0, if your conduit should be displayed in the **HotSync Progress** dialog box. Return a nonzero value if it should not be displayed.

Example

```
Private Function IPDClientNotify_GetConduitInfo(_
   ByVal infoType As PDDirectLib.EGetConduitInfo, _
   ByVal dwCreatorId As Long, ByVal dwUserId As Long, _
   ByVal bstrUserName As String) As Variant

   If infoType = eGetConduitName Then
      IPDClientNotify_GetConduitInfo = "SimpleDbExe"
   End If

   If infoType = eGetDefaultAction Then
      IPDClientNotify_GetConduitInfo = PDDirectLib.eFast
   End If
   If infoType = eGetMfcVersion Then
      IPDClientNotify_GetConduitInfo = _
         PDDirectLib.ePDMFC_NOT_USED
   End If

   If infoType = eGetConduitVersion Then
      IPDClientNotify_GetConduitInfo = 1#
   End If

   ' Run this conduit even if there is no application on the
   ' device with the corresponding creator ID.
   ' Removes the requirement that the creator ID for which
   ' the conduit is registered be present on the device for
   ' the conduit to run. The default behavior is to run your
   ' conduit always, even when an application with the same
   ' creator ID (as the one you registered your conduit with)
   ' does not exist on the device. If you do not want your
   ' conduit to run when the application is not present on
   ' device, then return ePDRunOnlyWhenAppExists.

   If infoType = ePDRunAlways Then
      IPDClientNotify_GetConduitInfo = _
         EPDRunOptions.ePDRunConduitAlways
   End If

   ' Do not opt out of display in HotSync Manager Custom
   ' dialog box.
   If infoType = ePDDoNotDisplayInConduitListForUser Then
      IPDClientNotify_GetConduitInfo = False
   End If

   ' Do not opt out of display in the HotSync Progress
   ' dialog box.
   If infoType = ePDDoNotDisplayProgress Then
      IPDClientNotify_GetConduitInfo = False
   End If

End Function
```

Compatibility

HotSync Manager Versions 6.0 and Later

These versions can pass in only these values in the *`infoType`* parameter:

- `eGetConduitName`
- `eGetDefaultAction`
- `ePDDoNotDisplayInConduitListForUser`
- `ePDRunAlways`
- `ePDDoNotDisplayProgress`

If a conduit does not handle the case when *`infoType`* `= ePDRunAlways`, then versions 6.0 and later of HotSync Manager run the conduit regardless of whether an application with the same creator ID is on the handheld.

HotSync Manager versions 6.0 and later do not need to check a conduit's MFC version, so they never pass in the `eGetMfcVersion` value via the *`infoType`* parameter.

HotSync Manager Versions Earlier than 6.0

These versions can pass in only these values in the *`infoType`* parameter:

- `eGetConduitName`
- `eGetMfcVersion`
- `eGetDefaultAction`

Versions of HotSync Manager earlier than 6.0 run the conduit only if an application with the same creator ID is on the handheld. These versions to not enable a conduit to opt out of this requirement.

If your conduit must work with HotSync Manager versions earlier than 6.0, your implementation of `GetConduitInfo()` must return the appropriate MFC version constant. If you are recompiling a conduit you created with an older version of the CDK and did not originally implement `GetConduitInfo()`, HotSync Manager assumes that your conduit is built on MFC 4.1. If it is not, HotSync Manager crashes when it calls your conduit.

GetConduitList

Purpose Returns a list of creator IDs of all the conduits registered for either the current Windows user or the system.

Applies to `PDCondMgr`, `PDSystemCondMgr` objects.

Prototype `Function **GetConduitList**() as Variant`

Parameters None.

Returns A list of creator IDs as an array of `Long` values.

Errors `eParamError`

Parameters were not passed correctly.

`eRegistryFailure`

Unable to access the conduit configuration entries.

Comments This method returns information about conduits that are registered for the current Windows user or the system, depending on whether it is called for a `PDCondMgr` or a `PDSystemCondMgr` object.

Example

```
Dim ConduitList As Variant
Dim PCondMgr As New PDCondMgr

ConduitList = PCondMgr.GetConduitList

' Check whether the returned array contained entries.
' If it's not empty, return the string CreatorID of the first
' conduit.
If Not IsEmpty(ConduitList) Then
   MsgBox "The first CreatorID is '" & _
      PCondMgr.CreatorIDToString(ConduitList(0)) & "'", _
      vbInformation
End If
```

See Also `CreatorIDToString()`, `GetConduitInfo()` methods

GetCount

Purpose	Retrieves the number of categories in a schema database.
Applies to	`PSDCategoryAdapter` object.
Prototype	`Function **GetCount**() As Long`
Parameters	None.
Returns	The category count.

GetCurrentRowID

Purpose	Retrieves the current row ID in this set of rows.
Applies to	`PSDRowSet` object.
Prototype	`Function` **`GetCurrentRowID`**`() As Variant`
Parameters	None.
Returns	The row ID of the row in this set that the cursor points to.

GetDatabaseHandle

Purpose	Returns the handle of this open schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Function` **`GetDatabaseHandle`**`() As Byte`
Parameters	None.
Returns	The handle of this open schema database.
Comments	This method is provided for testing purposes, so you probably do not need to use it. It returns the handle that the underlying C API uses to address this open schema database. No COM Sync methods require that you use this handle, though.

GetDatabaseInfo

Purpose	Retrieves information about this schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Function` **`GetDatabaseInfo`**`() As IPSDDatabaseInfo`
Parameters	None.
Returns	A `PSDDatabaseInfo` object.

GetDataSize

Purpose Retrieves the size of a column value in this row.

Applies to `PSDRowData` object.

Prototype

```
Function GetDataSize(ByVal ColumnName As String)
    As Long
```

Parameters → *`ColumnName`*

The name of a column in this row.

Returns The size of the specified column value in bytes.

Comments This method is useful for getting the size of variable-length data types such as strings.

GetDataType

Purpose Retrieves the data type of a column in this row.

Applies to `PSDRowData` object.

Prototype

```
Function GetDataType(ByVal ColumnName As String)
    As EPSDColumnDataType
```

Parameters → *`ColumnName`*
The name of a column in this row.

Returns One of the `EPSDColumnDataType` values that specifies the data type of data in this column.

GetDefaultDirectory

Purpose Retrieves the default directory on this volume on an expansion card for files of the specified type.

Applies to `PDVFSVolumeManager` object.

Prototype

```
Function GetDefaultDirectory(FileType As String)
   As String
```

Parameters → *`FileType`*

The file type may either be a MIME media type/subtype pair, such as “image/jpeg”, “text/plain”, or “audio/basic”; or a file extension, such as “.jpeg”. If you pass in a file extension, it must begin with a period ‘`.`’—for example “`.prc`”.

Returns The path of the default directory for the requested file type.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSBadName`
Invalid filename or path.

`eVFSFileNotFound`
The file was not found in the specified path.

`eVFSInvalidOperation`
A file system is not present.

Comments This method returns the full path to the default directory registered for the specified file type. A default directory can be registered for each type of media supported. The directory should be registered under media and file type. (Note that this directory is typically a "root" directory for the file type; any subdirectories under this root directory should also be searched for files of the appropriate type.) If this method finds no match for either the specified media type for this volume or the requested file type, it returns `eVFSFileNotFound`.

This method can be used by an image viewer application, for example, to find the directory containing images without having to know what type of media the volume was on. This could be "`/DCIM`", "`/images`", or something else depending on the type of media.

For more information, see "Determining the Default Directory for a Particular File Type" on page 102 in the *COM Sync Suite Companion*.

See Also `GetFileList()`, `GetSubDirectoryList()` methods.

GetDeskTopTrustStatus

Purpose Determines whether the HotSync operation in progress is with a trusted desktop.

Applies to `PSDDatabaseQuery`, `PSDDatabaseQuery` objects.

Prototype

```
Function GetDeskTopTrustStatus() As
    EPSDDesktopTrustStatus
```

Parameters None.

Returns A `EPSDDesktopTrustStatus` value.

Comments Call this method only during a HotSync operation, because the Sync Manager must retrieve the desktop trust status from the handheld.

This method passes back `ePSDDesktopTrustNotVerified`, if it is called after the Sync Manager connects to the handheld but before the Sync Manager performs authentication.

GetDWORDData

Purpose Retrieves a `DWORD` configuration entry value for the specified conduit.

Applies to `PDCondMgr`, `PDInstallConduit`, `PDSystemCondMgr` objects.

Prototype `PDCondMgr` and `PDSystemCondMgr`:

```
Function GetDWORDData(CreatorID As Long, Name As
    String) As Long
```

`PDInstallConduit`:

```
Function GetDWORDData(UniqueId As Long, Name As
    String) As Long
```

Parameters

→ *`CreatorID`*
If a `PDCondMgr` or `PDSystemCondMgr` object, this parameter is the creator ID of the conduit you want a value for.

→ *`UniqueId`*
If a `PDInstallConduit` object, this parameter is the unique ID of the install conduit you want a value for.

→ *`Name`*
The name of the `DWORD` configuration entry you want the value of.

Returns The value of the specified `DWORD` configuration entry.

Errors

`eInvalidID`
The specified conduit creator ID is not valid.

`eNoSuchConduit`
The specified install conduit does not exist.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

`eValueNotFound`
The specified value could not be found in the configuration entries for this conduit.

Comments This is a general purpose method for retrieving a conduit configuration entry value by name. If the conduit you want is a standard synchronization conduit (most are), specify the creator ID in the first parameter. If the conduit you want is an **install conduit**, specify the unique ID in the first parameter.

This method returns information about a conduit that is registered either for the current Windows user or the system, depending on whether it is called for a `PDCondMgr` or a `PDSystemCondMgr` object.

Example

```
Dim ExtraInfo As Long
Dim CreatorId As Long
Dim PCondMgr As New PDCondMgr

' Set the value for a custom field called "ExtraInfo"
' to 10.
CreatorId = PCondMgr.StringToCreatorID("memo")

Call PCondMgr.SetDWORDData(CreatorId, "ExtraInfo", 10)
ExtraInfo = PCondMgr.GetDWORDData(CreatorId, "ExtraInfo")
```

See Also `SetDWORDData()` method

GetExceptionDates

Purpose	Retrieves a list of dates that are exceptions to a Date Book repeating event.
Applies to	`PDDateBookDbHHRecord2` object.
Prototype	`Function` **`GetExceptionDates`**`() As Variant`
Parameters	None.
Returns	A `Variant` array of values of type `Date`. These are the dates on which a repeating Date Book event does not occur.
See Also	`SetExceptionDates()` method.

GetFileList

Purpose Retrieves the names of all the files in a given directory.

Applies to `PDVFSVolumeManager` object.

Prototype
```
Function GetFileList(Directory As String,
   FileList As Variant) As Long
```

Parameters

→ *`Directory`*
: The full path of the directory to retrieve the file list from.

← *`FileList`*
: A list of all the files in the specified directory, passed back as an array of `String` values represented as a `Variant`.

Returns The number of filenames passed back in `FileList`.

Errors

`eCommunications`
: Communications with the handheld has either not been initialized or has been lost.

`eParamError`
: Parameters were not passed correctly.

`eVFSBadName`
: Invalid path.

`eVFSEnumerationEmpty`
: No volumes are present to enumerate or none remain to enumerate.

`eVFSFileBadRef`
: The file reference number is invalid: it has been closed or was not obtained from `Open()`.

`eVFSInvalidOperation`
: A file system is not present.

`eVFSNoFileSystem`
: None of the file systems installed on the handheld support this operation.

`eVFSNotADirectory`
: This operation can be performed only on a directory.

`eVFSNotOpen`
: The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
: The volume reference number is invalid because, for example, the volume has not been mounted.

Comments This method retrieves only filenames. Use `GetSubDirectoryList()` to retrieve subdirectory names.

See Also `GetSubDirectoryList()`, `Open()` methods.

GetHHFileSize

Purpose Retrieves the size of the specified file that is queued to be installed to the handheld's main memory for a given user.

Applies to `PDInstall` object.

Prototype

```
Function GetHHFileSize(UserID As Long, FileName As
    String) As Long
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`FileName`*
The name of the file to get the size of (filename only, not a path).

Returns The size of the specified file in bytes.

Errors

`eInvalidUser`
`UserID` is an invalid number.

`eParamError`
Parameters were not passed correctly.

Comments This method returns the size of a file in the specified user's handheld-install directory. See "Installing Files on the Handheld with PDInstall" on page 58 in the *COM Sync Suite Companion*.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetAllQueuedHHFiles()`, `GetAllQueuedHHFilesOfType()` methods

GetIDFromName

Purpose Retrieves a unique user ID given the user's name.

Applies to `PDUserData` object.

Prototype `Function **GetIDFromName**(*UserName* As String) As Long`

Parameters

→ *`UserName`*
A string containing the name of the user. It must be no more than 20 characters long.

Returns The user ID.

Errors

`eNoCorePath`
No path to find the users data store was found.

`eNoUsers`
The users data store exists, but contains no information.

`eNoUsers`
`UserName` does not exist in the users data store.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

Comments Note that it is possible for the users data store to contain the same name more than once. Because the user ID is the only value that `PDUserData` ensures is unique, each instance of the same name has a different user ID. Therefore you must perform additional checking to determine whether the user name is unique before you use the user ID returned by this method.

See Also `GetIDFromPath()`, `GetUserNameFromID()`, `SetUserName()` methods

GetIDFromPath

Purpose Retrieves a user ID given the user directory.

Applies to `PDUserData` object.

Prototype `Function GetIDFromPath(Path As String) As Long`

Parameters

→ *`Path`*
A string containing the path to the user directory.

Returns The user ID.

Errors

`eInvalidUserDir`
The specified user directory does not match that of any current user.

`eNoCorePath`
No path for the users data store was found.

`eNoUsers`
The users data store exists, but contains no information, or the user does not exist in it.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

See Also `GetUserDirectory()`, `GetIDFromName()`, `SetUserDirectory()` methods

GetIDList

Purpose	Retrieves a list of the category IDs in a schema database.
Applies to	`PSDCategoryAdapter` object.
Prototype	`Function` **`GetIDList`**`() As Variant`
Parameters	None.
Returns	A category ID list as a variant.

GetIntegerValue

Purpose Retrieves an integer value from a key in the specified user's area of the users data store.

Applies to `PDUserData` object.

Prototype

```
Function GetIntegerValue(dwUserId As Long, Section
    As String, Key As String) As Long
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`Section`*
The section name in the specified user's area of the users data store.

→ *`Key`*
The key of the integer to retrieve.

Returns The integer value as specified.

Errors

`eInvalidUser`
`dwUserID` is an invalid number.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `SetIntegerValue()`, `DeleteKey()`, `GetStringValue()`, `SetStringValue()` methods

GetModifiedIDList

Purpose	Retrieves a list of the IDs of modified categories in a schema database.
Applies to	`PSDCategoryAdapter` object.
Prototype	`Function` **`GetModifiedIDList`**`() As Variant`
Parameters	None.
Returns	A category ID list as a variant.

GetModifiedTableNames

Purpose Retrieves the names of tables that have been modified since the last HotSync operation.

Applies to `PSDDatabaseAdapter` object.

Prototype
```
Function GetModifiedTableNames(vTableNames) As
    Long
```

Parameters ← *vTableNames*
A `Variant` array that contains a list of modified table names.

Returns The number of modified tables.

GetNameList

Purpose	Retrieves the names of all of the categories in a schema database.
Applies to	`PSDCategoryAdapter` object.
Prototype	`Function` **`GetNameList`**`() As Variant`
Parameters	None.
Returns	A category name list as a variant.

GetNotifierList

Purpose Returns a list of all the registered notifier filenames.

Applies to `PDCondMgr` object.

Prototype `Function GetNotifierList() as Variant`

Parameters None.

Returns A list of filenames as an array of `String` values.

Errors `eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Example

```
Dim NotifierList As Variant
Dim PCondMgr As New PDCondMgr
NotifierList = PCondMgr.GetNotifierList

' Check if the returned array contained entries.
' If it's not empty return the filename of the first
' notifier.
If Not IsEmpty(NotifierList) Then
   MsgBox "The first notifier is '" & NotifierList(0) & _
      "'", vbInformation
End If
```

See Also `RegisterNotifier()`, `ModifyNotifier()`, `UnregisterNotifier()` methods

GetPath

Purpose Retrieves one of the stored desktop paths.

Applies to `PDInstall` object.

Prototype `Function` **`GetPath`**`(`*`type`*` As EPDPathType) As String`

Parameters → *`type`*

A constant of type `EPDPathType` that specifies which path name you want to retrieve.

Returns The requested path as a string.

Errors `eParamError`

Parameters were not passed correctly.

Comments This method retrieves one of the paths stored in the HotSync Manager configuration entries on the desktop (see "HotSync Manager Configuration Entries" on page 188 in the *Introduction to Conduit Development*).

See Also `SetPath()` method.
`EPDPathType` constant.

GetRootDirectory

Purpose Retrieves the path of all user directories on the desktop computer (as stored in the `Core\Path` HotSync Manager configuration entry).

Applies to `PDUserData` object.

Prototype `Function` **`GetRootDirectory`**`() As String`

Parameters None.

Returns The path.

Errors `eNoCorePath`
No path for the users data store was found.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

Comments `GetRootDirectory()` retrieves the value stored in the `Core\Path` configuration entry. Because HotSync Manager versions 6.0 and later are aware of multiple Windows users, each Windows user must have a separate `Core\Path` value. To meet Windows standards for the placement of user data, a typical value is `C:\Documents and Settings\<WinUsername>\My Documents\Palm OS Desktop`. If you need the full path of a HotSync user's directory, then call `GetUserDirectory()` and append that path to the `Core\Path` value.

See Also `GetUserDirectory()`, `GetSlotInstallDirectory()` methods

GetRowAdapter

Purpose	Returns a row adapter object for this schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Function` **`GetRowAdapter`**`() As IPSDRowAdapter`
Parameters	None.
Returns	None.
Comments	Use row adapters for reading, writing, and modifying table rows in a schema database.

GetRowCount

Purpose	Retrieves the number of rows in this row set.
Applies to	`PSDRowSet` object.
Prototype	`Function **GetRowCount**() As Long`
Parameters	None.
Returns	The number of rows in this row set.

GetRowCountInTable

Purpose Retrieves the number of rows in a table in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Function GetRowCountInTable(ByVal bExcludeDelete
    As Boolean, ByVal TableName As String) As Long
```

Parameters

→ *`bExcludeDelete`*
If `True`, excludes from the count all rows that are marked as deleted. If `False`, includes all rows.

→ *`TableName`*
The name of the table to get the row count of.

Returns The number of rows in the table.

GetSlotCount

Purpose Retrieves the number of expansion slots on the handheld for the specified user.

Applies to `PDUserData` object.

Prototype `Function GetSlotCount(dwUserId As Long) As Integer`

Parameters → *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

Returns The number of slots.

Errors `eInvalidUser`
`dwUserID` is an invalid number.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErra`
No expansion slot information was previously saved for the specified user.

Comments HotSync Manager retrieves this information from the handheld at the beginning of each HotSync operation and saves it for the corresponding user in the user data store on the desktop. This method simply passes back the saved value. Therefore this value may not be accurate for the next HotSync operation, because the user may have changed or updated the handheld.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `GetSlotDisplayName()` methods

GetSlotDisplayName

Purpose Retrieves the display name for the given slot on the specified user's handheld.

Applies to `PDUserData` object.

Prototype
```
Function GetSlotDisplayName(dwUserId As Long,
    dwSlotId As Long) As String
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`dwSlotId`*
The ID of the slot for which to get the name. To get slot IDs, use the `PDUserData`'s `GetSlotList()` method.

Returns The display name of the specified slot.

Errors

`eInvalidUser`
`dwUserID` is an invalid number.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No expansion slot information was previously saved for the specified user.

`eParamError`
Parameters were not passed correctly.

Comments Use the display name to identify the slot for the user's benefit, not the slot ID.

HotSync Manager assigns names to slots based on their media type at the beginning of each HotSync operation and saves it for the corresponding user in the user information store on the desktop. This method simply passes back the saved information. Therefore it may not be accurate for the next HotSync operation, because the user may have changed or updated the handheld.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `GetSlotCount()` methods

GetSlotFileCount

Purpose Retrieves the number of files queued to install to the specified slot for a given user.

Applies to `PDInstall` object.

Prototype
```
Function GetSlotFileCount (UserID As Long, SlotID
    As Long) As Long
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`SlotID`*
The ID of the slot for which to get the number of queued files. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

Returns The number of queued files.

Errors

`eInvalidPath`
The path of the slot-install directory is longer than 256 characters and cannot be retrieved.

`eParamError`
Parameters were not passed correctly.

Comments This method returns the number of files in the specified user's slot-install directory. See "Installing Files on the Handheld with PDInstall" on page 58 in the *COM Sync Suite Companion*.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()` methods

GetSlotFileSize

Purpose Retrieves the size of the specified file queued to be installed to the handheld's specified expansion slot for a given user.

Applies to `PDInstall` object.

Prototype

```
Function GetSlotFileSize(UserID As Long, SlotID As
    Long, FileName As String) As Long
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`SlotID`*
The ID of the slot for which to get the number of queued files. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

→ *`FileName`*
The name of the file to get the size of.

Returns The size of the specified file in bytes.

Errors

`eInvalidUser`
`UserID` is an invalid number.

`eParamError`
Parameters were not passed correctly.

Comments This method returns the size of a file in the specified user's slot-install directory. See "Installing Files on the Handheld with PDInstall" on page 58 in the *COM Sync Suite Companion*.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `GetSlotFileCount()`, `GetAllQueuedSlotFiles()` methods

GetSlotInfo

Purpose Retrieves information about a specified expansion slot, including the reference number of a mounted volume.

Applies to `PDExpansionManager` object.

Prototype

```
Sub GetSlotInfo(SlotRefNum As Long, bIsCardPresent
    As Boolean, bIsVolumeMounted As Boolean,
    VolRefNum As Long)
```

Parameters

→ *`SlotRefNum`*
: The slot reference number of the slot containing the card to retrieve information about. Call `GetSlotReferenceNumbers()` to get these values.

← *`bIsCardPresent`*
: If `True`, a card is present; if `False`, no card is present.

← *`bIsVolumeMounted`*
: If `True`, a volume is mounted on the card in this slot; if `False`, no volume is mounted on this card.

← *`VolRefNum`*
: If `bIsVolumeMounted` is `True`, this value is its volume reference number. If not, ignore this value. The currently shipping slot driver from PalmSource supports only one volume per slot.

Returns None.

Errors

`eCommunications`
: Communications with the handheld has either not been initialized or has been lost.

`eVFSCardNotPresent`
: No card is present in the given slot.

`eVFSInvalidSlotNumber`
: The slot reference number is not valid.

`eVFSNoSectorReadWrite`
: The card does not support the slot driver block read/write API.

`eVFSSlotDeallocated`
: The slot reference number is within the valid range, but the Expansion Manager has unloaded the slot driver on the handheld.

`eVFSUnsupportedOperation`
Either virtual file systems are not supported on the handheld or the handheld does not have an expansion slot.

Comments Use the volume reference number passed back by this method to specify the volume to act upon when you call other volume-related methods, such as `GetVolumeManager()`.

See Also `GetSlotReferenceNumbers()`, `GetCardInfo()`, `IsExpansionSlotPresent()` methods.

GetSlotInstallDirectory

Purpose Retrieves the slot-install directory name (not the full path) for the specified user and handheld slot.

Applies to `PDUserData` object.

Prototype

```
Function GetSlotInstallDirectory(dwUserId As Long,
    dwSlotId As Long) As String
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`dwSlotId`*
The ID of the slot for which to get the slot-install directory. To get slot IDs, use the `PDUserData`'s `GetSlotList()` method.

Returns The name of the slot-install directory of the specified slot.

Errors

`eInvalidUser`
`dwUserID` is an invalid number.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No expansion slot information was previously saved for the specified user.

`eParamError`
Parameters were not passed correctly.

Comments The slot-install directory is the location on the desktop where the `PDInstall` object places files queued to be installed on the corresponding slot on the handheld for the specified user.

HotSync Manager saves this information at the beginning of each HotSync operation and puts it for the corresponding user in the user information store on the desktop. This method simply returns the saved information. Therefore it may not be accurate for the next HotSync operation because the user may have changed or updated the handheld. If files are queued to be installed to a slot and the slot information changes during the next HotSync operation, an install conduit may ignore those files but does not remove the slot-install directory itself.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()` methods

GetSlotList

Purpose Retrieves a list of all the slot IDs for each of the expansion slots present on the specified user's handheld.

Applies to `PDUserData` object.

Prototype `Function GetSlotList(dwUserId As Long) As Variant`

Parameters

→ *`dwUserId`*
: A unique ID to specify the user to reference in the users data store.

Returns A list of slot IDs as an array of `Long` values.

Errors

`eInvalidUser`
: `dwUserID` is an invalid number.

`eNoUsers`
: The users data store exists, but contains no information.

`eOtherUDErr`
: No expansion slot information was previously saved for the specified user.

`eParamError`
: Parameters were not passed correctly.

Comments The Expansion Manager on the handheld identifies slots by slot reference numbers. These slot reference numbers may change depending on the order in which slot drivers are loaded by the Expansion Manager. Moreover, slot reference numbers are available only to conduits during a HotSync operation. Therefore `PDUserData` uses slot IDs to identify slots instead.

HotSync Manager assigns slot IDs to slots on the handheld at the beginning of each HotSync operation and saves them for the corresponding user in the user information store on the desktop. This method simply returns the saved information. Therefore it may not be accurate for the next HotSync operation because the user may have changed or updated the handheld.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotDisplayName()`, `GetSlotCount()`, `GetSlotMediaType()` methods

GetSlotMediaType

Purpose Retrieves the media type of the given slot on the specified user's handheld.

Applies to `PDUserData` object.

Prototype

```
Function GetSlotMediaType(dwUserId As Long,
    dwSlotId As Long) As EPDSlotMediaType
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`dwSlotId`*
The ID of the slot for which to get the media type. To get slot IDs, use the `PDUserData`'s `GetSlotList()` method.

Returns The media type as one of the `EPDSlotMediaType` values.

Errors

`eInvalidUser`
`dwUserID` is an invalid number.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No expansion slot information was previously saved for the specified user.

`eParamError`
Parameters were not passed correctly.

Comments HotSync Manager retrieves this information from the handheld at the beginning of each HotSync operation and saves it for the corresponding user in the user information store on the desktop. This method simply returns the saved information. Therefore it may not be accurate for the next HotSync operation because the user may have changed or updated the handheld.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()` methods.
`EPDSlotMediaType` constant.

GetSlotReferenceNumbers

Purpose Retrieves a list of slot reference numbers on a handheld.

Applies to `PDExpansionManager` object.

Prototype `Function` **`GetSlotReferenceNumbers`**`() As Variant`

Parameters None.

Returns A list of slot reference numbers as an array of `Long` values.

Errors

`eCommunications`
: Communications with the handheld has either not been initialized or has been lost.

`eParamError`
: Parameters were not passed correctly.

`eVFSCardNotPresent`
: No card is present in the given slot.

`eVFSNoSectorReadWrite`
: The card does not support the slot driver block read/write API.

`eVFSUnsupportedOperation`
: Either virtual file systems are not supported on the handheld or the handheld does not have an expansion slot.

Comments Use the slot reference numbers returned by this method to specify the slot to act upon when you call other slot-related methods, such as `GetCardInfo()` and `GetSlotInfo()`.

See Also `GetCardInfo()`, `GetSlotInfo()` methods.

GetStringData

Purpose Retrieves a `String` configuration entry value for the specified conduit.

Applies to `PDCondMgr`, `PDInstallConduit`, `PDSystemCondMgr` objects.

Prototype `PDCondMgr` and `PDSystemCondMgr`:

```
Function GetStringData(CreatorID As Long,
   StringName As String) As String
```

`PDInstallConduit`:

```
Function GetStringData(UniqueId As Long,
   StringName As String) As String
```

Parameters

→ *`CreatorID`*
If a `PDCondMgr` or `PDSystemCondMgr` object, this parameter is the creator ID of the conduit you want a value for.

→ *`UniqueId`*
If a `PDInstallConduit` object, this parameter is the unique ID of the install conduit you want a value for.

→ *`StringName`*
The name of the string configuration entry you want the value of.

Returns The value of the specified string entry.

Errors

`eInvalidID`
The specified conduit creator ID is not valid.

`eNoSuchConduit`
The specified install conduit does not exist.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

`eValueNotFound`
The specified value could not be found in the configuration entries for this conduit.

Comments This is a general purpose method for retrieving a conduit configuration entry by name. If the conduit you want is a standard synchronization conduit (most are), specify the creator ID in the first parameter. If the conduit you want is an **install conduit**, specify the unique ID in the first parameter.

This method returns information about a conduit that is registered either for the current Windows user or the system, depending on whether it is called for a `PDCondMgr` or a `PDSystemCondMgr` object.

Example

```
Dim CreatorId As Long
Dim strExtra As String
Const strTestValue = "Hello World"

Dim PCondMgr As New PDCondMgr
CreatorId = PCondMgr.StringToCreatorID("memo")

' Set the value for a custom filed called "ExtraString"
Call PCondMgr.SetStringData(CreatorId, "ExtraString", _
   strTestValue)
strExtra = PCondMgr.GetStringData(CreatorId, "ExtraString")
```

See Also `SetStringData()` method

GetStringValue

Purpose Retrieves a string value from a key in the specified user's area of the users data store.

Applies to `PDUserData` object.

Prototype

```
Function GetStringValue(dwUserId As Long, Section
    As String, Key As String) As String
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`Section`*
The section name in the specified user's area of the users data store.

→ *`Key`*
The key of the string to retrieve.

Returns The string value as specified.

Errors

`eInvalidUser`
`dwUserID` is an invalid number.

`ePathBig`
The path or string is more than 256 characters long.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `SetStringValue()`, `DeleteKey()` methods

GetSubDirectoryList

Purpose Retrieves the names of all the subdirectories in a given directory.

Applies to `PDVFSVolumeManager` object.

Prototype
```
Function GetSubDirectoryList(Directory As String,
    DirList As Variant) As Long
```

Parameters

→ *`Directory`*
The full path of the directory to retrieve the subdirectory list from.

← *`DirList`*
A list of all the subdirectories in the specified directory, passed back as an array of `String` values represented as a `Variant`.

Returns The number of subdirectory names passed back in `DirList`.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSBadName`
Invalid path.

`eVFSEnumerationEmpty`
No volumes are present to enumerate or none remain to enumerate.

`eVFSFileBadRef`
The file reference number is invalid: it has been closed or was not obtained from `Open()`.

`eVFSInvalidOperation`
A file system is not present.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotADirectory`
This operation can be performed only on a directory.

`eVFSNotOpen`
: The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
: The volume reference number is invalid because, for example, the volume has not been mounted.

Comments This method retrieves only subdirectory names. Use `GetFileList()` to retrieve filenames.

See Also `GetFileList()`, `Open()` methods.

GetSyncTypeInfo

Purpose Retrieves the synchronization mode of a sync atom for this schema database in the current HotSync operation.

Applies to `PSDDatabaseAdapter` object.

Prototype
```
Function GetSyncTypeInfo(ByVal SyncAtom As
    EPSDSyncAtom, ByVal vChangeContext,
    bDeletesPurged As Boolean) As EPSDSyncType
```

Parameters

→ *`SyncAtom`*
The type of **sync atoms** of which to retrieve the sync type. This is one of the `EPSDSyncAtom` values.

→ *`vChangeContext`*
The **change context** for which the Sync Manager determines the synchronization type for each sync atom. Specifying this value is optional.

← *`bDeletesPurged`*
If `True`, then this parameter indicates that deleted sync atoms have been purged from the handheld since the last HotSync operation with this desktop. If `False`, these records have not been purged.

Returns The type of synchronization that the Sync Manager determines should occur for the specified sync atom. This is one of the `EPSDSyncType` values.

Comments This method determines the sync mode for each sync atom by comparing the change context that the Sync Manager cached on the desktop during the last HotSync operation with the one it obtains from the handheld when the database is opened. Alternatively, you can pass in the change context via the *`vChangeContext`* parameter that you received from a call to `GetChangeContext()` and cached (possibly on a networked server) during a previous HotSync operation.

See Also `GetChangeContext()` method.

GetTableCount

Purpose	Returns the total number of tables in this schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Function` **`GetTableCount`**`() As Long`
Parameters	None.
Returns	The number of tables in this schema database.

GetTableInfo

Purpose	Returns information about a table in this schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Function **GetTableInfo**(ByVal *TableName* As String) As IPSDTable`
Parameters	→ *`TableName`* The name of a table in this schema database.
Returns	A `PSDTable` object.

GetTableNames

Purpose	Retrieves the names of all of the tables in this schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Function` **`GetTableNames`**`(`*`TableNames`*`) As Long`
Parameters	← *`TableNames`* A `Variant` array that contains a list of all of the table names.
Returns	The number of tables in this schema database.

GetUserCount

Purpose Returns the number of users in the users data store.

Applies to `PDUserData` object.

Prototype `Function GetUserCount() As Long`

Parameters None.

Returns The user count.

Errors

`eNoCorePath`
No path for the users data store was found.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

See Also `GetUserList()`, `GetUserNameFromID()` methods

GetUserDirectory

Purpose Retrieves the user directory's name for the specified user ID.

Applies to `PDUserData` object.

Prototype

```
Function GetUserDirectory(dwUserId As Long) As
    String
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

Returns The user directory's name.

Errors

`eInvalidUser`
`dwUserID` is an invalid number.

`eNoCorePath`
No path for the users data store was found.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

Comments To get a complete path, concatenate the result of `GetRootDirectory()` and `GetUserDirectory()`—for example, `root_directory = "C:\Palm\"` and `user_directory = "NUser"`.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetUserList()`, `SetUserDirectory()`, `GetRootDirectory()` methods

GetUserList

Purpose	Retrieves a list of user IDs.
Applies to	`PDUserData` object.
Prototype	`Function` **`GetUserList`**`() As Variant`
Parameters	None.
Returns	A list of user IDs as an array of `Long` values.

Errors	`eNoCorePath`
	No path for the users data store was found.
	`eOtherUDErr`
	No users data store was found or another method or program is accessing the user data store.
	`eParamError`
	Parameters were not passed correctly.
	`eUDSemaphoreError`
	Another method or program is accessing the user data store.

See Also	`GetUserNameFromID()`, `GetUserDirectory()` methods

GetUserNameFromID

Purpose Retrieves a user name in the users data store given a user ID.

Applies to `PDUserData` object.

Prototype `Function` **`GetUserNameFromID`**`(`*`dwUserId`*` As Long) As String`

Parameters → *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

Returns A user name as a `String`.

Errors

`eNoCorePath`
No path for the users data store was found.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `SetUserName()` methods

GetUserPassword

Purpose Retrieves the encrypted user password for the specified user ID.

Applies to `PDUserData` object.

Prototype

```
Function GetUserPassword(dwUserId As Long) As
    String
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

Returns An encrypted password as a `String`.

Errors

`eNoCorePath`
No path for the users data store was found.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()` methods

GetUserPermanentSyncPreferences

Purpose Retrieves a conduit's permanent synchronization preferences for the specified user ID.

Applies to `PDUserData` object.

Prototype

```
Function GetUserPermanantSyncPreferences(dwUserId
    As Long, ConduitCreatorId As Long) As
    EPDUserSyncAction
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`ConduitCreatorId`*
The creator ID of the conduit you want the preferences of.

Returns The user's permanent synchronization preferences as a `EPDUserSyncAction` value.

Errors

`eNoCorePath`
No path for the users data store was found.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `SetUserPermanentSyncPreferences()`, `DeleteUserPermanentSyncPreferences()`, `GetUserTemporarySyncPreferences()`, `SetUserTemporarySyncPreferences()`, `DeleteUserTemporarySyncPreferences()`, `RemoveUserTemporarySyncPreferences()` methods.
`EPDUserSyncAction` constant.

GetUserTemporarySyncPreferences

Purpose Retrieves a conduit's temporary synchronization preferences for the specified user ID.

Applies to `PDUserData` object.

Prototype

```
Function GetUserTemporarySyncPreferences(dwUserId
    As Long, ConduitCreatorId As Long) As
    EPDUserSyncAction
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`ConduitCreatorId`*
The creator ID of the conduit you want the preferences of.

Returns The user's temporary synchronization preferences as a `EPDUserSyncAction` value.

Errors

`eInvalidUser`
`dwUserID` is an invalid number.

`eNoCorePath`
No path for the users data store was found.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `SetUserTemporarySyncPreferences()`, `DeleteUserTemporarySyncPreferences()`, `RemoveUserTemporarySyncPreferences()`, `GetUserPermanentSyncPreferences()`, `SetUserPermanentSyncPreferences()`, `DeleteUserPermanentSyncPreferences()` methods.
`EPDUserSyncAction` constant.

GetVolumeCount

Purpose Retrieves the total number of mounted volumes on cards in all expansion slots.

Applies to `PDVFSManager` object.

Prototype `Function **GetVolumeCount**() As Long`

Parameters None.

Returns The total number of mounted volumes on cards in all expansion slots. A return value of 0 indicates that either no volumes are mounted, no card is in a slot, or the handheld has no slots.

Errors None.

Comments The number of mounted volumes includes those on cards in all expansion slots, if multiple cards are present. To find which card and slot any volume is mounted from, check the `SlotReferenceNumber` property of a `PDVFSVolumeManager` object.

See Also `PDVFSVolumeManager` object.
`GetVolumeReferenceList()`, `GetVolumeManager()` methods.
`SlotReferenceNumber` property.

GetVolumeManager

Purpose Creates a PDVFSVolumeManager object to access a given volume.

Applies to PDVFSManager object.

Prototype

```
Function GetVolumeManager(volRefNo As Long) As
    Unknown
```

Parameters → *volRefNo*

The volume reference number of the volume to access. Call GetVolumeReferenceList() or GetSlotInfo() to get these values.

Returns A PDVFSVolumeManager object.

Errors

`eCommunications`

Communications with the handheld has either not been initialized or has been lost.

`eParamError`

Parameters were not passed correctly.

`eVFSInvalidOperation`

A file system is not present.

See Also PDVFSVolumeManager object.
GetVolumeReferenceList(), GetSlotInfo() methods.

GetVolumeReferenceList

Purpose Retrieves a list of the volume reference numbers of all mounted volumes.

Applies to `PDVFSManager` object.

Prototype

```
Function GetVolumeReferenceList(VolCount As Long)
    As Variant
```

Parameters ← *`VolCount`*

The number of mounted volumes in the retrieved list.

Returns A list of volume reference numbers as an array of `Long` values.

Errors

`eCommunications`

Communications with the handheld has either not been initialized or has been lost.

`eParamError`

Parameters were not passed correctly.

`eVFSInvalidOperation`

A file system is not present.

Comments This method returns a list of reference numbers of all of the volumes that are mounted. The list can span across expansion cards, if multiple cards are present.

NOTE: Volume reference numbers can change each time the handheld mounts a given volume. If you need to keep track of a particular volume from one HotSync operation to the next, save the volume's `Label` property rather than its reference number.

See "Volume Operations" on page 91 in the *COM Sync Suite Companion* for details and an example of checking for the presence of slots, cards, and volumes and then using `GetVolumeReferenceList`.

Example

```
Dim VolumeCount As Long
Dim VFSManager As New PDVFSManager
Dim VFSVolume As PDVFSVolumeManager
Dim VolList() As Integer

' Check whether an expansion slot and a card are present
' first.

' Get a list of volume reference numbers.
VolList = VFSManager.GetVolumeReferenceList(VolumeCount)

' Use the first available volume.
Set VFSVolume = VFSManager.GetVolumeManager(VolList(0))
```

See Also

`PDVFSVolumeManager` object.
`GetVolumeManager()`, `GetVolumeCount()` methods.
`SlotReferenceNumber` property.

HHOsVersion

Purpose Returns the Palm OS® software version.

Applies to `PDSystemAdapter` object.

Prototype

```
Sub HHOsVersion(nVMajor as Integer, nVMinor as
    Integer)
```

Parameters

← *nVMajor*
Major version number.

← *nVMinor*
Minor version number.

Returns None.

Example

```
Dim pSystem as new PDSystemAdapter
Dim VMajor as Integer
Dim VMinor as Integer
PSystem.HHOsVersion(VMajor, VMinor)
```

ImportDatabaseFromFile

Purpose Creates a database from the specified PDB or PRC file on an expansion card. Works only with classic databases.

Applies to `PDVFSVolumeManager` object.

Prototype
```
Function ImportDatabaseFromFile(PathName As
    String, cardNo As Long) As String
```

Parameters

→ *`PathName`*
The full path and filename of the source file from which to create the database.

← *`cardNo`*
The RAM card number on which this method created the database in primary storage. Note that this does not refer to the expansion card and is therefore not related to the slot reference number. The card number for the first RAM memory card on the handheld is 0, which is the only one that most handhelds have.

Returns The `String` name of the database created in primary storage memory on the handheld.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eFileExists`
A database with the specified name already exists in primary storage memory on the handheld.

`eParamError`
Parameters were not passed correctly.

`eVFSBadData`
The operation could not be completed because of invalid data—for example, importing a database from a corrupted PRC file.

`eVFSBadName`
Invalid filename or path.

`eVFSInvalidOperation`
A file system is not present.

Comments This utility method imports a PDB or PRC file on an expansion card into a new database in the handheld storage heap. If the database already exists, this method passes back a value in `cardNo` and returns the name of the existing database and generates an error code of `eFileExists`. This method is the opposite of `ExportDatabaseToFile()`. This method is used, for example, to copy applications from a volume on an expansion card to primary storage memory on a handheld.

IMPORTANT: This method works only with classic databases. It cannot import schema or extended databases.

See Also `ExportDatabaseToFile()`, `Read()` methods.

InstallAndBackupDatabase

Purpose Installs a database on the handheld from an image file on the desktop and then backs up the same database.

Applies to `PSDDatabaseUtilities` object.

Prototype
```
Function InstallAndBackupDatabase(ByVal FilePath
    As String, backupPath As String, bIsInstalled
    As Boolean) As IPSDDatabaseInfo
```

Parameters

→ *`FilePath`*
The full path and filename of the image file to install, as a null-terminated string. Do not pass in a null value.

↔ *`backupPath`*
The destination path or filename of the backup file, as a null-terminated string. If the caller specifies a directory path, then the Sync Manager generates the filename automatically, appends it to the specified directory path, and backs up the database using this full path and filename. If the caller passes in a null value, then the default backup path is used: `<CurrentHotSyncUserFolder>\Backup`. Upon return, this parameter receives the actual path and filename used.

← *`bIsInstalled`*
If `True`, the database was successfully installed; otherwise, the installation failed.

Returns A `PSDDatabaseInfo` object that describes the database that this method installed. The properties of this object are valid only when *`bIsInstalled`* is `True`.

Comments This method installs a Palm OS database on the handheld in the same way that `InstallDatabase()` does and backs up the same database in the same way that `BackupDatabase()` does. However, this method is significantly faster than performing the two operations separately, because the backup operation does not transfer the database back from the handheld; it simply copies the install file to the backup directory.

This method performs the backup operation only when the database backup bit is set *or* the database creator ID does not consist entirely of lowercase letters—that is, the creator ID is not reserved for use by PalmSource, Inc.; in the latter case, the backup bit is automatically set in both the handheld database and the desktop image. When the backup is performed, the backup image's creation, modification, and backup dates are updated with the corresponding handheld values.

This method's generated error along with the output parameters indicate the success or failure of each operation. If this method generates `S_OK`, then *`bIsInstalled`* is `True`. If this method returns `S_OK` and `(PSDDatabaseInfo.Attributes And ePSDBackupDb)` is nonzero, then the database was successfully backed up. If this method generates any other error and *`bIsInstalled`* is `True`, then the error occurred during the backup. Otherwise, *`bIsInstalled`* is `False`, and the method failed during installation; this implies that the properties of the returned `PSDDatabaseInfo` are not updated.

Compatibility Palm OS version: Palm OS Cobalt, version 6.0 or later.

InstallDatabase

Purpose Installs a database image file on the desktop to primary storage on a handheld.

Applies to `PSDDatabaseQuery`, `PSDDatabaseUtilities` objects.

Prototype `Sub InstallDatabase(ByVal FilePath As String)`

Parameters

→ *`FilePath`*
The path and filename of the image file to install as a database, including a null terminator value. Do not pass in only a null value.

Returns None.

Comments With this method, a conduit can write an entire database to the handheld in one call. A conduit can call this method at any time after HotSync Manager calls its `BeginProcess()` entry point and before it returns. Alternatively, before a HotSync operation begins, a desktop application or installer can queue a database or file to be installed by a default install conduit during the next HotSync; see `PDInstall` for details.

This method reads a file on the desktop that is an image of any valid Palm OS database. The Sync Manager validates the file to some extent and then transfers it as a database to memory on the handheld. If a database with the same name and creator ID in the relevant namespace already exists on the handheld, this method deletes it and writes a new database from the image file.

If the database creator ID does not consist of all lowercase letters—that is, the creator ID is not reserved for use by PalmSource, Inc.—this method automatically sets the database's backup bit.

InstallFileToHH

Purpose Queues a file to be installed in primary storage on a user's handheld.

Applies to `PDInstall` object.

Prototype
```
Sub InstallFileToHH(UserID As Long, FileName As
    String)
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`FileName`*
The name of the file to install.

Returns None.

Errors

`eInvalidUser`
`UserID` is an invalid number.

`eOtherError`
An unspecified error occurred.

`eParamError`
Parameters were not passed correctly.

Comments This method copies the file specified by `FileName` into the handheld-install directory for the user name specified by `UserID`, and then sets an "Install" configuration entry to specify which install conduit that HotSync Manager must run during the next synchronization operation to install the specified file.

IMPORTANT: When calling `InstallFileToHH`, you must specify a file of a type that is supported for installation as a database in the handheld's primary storage. If you specify an unsupported file type, `InstallFileToHH` does not generate an error and the file is not installed during the next HotSync operation. To avoid this problem, check the extension of the filename before you call this method to install it. If the extension is not one of the Palm OS® platform's standard extensions (`.prc`, `.pdb`, `.pqa`, `.pnc`, or `.scp`), do not call this method to install it. However, note that all file types may be installed to an expansion card with `InstallFileToSlot()`.

Example

```
Dim PInstall As New PDInstall
Dim UserData As New PDUserData
Dim UserId As Long

' Retrieve the user ID from the HotSync Manager user name.
UserId = UserData.GetIDFromName("Palm OS Emulator")

Call PInstall.InstallFileToHH(UserId, "c:\temp\MyApp.prc")
```

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `RemoveFileFromHHQueue()`, `InstallFileToSlot()` methods

InstallFileToSlot

Purpose Queues a file to be installed in secondary storage in an expansion slot of a user's handheld.

Applies to `PDInstall` object.

Prototype

```
Sub InstallFileToSlot(UserID As Long, SlotID As
    Long, File As String)
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`SlotID`*
The ID of the slot to install the file to. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

→ *`File`*
The name of the file to install.

Returns None.

Errors

`eInvalidUser`
`UserID` is an invalid number.

`eParamError`
Parameters were not passed correctly.

Comments This method copies the file specified by `File` into the slot-install directory specified by `UserID` and `SlotID`, and then modifies the conduit configuration entries to notify HotSync Manager that it needs to install the file during the next HotSync operation. This method accepts all file types.

Example

```
Dim PInstall As New PDInstall
Dim UserData As New PDUserData
Dim UserId As Long

' Retrieve the user ID from the HotSync Manager user name.
UserId = UserData.GetIDFromName("Palm OS Emulator")

Call PInstall.InstallFileToSlot(UserId, 0, _
   "c:\temp\MyApp.prc")
```

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `RemoveFileFromSlotQueue()`, `InstallFileToHH()` methods

IsArchived

Purpose	Determines whether this row is marked for archiving.
Applies to	`PSDRowData` object.
Prototype	`Function` **`IsArchived`**`() As Boolean`
Parameters	None.
Returns	`True`, if this row is marked to be archived; `False`, if not.

IsDatabaseBackupNeeded

Purpose Determines whether the desktop backup file for a database on the handheld is out-of-date.

Applies to `PSDDatabaseUtilities` object.

Prototype

```
Function IsDatabaseBackupNeeded(ByVal DBName As
    String, ByVal vCreatorID, ByVal vType,
    FilePath As String, ByVal Attribute As
    EPSDDBAttribute, HHDBInfo As IPSDDatabaseInfo,
    DTDBInfo As IPSDDatabaseInfo, bDbExists As
    Boolean, bFileExists As Boolean) As Boolean
```

Parameters

→ *`DBName`*
The **database name** as a null-terminated string. Do not pass in a null value. See `PSDDatabaseInfo`.`Name`.

→ *`vCreatorID`*
The creator ID of the database. See `PSDDatabaseInfo`.`CreatorID`.

→ *`vType`*
The database type. See `PSDDatabaseInfo`.`Type`.

↔ *`FilePath`*
The desktop filename or directory of the corresponding backup file to test. Do not pass in a null value. If this parameter specifies a directory that exists and the handheld database exists, then this parameter receives the full path with the automatically generated filename appended.

→ *`Attributes`*
A `EPSDDBAttribute` enum value that specifies whether the database is a schema, extended, or classic database.

← *`HHDBInfo`*
A `PSDDatabaseInfo` object that receives values for the following properties if the database exists on the *handheld:* `Attributes`, `BackupDate`, `CreationDate`, `CreatorID`, `Flags`, `IsReadOnlyDatabase`, `ModifyDate`, `ModifyNumber`, `Name`, `Type`, and `Version`. Note that to generate the backup filename if `FilePath` is a directory, this method uses the `Type` and `Attributes` that it passes *back*.

← *`DTDBInfo`*

A `PSDDatabaseInfo` object that receives values for the following properties if the specified image file on the *desktop* exists: `Attributes`, `BackupDate`, `CreationDate`, `CreatorID`, `Flags`, `IsReadOnlyDatabase`, `ModifyDate`, `ModifyNumber`, `Name`, `Type`, and `Version`. .

← *`bDbExists`*

If `True`, then the specified database exists on the handheld. If `False`, it does not exist.

← *`bFileExists`*

If `True`, then the specified backup file exists on the desktop. If `False`, it does not exist.

Returns `True`, if the specified backup file is out-of-date compared to the specified database on the handheld. `False`, if the file is not out-of-date as defined in the “Comments” section.

Comments This method determines whether a desktop backup file exists for, or is older than, a corresponding database on the handheld. A conduit can call this method at any time after HotSync Manager calls its `BeginProcess()` entry point and before it returns. Note that the Sync Manager can back up secure databases only to trusted desktops.

This method considers a backup file on the desktop to be out-of-date (and therefore returns `True`) only if *all* of the following statements are true:

- The specified database is present on the handheld.
- The database’s backup bit is set.
- One or more of the following is true:
  - The backup file does not exist on the desktop.
  - The backup file exists, but it is not a backup file for the specified database, either because it isn’t of the same type or it doesn’t have the same database name and creator ID.
  - The handheld and desktop creation dates differ.
  - The handheld and desktop modification dates differ.
  - The handheld last backup date is zero.

NOTE: If the database's backup bit is not set, this function always returns `False`.

Consider when the *`FilePath`* parameter specifies a directory that exists on the desktop and the *`HHDBInfo`* parameter specifies a database that exists on the handheld. In this case, the Sync Manager generates the filename of the backup file based on information in the handheld database header. When this method returns, it appends the filename to the original directory path and passes back this full path and filename via *`FilePath`*.

However, consider when the *`FilePath`* parameter specifies a directory that does *not* exist on the desktop and the specified database exists on the handheld. In this case, the Sync Manager assumes that the last part of the path is the filename of the backup file. When this method returns, it does *not* change *`FilePath`* by appending a generated filename.

Compatibility Palm OS version: Palm OS Cobalt, version 6.0 or later.

IsDataModified

Purpose	Determines whether this row contains column data that is marked as modified since the last HotSync operation.
Applies to	`PSDRowData` object.
Prototype	`Function **IsDataModified**() As Boolean`
Parameters	None.
Returns	`True`, if this row contains column data that has been modified since the last HotSync operation; `False`, if not.
Comments	This method does not indicate whether the row's category memberships have changed. To determine that, call `IsMembershipModified()` instead.

IsDeleted

Purpose	Determines whether this row has been marked as deleted.
Applies to	`PSDRowData` object.
Prototype	`Function IsDeleted() As Boolean`
Parameters	None.
Returns	`True`, if this row has been deleted on the handheld since the last HotSync operation; `False`, if not.

IsDirty

Purpose Determines whether a category has been modified since the last HotSync operation.

Applies to `PSDCategoryAdapter` object.

Prototype

```
Function IsDirty(ByVal CategoryID As Long) As
    Boolean
```

Parameters → *`CategoryID`*
Specifies the category ID of the category to query.

Returns `True`, if the specified category has been modified; `False`, if not.

IsExpansionSlotPresent

Purpose Verifies the presence of an expansion slot on the handheld.

Applies to `PDExpansionManager` object.

Prototype `Function` **`IsExpansionSlotPresent`**`() As Boolean`

Parameters None.

Returns `True` if an expansion slot is present, `False` if not.

Errors None.

Comments The information returned by this method has already been obtained by the desktop VFS Manager, so no additional calls are made to the handheld at the time you call this method.

This method determines only whether the optional Expansion Manager and VFS Manager system extensions are present on the handheld. From this you can infer that an expansion slot is present, because no handheld ships with these extensions unless it has a slot. Then to confirm whether the slot has a card in it, you must use the `GetSlotInfo()` method.

NOTE: PalmSource recommends that you call this method to confirm that a slot is present before calling other expansion-related methods. Errors generated by other methods may not clearly indicate that they failed because Expansion Manager and VFS Manager are not present on the handheld.

See Also `GetSlotInfo()`, `GetSlotReferenceNumbers()` methods.

IsMembershipModified

Purpose	Determines whether this row's category memberships have been modified.
Applies to	`PSDRowData` object.
Prototype	`Function` **`IsMembershipModified`**`() As Boolean`
Parameters	None.
Returns	`True`, if this row has been added or deleted from one or more categories since the last HotSync operation; `False`, if not.
Comments	Row category membership is not modified by a change in the category's name.

IsProfileUser

Purpose Determines whether an account is a **user profile**.

Applies to `PDUserData` object.

Prototype

```
Function IsProfileUser(dwUserId As Long) As
    Boolean
```

Parameters → *`dwUserId`*

A unique ID to specify the user to reference in the users data store.

Returns A `Boolean` value: if `True`, this account is a user profile; if `False`, it is a regular user account.

Errors

`eNoCorePath`
No path for the users data store was found.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

Comments For more information on user profiles, see the glossary in the *Introduction to Conduit Development*.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `AddNewUser()` methods

IsRowInCategory

Purpose Determines whether a row belongs to a set of categories.

Applies to `PSDRowAdapter` object.

Prototype

```
Function IsRowInCategory(ByVal vRowID,
    vCategoryIDList, ByVal MatchMode As
    EPSDMatchMode) As Boolean
```

Parameters

→ *`vRowID`*
The row ID of the row to check the category memberships of.

→ *`vCategoryIDList`*
The set of categories to check against. This is an array of category IDs.

→ *`MatchMode`*
The category match mode that this method uses to match the specified category ID list against the row's category memberships. Specify one of the `EPSDMatchMode` values.

Returns `True`, if the row is a member of the specified set of categories according to the match mode. `False`, if it is not.

IsSyncInProgress

Purpose Determines whether the HotSync Manager application is currently busy synchronizing a handheld.

Applies to `PDHotSyncUtility` object.

Prototype `Function IsSyncInProgress() As Boolean`

Parameters None.

Returns The status of HotSync Manager:

- If `True`, HotSync Manager is performing a HotSync operation.
- If `False`, HotSync Manager is idle.

Errors `eHotSyncNotFound`
HotSync Manager is not running.

See Also `StartHotSyncMgr()`, `RestartHotSyncMgr()`, `RefreshConduitInfo()`, `TerminateHotSyncMgr()` methods

IsVolumeAvailable

Purpose	Determines whether there is a volume available on the handheld.
Applies to	`PDVFSManager` object.
Prototype	`Function` **`IsVolumeAvailable`**`() As Boolean`
Parameters	None.
Returns	`True` if a mounted volume is present, `False` if not.
Errors	None.
See Also	`PDVFSVolumeManager` object. `GetVolumeReferenceList()`, `GetVolumeManager()`, `GetVolumeCount()` methods.

LaunchCustomDlg

Purpose Displays the **Custom** dialog box of the HotSync Manager application.

Applies to `PDHotSyncUtility` object.

Prototype `Sub` **`LaunchCustomDlg`**`();`

Parameters None.

Returns None.

Errors `eHotSyncNotFound`
HotSync Manager is not running.

Comments The **Custom** dialog box enables the user to view or change the synchronization preferences of each conduit—for example, Synchronize, Desktop overwrites handheld, and so on.

See Also `StartHotSyncMgr()`, `LaunchFileLinkDlg()`, `LaunchSetupDlg()`, methods

LaunchFileLinkDlg

Purpose *(Deprecated)* Displays the File Link wizard of the HotSync Manager application.

Applies to `PDHotSyncUtility` object.

Prototype `Sub` **`LaunchFileLinkDlg`**`(`*`dwUserId`*` As Long)`

Parameters → *`dwUserId`*
A unique ID to specify the user for whom you want to set up a file link. If this value is 0, the current user's file link information is displayed.

Returns None.

Errors `eHotSyncNotFound`
HotSync Manager is not running.

Comments The File Link wizard enables the user to create or modify a **file link**.

Compatibility This method is deprecated because the file link feature has been removed from HotSync Manager versions 6.0.1 and later.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `StartHotSyncMgr()`, `LaunchCustomDlg()`, `LaunchSetupDlg()`, methods

LaunchSetupDlg

Purpose	Displays the **Setup** dialog box of the HotSync Manager application.
Applies to	`PDHotSyncUtility` object.
Prototype	`Sub` **`LaunchSetupDlg`**`()`
Parameters	None.
Returns	None.
Errors	`eHotSyncNotFound` HotSync Manager is not running.
Comments	The **Setup** dialog box enables the user to select a serial port, configure a modem, and set up network HotSync operation for each user.
See Also	`StartHotSyncMgr()`, `LaunchCustomDlg()`, `LaunchFileLinkDlg()` methods

ModifyNotifier

Purpose Modifies the path or filename of a notifier already registered with HotSync Manager.

Applies to `PDCondMgr` object.

Prototype

```
Sub ModifyNotifier(OriginalPath As String, NewPath
    As String)
```

Parameters

→ *`OriginalPath`*
The full path or filename of a registered notifier.

→ *`NewPath`*
The full path or filename you want to change this notifier to.

Returns None.

Errors

`eNotifierNotFound`
The `OriginalPath` notifier name was not found in the list of registered notifiers.

`eParamError`
Parameters were not passed correctly.

`ePathBig`
The path specified by `NewPath` is longer than 256 characters.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments You can use this method to specify a new location for a Notifier.

IMPORTANT: Do not set `NewPath` to `Null` as a way of unregistering a notifier. Use the `UnregisterNotifier()` method instead.

Example

```
Dim PDCondMgr As New PDCondMgr

Call PDCondMgr.RegisterNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll")
Call PDCondMgr.ModifyNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll", "C:\PdN20d.dll")
Call PDCondMgr.UnregisterNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll")
```

See Also

`GetNotifierList()`, `RegisterNotifier()`, `UnregisterNotifier()` methods

ModifyRow

Purpose Writes an entire row—attributes, category memberships, and column values—to a schema database on the handheld.

Applies to `PSDRowAdapter` object.

Prototype

```
Sub ModifyRow(ByVal RowID, PSDRowData As
    IPSDRowData)
```

Parameters

→ *`RowID`*
The row ID of the row to write.

→ *`PSDRowData`*
A `PSDRowData` object that specifies the row's data to write.

Returns None.

MoveFirst

Purpose	Moves the cursor to the first row in this set and returns an object representing the first row.
Applies to	`PSDRowSet` object.
Prototype	`Function **MoveFirst**() As IPSDRowData`
Parameters	None.
Returns	A `PSDRowData` object that represents the first row in this set.

MoveLast

Purpose	Moves the cursor to the last row in this set and returns an object representing the last row.
Applies to	`PSDRowSet` object.
Prototype	`Function MoveLast() As IPSDRowData`
Parameters	None.
Returns	A `PSDRowData` object that represents the last row in this set.

MoveNext

Purpose	Moves the cursor to the next row in this set and returns an object representing this row.
Applies to	`PSDRowSet` object.
Prototype	`Function` **`MoveNext`**`() As IPSDRowData`
Parameters	None.
Returns	A `PSDRowData` object that represents the next row in this set.

MovePrevious

Purpose	Moves the cursor to the previous row in this set and returns an object representing this row.
Applies to	`PSDRowSet` object.
Prototype	`Function` **`MovePrevious`**`() As IPSDRowData`
Parameters	None.
Returns	A `PSDRowData` object that represents the previous row in this set.

MoveRowsToCategory

Purpose Moves all of the rows that belong to a specified set of categories into another category.

Applies to `PSDDatabaseAdapter` object.

Prototype

```
Sub MoveRowsToCategory(vCategoryIDList, ByVal
    MatchMode As EPSDMatchMode, ByVal
    TargetCategoryID As Long)
```

Parameters

→ *`vCategoryIDList`*
A `Variant` array of category IDs to match.

→ *`MatchMode`*
The category match mode that this method uses to match the specified category ID list against rows’ category memberships. Specify one of the `EPSDMatchMode` values.

→ *`TargetCategoryID`*
The category ID of the category to move all matching rows into.

Returns None.

Comments All matching rows lose their existing category memberships and are moved into only the *`TargetCategoryID`* category.

MoveTo

Purpose Moves the cursor to the specified row in this set and returns an object representing this row.

Applies to `PSDRowSet` object.

Prototype `Function` **`MoveTo`**`(ByVal` *`nRow`* `As Long) As IPSDRowData`

Parameters → *`nRow`*

A zero-based index of a row in this row set. Call `GetRowCount()` to determine the valid index range.

Returns A `PSDRowData` object that represents the specified row in this set.

Open

Purpose Opens a file or directory on an expansion card and returns a `PDVFSFileManager` object.

Applies to `PDVFSVolumeManager` object.

Prototype

```
Function Open(PathName As String, openMode As
    EPDVFSFileOpenAttr) As Unknown
```

Parameters

→ *`PathName`*
: The full path and filename (or only the path) of the file (or directory) to create. All parts of the path, including the filename, must already exist. This parameter cannot be empty and cannot contain `Null` characters. The format of the path should match what the underlying file system supports. See "Directory Paths" on page 100 in the *COM Sync Suite Companion* for a description of how to construct a valid path.

→ *`openMode`*
: One of the `EPDVFSFileOpenAttr` constants to specify the mode to use when opening the file or directory. See "EPDVFSFileOpenAttr" on page 553 for a list of accepted modes.

Returns A `PDVFSFileManager` object representing the open file or directory.

Errors

`eCommunications`
: Communications with the handheld has either not been initialized or has been lost.

`eParamError`
: Parameters were not passed correctly.

`eVFSBadName`
: Invalid filename or path.

`eVFSFileNotFound`
: The file was not found in the specified path.

`eVFSFilePermissionDenied`
: Permission denied to perform requested operation—for example, an attempt to write to a read-only file or to read a file already opened in the `eVFSModeExclusive` mode.

`eVFSInvalidOperation`
: A file system is not present.

`eVFSNotOpen`
: The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
: The volume reference number is invalid because, for example, the volume has not been mounted.

Comments The `PDVFSFileManager` object obtained for a directory cannot be used for all methods. For example, it is not permitted (or logical) to read directly from an opened directory.

See Also `PDVFSFileManager` object.
`Close()`, `Read()`, `Write()`, `Tell()`, `Seek()`, `GetFileList()` methods.

OpenDatabase

Purpose Opens a schema database.

Applies to `PSDDatabaseQuery` object.

Prototype

```
Function OpenDatabase(ByVal DatabaseName As
    String, ByVal vCreatorID, [ByVal openMode As
    EPSDOpenMode], [ByVal shareMode As
    EPSDShareMode = EPSDShareNone]) As
    IPSDDatabaseAdapter
```

Parameters

→ *`DatabaseName`*
The **database name** as a null-terminated string. Do not pass in a null value. See `PSDDatabaseInfo`.`Name`.

→ *`vCreatorID`*
Creator ID of the database. See `PSDDatabaseInfo`.`CreatorID`.

→ *`openMode`*
The access mode in which to open the database—read-only, read/write, and show private records. Specify a nonexclusive combination of the `EPSDOpenMode` values.

→ *`shareMode`*
The share mode in which to open the database—share for read-only access or do not share. Specify one of the `EPSDShareMode` values.

Returns A `PSDDatabaseAdapter` object representing the open schema database.

Comments Note that in the *`openMode`* parameter, you cannot specify `ePSDShowSecret` alone; you must also specify one of the other values: `ePSDReadOnly` or `ePSDReadWrite`.

OpenRecordDatabase

Purpose Opens a classic or extended record database on the handheld.

Applies to `DmDatabaseQuery`, `PDDatabaseQuery` objects.

Prototype `DmDatabaseQuery`:

```
Function OpenRecordDatabase(ByVal pDbName As
    String, ByVal vCreator, ByVal pAdapterName As
    String, [ByVal eAccessMode As EAccessModes])
    As IUnknown
```

`PDDatabaseQuery`:

```
Function OpenRecordDatabase(ByVal pDbName As
    String, ByVal pAdapterName As String, [ByVal
    eAccessMode As EAccessModes]) As IUnknown
```

Parameters

→ *`pDbName`*
Name of database to open (case sensitive, 1-31 characters).

→ *`vCreator`*
(`DmDatabaseQuery` only) Creator ID of the database as a `Variant`—for example, `'adrs'`. Unlike classic databases, extended databases must be specified by both name and creator ID.

→ *`pAdapterName`*
Full name of the COM Sync database adapter to use. Names are of this form:

```
LibraryName.AdapterName
```

Do not include “Lib” in the name—for example, use `PDDirect.PDRecordAdapter`, not `PDDirectLib.PDRecordAdapter`.

→ *`eAccessMode`*
Access modes from the `EAccessModes` constants.

Returns A database adapter object of the type you specify in the *`pAdapterName`* parameter. Possible returned objects include `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDMemoDbHHRecordAdapter`, and `PDTodoDbHHRecordAdapter` objects.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", _
   "PDDirect.PDRecordAdapter")
```

See Also

`DmRecordAdapter`, `DmDatabaseQuery`, `PDRecordAdapter`, `PDDatabaseQuery` objects.
`CreateRecordDatabase()` method.
`EAccessModes` constants.

OpenResourceDatabase

Purpose Opens a resource database on the handheld.

Applies to `PDDatabaseQuery` object.

Prototype

```
Function OpenResourceDatabase(pDbName as String,
    pAdapterName as String, [nAccessMode as
    EAccessModes = eRead Or eWrite Or
    eShowSecret]) as PDResourceAdapter
```

Parameters

→ *`pDbName`*
Name of database to open.

→ *`pAdapterName`*
ProgID of the database adapter to use. Do not include “Lib” in the ProgID—for example, use `PDDirect.PDResourceAdapter`, not `PDDirectLib.PDResourceAdapter`.

→ *`nAccessMode`*
Access modes from the `EAccessModes` constants.

Returns A `PDResourceAdapter` object.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDResourceAdapter
Set Adapter = DbQuery.OpenResourceDatabase("Saved _
   Preferences", "PDDirect.PDResourceAdapter")
```

See Also `PDResourceAdapter` object
`EAccessModes` constants

PurgeAllRowsInTable

Purpose	Removes all the rows that are marked as deleted in a table in a schema database.
Applies to	`PSDRowAdapter` object.
Prototype	`Sub` **`PurgeAllRowsInTable`**`(ByVal` *`TableName`* `As String)`
Parameters	→ *`TableName`* The name of the table to purge. Specify a null value to purge all rows in all tables in the database.
Returns	None.

Read

Purpose Reads data from a file on an expansion card into the specified buffer.

Applies to `PDVFSFileManager` object.

Prototype

```
Function Read(numBytes As Long, Buffer As Variant)
    As Long
```

Parameters

→ *`numBytes`*
The number of bytes to read.

← *`Buffer`*
A `Variant` to receive the array of bytes to be read.

Returns The number of bytes (as a `Long`) that were actually read.

Errors

`eParamError`
The `Buffer` or `numBytes` parameter is `Null`.

`eVFSFileBadRef`
The file reference number is invalid: it has been closed or was not obtained from `Open()`.

`eVFSFilePermissionDenied`
Permission denied to perform requested operation.

`eVFSInvalidOperation`
A file system is not present.

`eVFSIsADirectory`
This operation can be performed only on a regular file, not a directory.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
The volume reference number is invalid because, for example, the volume has not been mounted.

Comments This method operates only on files and cannot be used with directories; use `GetSubDirectoryList()` and `GetFileList()` to explore the contents of a directory.

See Also `Open()`, `Tell()`, `Seek()`, `Write()`, `ImportDatabaseFromFile()` methods.

ReadAppInfoBlock

Purpose Reads this database's application info block.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype `Function` **`ReadAppInfoBlock`** `() as Variant`

Returns A `Byte` array containing the application info block.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Read the AppInfo block
Dim vAppInfo as Variant
vAppInfo = Adapter.ReadAppInfoBlock
```

ReadAppPreference

Purpose Reads an application's preference block.

Applies to `PDSystemAdapter` object.

Prototype

```
Function ReadAppPreference(vCreator as Variant,
   nId as Long, bBackup as Boolean, nVersion as
   Integer) as Variant
```

Parameters

→ *`vCreator`*
Creator ID. The unique ID associated with each database and application on the device.

→ *`nId`*
Preference ID.

→ *`bBackup`*
Saved or unsaved preference. When `True`, this method will read from the Saved Preferences database. When `False`, this method will read from the Unsaved Preferences database.

← *`nVersion`*
Preference version.

Returns A `Byte` array containing the application preference data.

Example

```
Dim pSystem as New PDSystemAdapter
Dim vAppPref as Variant
Dim Version as Integer
vAppPref = PSystem.ReadAppPreference("mail", 1, True, _
   Version)
```

See Also `WriteAppPreference()` method.

ReadBackupImageInfo

Purpose Reads the database header information from a backup image file on the desktop.

Applies to `PSDDatabaseQuery`, `PSDDatabaseUtilities` objects.

Prototype

```
Function ReadBackupImageInfo(ByVal FilePath As
    String) As IPSDDatabaseInfo
```

Parameters

→ *`FilePath`*
The path and filename of the backup image file, as a null-terminated string. Do not pass in a null value.

Returns A `PSDDatabaseInfo` object that contains information about the backup image file.

Comments This method can be called outside of a HotSync operation.

ReadById

Purpose Reads a record using its unique ID.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype For `DmRecordAdapter` and `PDRecordAdapter` objects:

```
Function ReadById(nIndex As Long, ByVal vUniqueId,
   nCategory As Long, eAttributes As
   ERecordAttributes) As Variant
```

For `PD<PIM>DbHHRecordAdapter` objects:

```
Function ReadById(vUniqueId as Variant) as Unknown
```

Parameters

← *`nIndex`*
Index of returned record.

→ *`vUniqueId`*
The unique ID for the record.

← *`nCategory`*
Category ID of returned record.

← *`eAttributes`*
Returned record attributes.

Returns For a `DmRecordAdapter` or `PDRecordAdapter` object, returns a `Byte` array containing the value of the record specified by *`vUniqueId`*.

For any of the objects representing classic databases used by any of four standard Palm OS applications (denoted `PD<PIM>DbHHRecordAdapter`), this method returns a corresponding object (denoted `PD<PIM>DbHHRecord`) representing the record specified by *`vUniqueId`*.

Example

```
Dim pUtil as New PDUtility
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Read a record
Dim vUniqueId as Variant
Dim StrID as String
StrId = 12345
vUniqueID = pUtil.StringtoRecordId (StrID)
Dim Index as Long
Dim Category as Long
Dim Attributes ERecordAttributes
Dim RecordData as Variant
RecordData = Adapter.ReadById(Index, vUniqueId, Category, _
   Attributes)
```

See Also `ERecordAttributes` constants.

ReadByIndex

Purpose Reads a record using its index.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype For `DmRecordAdapter` and `PDRecordAdapter` objects:

```
Function ReadByIndex(ByVal nIndex As Long,
    pvUniqueId, nCategory As Long, eAttributes As
    ERecordAttributes) As Variant
```

For `PD<PIM>DbHHRecordAdapter` objects:

```
Function ReadByIndex(nIndex as Long) as Unknown
```

Parameters

→ *`nIndex`*
Record index.

← *`pvUniqueId`*
Returned record ID.

← *`nCategory`*
Category of returned record.

← *`eAttributes`*
Returned attributes from the `ERecordAttributes` constants.

Returns For a `DmRecordAdapter` or `PDRecordAdapter` object, returns a `Byte` array containing the record data.

For any of the objects representing classic databases used by any of four standard Palm OS applications (denoted `PD<PIM>DbHHRecordAdapter`), this method returns a corresponding object (denoted `PD<PIM>DbHHRecord`) representing the record specified by `nIndex`.

Example

```
Dim pUtil as New PDUtility
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Read a record
Dim UniqueId as Variant
Dim Category as Byte
Dim Attributes as ERecordAttributes
Dim RecordData as Variant
RecordData = Adapter.ReadByIndex(2, UniqueId, Category, _
   Attributes)
Dim StrUniqueId as String
StrUniqueID = pUtil.RecordIdtoString(UniqueID)
```

See Also

`ERecordAttributes` constants.

ReadColumnValue

Purpose Reads the specified bytes of a column value from a row in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Function ReadColumnValue(ByVal vRowID, ByVal
    ColumnID As Long, ByVal DataOffset As Long,
    ByVal BytesToRead As Long, BytesRemaining As
    Long, vData) As Long
```

Parameters

→ *`vRowID`*
The row ID of the row to read.

→ *`ColumnID`*
The column ID of the column to read the value of.

→ *`DataOffset`*
An offset from the first byte in a column value from which to start retrieving data.

→ *`BytesToRead`*
The number of bytes of a column value to retrieve starting from the *`dataOffset`* position.

← *`BytesRemaining`*
The number of bytes of the column value that remain—that is, the number of bytes after the last one read (*`DataOffset`* + *`BytesToRead`*) to the end of the column value.

← *`vData`*
A `Variant` byte array that contains the specified bytes of the column value.

Returns An offset to the next unread byte.

Comments For example, if there are 25 bytes in a column and you want to read 5 bytes (*`BytesToRead`*) starting from an offset of 5 (*`DataOffset`*), then the return value is 10 bytes and the number of unread bytes is 15 (*`BytesRemaining`*).

ReadColumnValues

Purpose Reads the specified column values from a row in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Function ReadColumnValues(ByVal vRowID,
    vColumnIDList) As IPSDRowData
```

Parameters

→ *`vRowID`*
The row ID of the row to read.

→ *`vColumnIDList`*
Specifies a `Variant` array of column IDs of the column values to read.

Returns A `PSDRowData` object with only the `Value` property filled in for each column specified in the *`vColumnIDList`* parameter.

ReadDatabaseInfoByName

Purpose Retrieves information about a database given its name, creator ID, and type.

Applies to `PSDDatabaseQuery` object.

Prototype

```
Function ReadDatabaseInfoByName(ByVal DatabaseName
    As String, ByVal vCreatorID, ByVal vType) As
    IPSDDatabaseInfo
```

Parameters

→ *`DatabaseName`*
The **database name** as a null-terminated string. Do not pass in a null value. See `PSDDatabaseInfo`.`Name`.

→ *`vCreatorID`*
Creator ID of the database. See `PSDDatabaseInfo`.`CreatorID`.

→ *`vType`*
The database type. See `PSDDatabaseInfo`.`Type`.

Returns A `PSDDatabaseInfo` object that contains information about the database.

Comments This method returns information about unopened databases, so the following properties of the returned `PSDDatabaseInfo` object are not set:

- `DisplayName`
- `Encoding`
- `TableCount`

ReadDatabaseInfoByNameCreator

Purpose Returns a `DmDatabaseInfo` object for an extended database specified by name and creator ID.

Applies to `DmDatabaseQuery` object.

Prototype

```
Function ReadDatabaseInfoByNameCreator(ByVal
    DatabaseName As String, ByVal vCreatorID) As
    IDmDatabaseInfo
```

Parameters

→ *`DatabaseName`*
Database name.

→ *`vCreatorID`*
Creator ID. The unique ID associated with each application and its associated databases on the device.

Returns A `DmDatabaseInfo` object that describes the specified database.

Example

```
Dim DbQuery as New DmDatabaseQuery
Dim DbInfo as DmDatabaseInfo
Set DbInfo = DbQuery.ReadDatabaseInfoByNameCreator _
   ("MyDatabase", "MyCr")
```

See Also `DmDatabaseInfo` object.

ReadDatabaseNameList

Purpose Returns the names of all databases on the handheld that match the specified creator ID and type.

Applies to `PSDDatabaseQuery` object.

Prototype

```
Function ReadDatabaseNameList(ByVal vCreatorID,
    ByVal vType) As Variant
```

Parameters

→ *vCreatorID*
Creator ID of the database as a `Variant`—for example, `'adrs'`. See `PSDDatabaseInfo`.`CreatorID`.

→ *vType*
The database type as a `Variant`—for example, `'DATA'`. See `PSDDatabaseInfo`.`Type`.

Returns A `Variant` array that lists the names of all the matching databases.

Comments This method returns a list of all databases—classic, extended, and schema—that match the specified creator ID and type.

ReadDatabaseNameList

Purpose Returns a list of **non-schema database** names that are either in RAM or ROM on the handheld.

Applies to `DmDatabaseQuery` objects.

Prototype

```
Function ReadDatabaseNameList(ByVal bRam As
    Boolean) As Variant
```

Parameters → *bRam*

RAM or ROM. When `True`, specifies RAM. When `False`, specifies ROM.

Returns A `String` array containing the database names.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim DbNames as Variant
' Read the RAM name list
DbNames = DbQuery.ReadDatabaseNameList(True)
```

ReadDbInfoByCreatorType

Purpose Returns a `PDDatabaseInfo` object for a creator/type pair.

Applies to `PDDatabaseQuery` object.

Prototype

```
Function ReadDbInfoByCreatorType(vCreator as
    Variant, vDbType as Variant, bFirstQuery as
    Boolean) as PDDatabaseInfo
```

Parameters

→ *`vCreator`*
Creator ID. The unique ID associated with each database and application on the device. Each conduit is associated with a specific creator ID.

→ *`vDbType`*
Database Type, 4 characters that can be in either Long (VT_I4) or Little Endian form. BSTR (VT_BSTR), first 4 characters used.

→ *`bFirstQuery`*
First or subsequent queries. When `True`, this is the first query, starting at the beginning of the database list. When `False`, continue from wherever you are.

Returns A `PDDatabaseInfo` object.

Comments `ReadDbInfoByCreatorType` iterates through the database list returning all databases of a given creator, type, or both. Creator or type may be zero or an empty string. In this case, every database of a given creator or type is returned.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim DbInfo as PDDatabaseInfo
' Find all the applications
Dim bFirst As Boolean, bLoop as Boolean
bFirst = True
bLoop = True
Do While bLoop
   On Error Resume Next
   Set pDbInfo = _
      pDbQuery.ReadDbInfoByCreatorType(0, "appl", bFirst)
   bFirst = False
   If TypeName(pDbInfo) = "Nothing" Then bLoop= False
Loop
```

See Also `PDDatabaseInfo` object.

ReadDbInfoByName

Purpose Returns a `PDDatabaseInfo` object for a named database.

Applies to `PDDatabaseQuery` object.

Prototype

```
Function ReadDbInfoByName(pName as String) as
    PDDatabaseInfo
```

Parameters

→ *pName*
Database name.

Returns The `PDDatabaseInfo` object that describes the database named by `pName`.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim DbInfo as PDDatabaseInfo
Set DbInfo = DbQuery.ReadDbInfoByName("MemoDB")
```

See Also `PDDatabaseInfo` object.

ReadDbNameList

Purpose Returns a list of classic database names that are either in RAM or ROM on the handheld.

Applies to `PDDatabaseQuery` objects.

Prototype `Function` **`ReadDbNameList`**`(ByVal` *`bRam`* `As Boolean) As Variant`

Parameters → *`bRam`*

RAM or ROM. When `True`, specifies RAM. When `False`, specifies ROM.

Returns A `String` array containing the database names.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim DbNames as Variant
' Read the RAM name list
DbNames = DbQuery.ReadDbNameList(True)
```

ReadFeature

Purpose Reads a feature value from the Feature Manager on the handheld.

Applies to `PDSystemAdapter` object.

Prototype

```
Function ReadFeature(vCreator as Variant, nFeature
    as Long) as Long
```

Parameters

→ *`vCreator`*
Creator ID. The unique ID associated with each database and application on the device. Each conduit is associated with a specific creator ID. It is four characters that can be in either `Long` (`VT_I4`) or Little Endian form.

→ *`nFeature`*
Feature number.

Returns The feature value.

Comments This method retrieves a feature value that is registered with the Feature Manager on the handheld. Features are stored in volatile storage that is erased and re-initialized during system reset. Palm OS and applications can register features using their own creator ID. The contents of features are completely application-specific.

Example

```
Dim pSystem as New PDSystemAdapter
Dim Feature as Long
Feature = pSystem.ReadFeature("AbCd", 2)
```

ReadIDList

Purpose Retrieves the row IDs of all the rows in a table that are in a set of categories.

Applies to `PSDRowAdapter` object.

Prototype

```
Function ReadIDList(ByVal TableName As String,
    CategoryIDList, [ByVal MatchMode As
    EPSDMatchMode = eDbMatchAny]) As Variant
```

Parameters

→ *`TableName`*
The name of the table.

→ *`CategoryIDList`*
An array of category IDs to match.

→ *`MatchMode`*
The category match mode that this method uses to match the specified category ID list against rows' category memberships. Specify one of the `EPSDMatchMode` values.

Returns A `Variant` array of the row IDs of rows that are members of the specified set of categories according to the match mode.

ReadModifiedIDList

Purpose Retrieves the row IDs of all the modified rows in a table that are in a set of categories.

Applies to `PSDRowAdapter` object.

Prototype

```
Function ReadModifiedIDList(ByVal TableName As
    String, CategoryIDList, [ByVal MatchMode As
    EPSDMatchMode = eDbMatchAny]) As Variant
```

Parameters

→ *`TableName`*
The name of the table.

→ *`CategoryIDList`*
An array of category IDs to match.

→ *`MatchMode`*
The category match mode that this method uses to match the specified category ID list against modified rows' category memberships. Specify one of the `EPSDMatchMode` values.

Returns A `Variant` array of the row IDs of modified rows that are members of the specified set of categories according to the match mode.

ReadModifiedRows

Purpose Reads the modified rows in a table that match the specified criteria.

Applies to `PSDRowAdapter` object.

Prototype

```
Function ReadModifiedRows(ByVal TableName As
    String, vCategoryIDList, [ByVal MatchMode As
    EPSDMatchMode = eDbMatchAny]) As IPSDRowSet
```

Parameters

→ *`TableName`*
The name of the table.

→ *`vCategoryIDList`*
An array of category IDs to match.

→ *`MatchMode`*
The category match mode that this method uses to match the specified category ID list against modified rows' category memberships. Specify one of the `EPSDMatchMode` values.

Returns A `PSDRowSet` object that contains the set of modified rows that are in the specified table and are members of the specified set of categories according to the match mode.

ReadNext

Purpose Reads the next record.

Applies to DmRecordAdapter, PDRecordAdapter, PDAddressDbHHRecordAdapter, PDDateBookDbHHRecordAdapter2, PDDateBookDbHHRecordAdapter, PDMemoDbHHRecordAdapter, PDTodoDbHHRecordAdapter objects.

Prototype For DmRecordAdapter and PDRecordAdapter objects:

```
Function ReadNext(nIndex As Long, pvUniqueId,
   nCategory As Long, eAttributes As
   ERecordAttributes) As Variant
```

For `PD<PIM>DbHHRecordAdapter` objects:

```
Function ReadNext() as Unknown
```

Parameters

← *`nIndex`*
Index of returned record.

← *`pvUniqueId`*
Unique ID of returned record.

← *`nCategory`*
Returned category.

← *`eAttributes`*
Attributes of returned record from the `ERecordAttributes` constants.

Returns For a DmRecordAdapter or PDRecordAdapter object, returns a `Byte` array containing returned record data.

For any of the objects representing classic databases used by any of four standard Palm OS applications (denoted `PD<PIM>DbHHRecordAdapter`), this method returns a corresponding object (denoted `PD<PIM>DbHHRecord`) representing the next record.

Comments This is an iterator method that uses the current iterator index. To begin at the first record, set the `IterationIndex` property to zero. If you use this method in conjunction with the `EOF` property to read all records in a database, note that `EOF` is set *after* the `ReadNext()` method returns nothing.

Example

```
Dim PUtil as New PDUtility
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter

' Open the database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")

Dim UniqueId as Variant
Dim Index as Long
Dim Category as Long
Dim Attributes as ERecordAttributes
Dim Data as Variant

' Reset the iteration index
Adapter.IterationIndex = 0

' Read the first record
Data = Adapter.ReadNext(Index, UniqueId, Category, _
   Attributes)

' Loop through all the remaining records until reaching EOF
Do While Not Adapter.EOF
   ' Do something with the current record
   ' Read the next record
   Data = Adapter.ReadNext(Index, UniqueId, Category, _
      Attributes)
   Dim Nextoffset as Long
   Dim varray as Variant
   nextoffset = Util.RocordIdtoByteArray (UniqueId, 0, _
      False, vArray)
Loop
```

See Also `EOF` property.
`ERecordAttributes` constant.

ReadNextInCategory

Purpose Reads the next record in a category.

Applies to DmRecordAdapter, PDRecordAdapter, PDAddressDbHHRecordAdapter, PDDateBookDbHHRecordAdapter2, PDDateBookDbHHRecordAdapter, PDMemoDbHHRecordAdapter, PDTodoDbHHRecordAdapter objects.

Prototype For DmRecordAdapter and PDRecordAdapter objects:

```
Function ReadNextInCategory(nIndex As Long,
    pvUniqueId, ByVal nCategory As Long,
    eAttributes As ERecordAttributes) As Variant
```

For `PD<PIM>DbHHRecordAdapter` objects:

```
Function ReadNextInCategory(nCategory as Long) as
    Unknown
```

Parameters

← *nIndex*
Index of returned record.

← *pvUniqueId*
Unique ID of returned record.

→ *nCategory*
Category ID of desired record.

← *eAttributes*
Attributes of returned record from the ERecordAttributes constants.

Returns For a DmRecordAdapter or PDRecordAdapter object, returns a `Byte` array containing the value of the next record belonging to the category specified by *nCategory*.

For any of the objects representing classic databases used by any of four standard Palm OS applications (denoted `PD<PIM>DbHHRecordAdapter`), this method returns a corresponding object (denoted `PD<PIM>DbHHRecord`) representing the record specified by *nCategory*.

Comments This is an iterator method that uses the current iterator index. To begin at the first record, set the IterationIndex property to zero.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
' Open the database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
Dim UniqueId as Variant
Dim Index as Long
Dim Attributes as ERecordAttributes
Dim Data as Variant
' Reset the iteration index
Adapter.IterationIndex = 0
' Loop through all the records
Data = Adapter.ReadNextInCategory(Index, UniqueId, 0,_
   Attributes)
Do While Not Adapter.EOF
   ' Do something with the current record
   ' Read the next record
   Data = Adapter.ReadNextInCategory(Index, UniqueId, 0, _
      Attributes)
Loop
```

See Also `ERecordAttributes` constants.

ReadNextModified

Purpose Reads the next modified record.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype For `PDRecordAdapter` objects:

```
Function ReadNextModified(nIndex As Long,
    pvUniqueId, nCategory As Long, eAttributes As
    ERecordAttributes) As Variant
```

For `PD<PIM>DbHHRecordAdapter` objects:

```
Function ReadNextModified() as Unknown
```

Parameters

← *`nIndex`*
Index of returned record.

← *`pvUniqueId`*
Unique ID of returned record.

← *`nCategory`*
Returned category ID of returned record.

← *`eAttributes`*
Attributes of returned record from the `ERecordAttributes` constants.

Returns For a `DmRecordAdapter` or `PDRecordAdapter` object, returns a `Byte` array containing the value of the next modified record.

For any of the objects representing classic databases used by any of four standard Palm OS applications (denoted `PD<PIM>DbHHRecordAdapter`), this method returns a corresponding object (denoted `PD<PIM>DbHHRecord`) representing the next modified record.

Comments This is an iterator method that uses the current iterator index. To begin at the first record, set the `IterationIndex` property to zero.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
' Open the database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
Dim UniqueId as Variant
Dim Index as Long
Dim Category as Long
Dim Attributes as ERecordAttributes
Dim Data as Variant
' Reset the iteration index
Adapter.IterationIndex = 0
' Loop through all the records
Data = Adapter.ReadNextModified(Index, UniqueId, Category, _
   Attributes)
Do While Not Adapter.EOF
   ' Do something with the current record
   ' Read the next record
   Data = Adapter.ReadNextModified(Index, UniqueId, _
      Category, Attributes)
Loop
```

See Also `ERecordAttributes` constants.

ReadNextModifiedInCategory

Purpose Reads the next modified record in a category.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype For `DmRecordAdapter` and `PDRecordAdapter` objects:

```
Function ReadNextModifiedInCategory(nIndex As
    Long, pvUniqueId, ByVal nCategory As Long,
    eAttributes As ERecordAttributes) As Variant
```

For `PD<PIM>DbHHRecordAdapter` objects:

```
Function ReadNextModifiedInCategory(nCategory as
    Long) as Unknown
```

Parameters

← *`nIndex`*
Index of returned record.

← *`pvUniqueId`*
Unique ID of returned record.

→ *`nCategory`*
Category ID of record to return.

← *`eAttributes`*
Attributes from the `ERecordAttributes` constants.

Returns For a `DmRecordAdapter` or `PDRecordAdapter` object, returns a `Byte` array containing record data.

For any of the objects representing databases used by any of four standard Palm OS applications (denoted `PD<PIM>DbHHRecordAdapter`), this method returns a corresponding object (denoted `PD<PIM>DbHHRecord`) representing the next modified record in the category specified by *`nCategory`*.

Comments This is an iterator method that uses the current iterator index. To begin at the first record, set the `IterationIndex` property to zero.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
' Open the database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
Dim UniqueId as Variant
Dim Index as Long
Dim Attributes as ERecordAttributes
Dim Data as Variant
' Reset the iteration index
Adapter.IterationIndex = 0
' Loop through all the records
Data = Adapter.ReadNextModifiedInCategory(Index, UniqueId, _
        0, Attributes)
Do While Not Adapter.EOF
    ' Do something with the current record
    ' Read the next record
    Data = Adapter.ReadNextModifiedInCategory(Index, _
        UniqueId, 0, Attributes)
Loop
```

See Also `ERecordAttributes` constants.

ReadNextResource

Purpose Reads the next record in a resource database.

Applies to `PDResourceAdapter` object.

Prototype `Function` **`ReadNextResource`**`(`*`nIndex`*` as Long, `*`nType`*` as Long, `*`nId`*` as Long) as Variant`

Parameters

← *`nIndex`*
Index of returned record.

← *`nType`*
Type of returned record.

← *`nId`*
ID of returned record.

Returns A `Byte` array containing the resource data.

Comments Uses the current iterator index. To begin at the first resource, set the `IterationIndex` property to zero.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDResourceAdapter
' Open the database
Set Adapter = DbQuery.OpenRecordDatabase("Application")
Dim Index as Long
Dim Type as Long
Dim Id as Long
Dim Data as Variant
' Reset the iteration index
Adapter.IterationIndex = 0
' Loop through all the resources
Data = Adapter.ReadNextResource(Index, Type, Id)
Do While Not Adapter.EOF
' Do something with the current record
' Read the next resource
Data = Adapter.ReadNextResource(Index, Type, Id)
Loop
```

See Also `PDRecordAdapter` object.

ReadResource

Purpose Reads a resource record by index.

Applies to `PDResourceAdapter` object.

Prototype

```
Function ReadResource(nIndex as Long, nType as
    Long, nId as Long) as Variant
```

Parameters

→ *nIndex*
Resource index of desired record.

← *nType*
Resource type of returned record.

← *nId*
Resource ID of resource record.

Returns A `Byte` array containing the value of the resource record specified by `nIndex`.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDResourceAdapter
Set Adapter =_
DbQuery.OpenResourceDatabase("Saved Preferences")
' Read a record
Dim Type as Long
Dim Id as Long
Dim ResourceData as Variant
ResourceData = Adapter.ReadResource(2, Type, Id)
```

ReadRow

Purpose Reads an entire row—attributes, category memberships, and column values—from a schema database on the handheld.

Applies to `PSDRowAdapter` object.

Prototype `Function` **`ReadRow`**`(ByVal` *`RowID`*`) As IPSDRowData`

Parameters → *`RowID`*
The row ID of the row to read.

Returns A `PSDRowData` object that contains all of the row's data.

ReadRowInfo

Purpose Retrieves information about a row, but no column values, from a schema database on the handheld.

Applies to `PSDRowAdapter` object.

Prototype `Function **ReadRowInfo**(ByVal *vRowID*) As IPSDRowData`

Parameters → *`vRowID`*

The row ID of the row.

Returns A `PSDRowData` object that contains only information about the row but no column values.

Comments The returned `PSDRowData` object responds with valid information for only the following properties and methods:

- `TableName` property
- `RowID` property
- `CategoryIDList` property
- `IsArchived()` method
- `IsDeleted()` method
- `IsReadOnly` property
- `IsDataModified()` method
- `IsMembershipModified()` method
- `IsPrivate` property

ReadRows

Purpose Reads entire rows that match the given criteria from a schema database on the handheld.

Applies to `PSDRowAdapter` object.

Prototype
```
Function ReadRows(ByVal TableName As String,
    CategoryIDList, [ByVal MatchMode As
    EPSDMatchMode = eDbMatchAny]) As IPSDRowSet
```

Parameters

→ *`TableName`*
The name of the table.

→ *`CategoryIDList`*
An array of category IDs to match.

→ *`MatchMode`*
The category match mode that this method uses to match the specified category ID list against the rows' category memberships. Specify one of the `EPSDMatchMode` values.

Returns A `PSDRowSet` object that contains the set of all rows that are in the specified table and are members of the specified set of categories according to the match mode.

ReadRowsByIDList

Purpose Reads entire rows that are on the specified row ID list from a schema database on the handheld.

Applies to `PSDRowAdapter` object.

Prototype

```
Function ReadRowsByIDList(vRowIDList) As
    IPSDRowSet
```

Parameters → *`vRowIDList`*

A `Variant` array of row IDs.

Returns A `PSDRowSet` object that contains the set of all rows whose row IDs are on the specified list.

ReadSortInfoBlock

Purpose Reads a record database's sort info block.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype `Function ReadSortInfoBlock () as Variant`

Parameters None.

Returns A `Byte` array containing the sort info block.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Read the SortInfo block
Dim vSortInfo as Variant
vSortInfo = Adapter.ReadSortInfoBlock
```

ReadUniqueIdList

Purpose Creates a list of unique IDs in record index order.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype

```
Function ReadUniqueIdList(ByVal nFirstIndex As
    Long, nRead As Long) As Variant
```

Parameters

→ *`nFirstIndex`*
Beginning index.

↔ *`nRead`*
Before the call, the number of IDs to read. After the call, the number of IDs that were returned.

Returns An array of unsigned `Long`s containing the unique IDs. Also returns the number of elements read in *`nRead`*.

Comments The `Variant` is a usable array. Index the variable for each value to set a `Long` array to the variant. If *`nFirstIndex`* and *`nRead`* specify an illegal range, the request is truncated.

See "Miscellaneous Changes" on page 594 for details on a related problem fixed in the COM Sync module in CDK 6.0.

Example

```
Dim pArray1 as Variant
Dim pArray2 () as Long
Dim nRead as Long
Dim UniqueId as Long
Dim Idx as Long
Dim pAdapter As PDRecordAdapter
nRead = 10
pArray1 = pAdapter.ReadUniqueIdList (0, nRead)
   ' Alt #1
For Idx = 0 to nRead - 1
   UniqueId = pArray1 (Idx)
Next
   ' Alt #2
pArray2 = pArray1
For Idx = 0 to nRead - 1
   UniqueId = pArray2 (Idx)
Next
```

RebootSystem

Purpose Sends a request to soft-reset the handheld at the end of the HotSync operation.

Applies to `PDSystemAdapter` object.

Prototype `Sub` **`RebootSystem`** `()`

Parameters None.

Returns None.

Example

```
Dim pSystem as New PDSystemAdapter
' Reboot
pSystem.RebootSystem
```

RecordIdToByteArray

Purpose Converts a record ID to a `Byte` array.

Applies to `PDUtility` object.

Prototype
```
Function RecordIdToByteArray(vRecordId as Variant,
    nOffset as Long, bSwap as Boolean, vData as
    Variant) as Long
```

Parameters

→ *`vRecordId`*
: Record ID to convert.

→ *`nOffset`*
: Offset from beginning of `Byte` array where `vRecordId` is to be inserted.

→ *`bSwap`*
: If `True`, this method swaps the bytes before returning *`vData`*.

↔ *`vData`*
: `Byte` array used for insertion.

Returns The next offset in the `Byte` array.

Comments This method converts a record ID into a `Byte` array. You can use the `Byte` array format to read/write the record ID from and to a binary file. Palm OS record IDs are long integers but may change in the future. PalmSource, Inc. strongly recommend that you use the `PDUtility` methods to convert record IDs from and to `Byte` array and `String` formats.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim pUtil as New PDUtility
Dim Adapter as PDRecordAdapter
' Open the database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
Dim UniqueId as Variant
Dim Index as Long
Dim Category as Long
Dim Attributes as ERecordAttributes
Dim Data as Variant
' Reset the iteration index
Adapter.IterationIndex = 0
Data = Adapter.ReadNext(Index, UniqueId, Category,_
   Attributes)
Dim vArray as Variant
NextOffset = pUtil.RecordIdToByteArray(UniqueId, 0, _
   False, vArray)
```

RecordIdToString

Purpose Converts record ID to a readable `String`.

Applies to `PDUtility` object.

Prototype
```
Function RecordIDToString(vRecordId as Variant) as
    String
```

Parameters ← *`vRecordId`*
Unique ID of record.

Returns A string (`BSTR`) containing returned record data in readable string format.

Comments This method is provided to convert a record ID into a string. You can subsequently use the string to read/write the record ID from/to a text file. Currently Palm OS record IDs are long integers. This may change in future. PalmSource, Inc. strongly recommends using `PDUtility` methods such as these to convert record IDs from/to `Byte` array/`String` formats.

Example
```
Dim DbQuery as New PDDatabaseQuery
Dim pUtil as New PDUtility
Dim Adapter as PDRecordAdapter
' Open the database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
Dim UniqueId as Variant
Dim Index as Long
Dim Category as Long
Dim Attributes as ERecordAttributes
Dim Data as Variant
' Reset the iteration index
Adapter.IterationIndex = 0
Data = Adapter.ReadNext(Index, UniqueId, Category,_
   Attributes)
Dim strRecordId as String
strRecordId = pUtil.RecordIdToString(UniqueId)
```

Refresh

Purpose Reinitializes this object from its source, discarding any changes in the cache.

Applies to `DmCategories`, `DmDatabaseInfo`, `PDCategories`, `PDDatabaseInfo` objects.

Prototype `Sub Refresh()`

Parameters None.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim DbInfo as PDDatabaseInfo
Set DbInfo = DbQuery.ReadDatabaseInfoByName("MemoDB") _
   ' Reread the database information
DbInfo.Refresh
```

RefreshConduitInfo

Purpose Requests that HotSync Manager reload information about all registered conduits.

Applies to `PDHotSyncUtility` object.

Prototype `Sub` **`RefreshConduitInfo`**`()`

Parameters None.

Returns None.

Errors `eHotSyncNotFound`
HotSync Manager is not running.

Comments HotSync Manager versions *earlier than 6.0* must be refreshed or restarted after registering a conduit. *Versions 6.0 and later* automatically refresh their lists so that calling this method is unnecessary.

If you register a conduit while HotSync Manager is running, your installer can call this method to make HotSync Manager reload the conduit configuration entries and recognize your newly registered conduit. If your installer changes other settings not related to a conduit (HotSync Manager communication settings, backup conduit, and so on), this method does not reload those settings; use `RestartHotSyncMgr()` instead.

See Also `RestartHotSyncMgr()`, `RegisterConduit()`, `UnregisterConduit()` methods

RegisterConduit

Purpose Registers a conduit based on the information provided in a `PDConduitInfo` object.

Applies to `PDCondMgr`, `PDSystemCondMgr` objects.

Prototype `Sub **RegisterConduit**(*ConduitInfo* As PDConduitInfo)`

Parameters → *`ConduitInfo`*
A `PDConduitInfo` object specifying the conduit you want to register.

Returns None.

Errors

`eAlreadyExists`
Another conduit is already registered with this creator ID.

`eCantCreateConduit`
The conduit could not be registered with HotSync Manager.

`eCantSetValue`
One or more conduit configuration entries could not be set.

`eInvalidID`
The specified conduit creator ID is not valid.

`eLocalMemory`
Not enough memory on the desktop to perform the requested operation.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments This method registers a conduit either for the current Windows user or the system, depending on whether it is called for a `PDCondMgr` or a `PDSystemCondMgr` object.

Only *one* system and user conduit can be registered with a given creator ID. If another system or user conduit is already registered with the creator ID you specify in `PDConduitInfo`, this method generates an `eAlreadyExists` error. Note that you can register a user (or system) conduit with the same creator ID used by another system (or user) conduit. For more information, see “User- and System-registered Conduits and Notifiers” on page 78 in *Introduction to Conduit Development*.

IMPORTANT: For HotSync Manager versions *earlier than 6.0*: after registering a conduit, you must call either `RefreshConduitInfo()` or `RestartHotSyncMgr()` (or exit and relaunch HotSync Manager manually) for HotSync Manager to recognize your newly registered conduit. Versions 6.0 and later do not require this.

Example

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

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

   On Error GoTo ErrorHandler

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

   ' Check whether a conduit with the specified CreatorID
   ' currently exists.
   If CreatorIDExists Then
      If MsgBox("A conduit with CreatorID '" & _
         strConduitCreatorID & "' already exists." & _
         " Do you want to remove it ?", vbYesNo + _
         vbQuestion, "Remove Conduit") = vbYes Then
         Call PDCondMgr.UnregisterConduit(CreatorID)
      Else
         Exit Function
      End If
   End If
   Set RetrievePDConduitInfo = Nothing

   ' Set the conduit entries
   With PConduitInfo
      .COMClassID = "c:\winnt\system32\calc.exe"
      .CreatorID = CreatorID
      .DeskTopDataDirectory = "DeskTopDataDirectory"
      .HandHeldDB = "HandHeldDB"
      .DeskTopDataFile = "DeskTopDataFile"
      .DisplayName = "DisplayName"
      .Priority = 2
   End With

   Call PDCondMgr.RegisterConduit(PConduitInfo)

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

   RegisterConduit = True
   Exit Function


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

See Also `PDConduitInfo` object
`UnregisterConduit()`, `RefreshConduitInfo()`,
`RestartHotSyncMgr()` methods

RegisterIC

Purpose Registers an install conduit based on the information provided in a `PDInstallConduitInfo` object.

Applies to `PDInstallConduit` object.

Prototype `Sub` **`RegisterIC`**`(`*`ICInfo`*` As PDInstallConduitInfo)`

Parameters → *`ICInfo`*
A `PDInstallConduitInfo` object for the install conduit you want to register.

Returns None.

Errors

`eAlreadyExists`
Another install conduit is already registered with this unique ID.

`eCantCreateConduit`
The install conduit could not be registered with HotSync Manager.

`eCantSetValue`
A conduit configuration entry could not be set.

`eInvalidInstallID`
The specified unique ID is not valid.

`eLocalMemory`
Not enough memory on the desktop to perform the requested operation.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments Only *one* install conduit can be registered with a given unique ID. Fill in the fields of a `PDInstallConduitInfo` object and call this method to register this install conduit with HotSync Manager. If another install conduit is already registered with the unique ID you specify in `PDInstallConduitInfo`, this method generates an `eAlreadyExists` error.

IMPORTANT: After registering an install conduit, you must call either `RefreshConduitInfo()` or `RestartHotSyncMgr()` (or exit and relaunch HotSync Manager manually) for HotSync Manager to recognize your newly registered install conduit.

Example

```
Dim PInstall As New PDInstallConduit
Dim PInfo As New PDInstallConduitInfo

PInfo.Directory = "Install"
PInfo.Extension = "All Files (*.*)|*.*"
PInfo.Module = "MyInstallConduit.dll"
PInfo.Name = "Test Install"
PInfo.UniqueId = 1952805748

Call PInstall.RegisterIC(PInfo)
Call PInstall.UnregisterIC(1952805748)
```

See Also `PDInstallConduitInfo` object
`UnregisterIC()`, `RefreshConduitInfo()`,
`RestartHotSyncMgr()` method

RegisterNotifier

Purpose Registers a notifier with HotSync Manager.

Applies to `PDCondMgr` object.

Prototype `Sub **RegisterNotifier**(NotifierPath As String)`

Parameters → *`NotifierPath`*
The full path and filename of the notifier you want to register.

Returns None.

Errors `eAlreadyInstalled`
A notifier with this path is already registered.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments **IMPORTANT:** After registering a notifier, you must call `RestartHotSyncMgr()` (or exit and relaunch HotSync Manager manually) for HotSync Manager to recognize your newly registered notifier.

Example

```
Dim PDCondMgr As New PDCondMgr

Call PDCondMgr.RegisterNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll")
Call PDCondMgr.ModifyNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll", "C:\PdN20d.dll")
Call PDCondMgr.UnregisterNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll")
```

See Also `UnregisterNotifier()`, `ModifyNotifier()`, `RestartHotSyncMgr()` methods

Remove

Purpose Deletes the specified record from an open classic or extended record database on the handheld.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype `Sub` **`Remove`**`(ByVal` *`varUniqueId`*`)`

Parameters → *`varUniqueId`*
ID of the record to remove.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Remove the first record
Dim UniqueId as Variant
Dim Category as Long
Dim Attributes as ERecordAttributes
Adapter.ReadByIndex0, UniqueId, Category, Attributes
Adapter.Remove UniqueId
```

RemoveAllResources

Purpose Deletes all resources from an open resource database on the handheld.

Applies to `PDResourceAdapter` object.

Prototype `Sub` **`RemoveAllResources`**`()`

Parameters None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDResourceAdapter
Set Adapter = DbQuery.OpenResourceDatabase("Application")
' Remove all resource records
Adapter.RemoveAllResources
```

RemoveAllSecretRowsInTable

Purpose	Removes all of the secret rows in a table in a schema database.
Applies to	`PSDRowAdapter` object.
Prototype	`Sub` **`RemoveAllSecretRowsInTable`**`(ByVal` *`TableName`* `As String)`
Parameters	→ *`TableName`* The name of the table.
Returns	None.
Comments	This method destroys all of the data in the affected rows.

RemoveCategory

Purpose	Removes a category from a schema database.
Applies to	`PSDCategoryAdapter` object.
Prototype	`Sub` **`RemoveCategory`**`(ByVal` *`CategoryID`* `As Long)`
Parameters	→ *`CategoryID`* Specifies the category ID of the category to remove.
Returns	None.
Comments	This method removes the membership in this category from all rows, but otherwise leaves these rows intact.

RemoveCategoryFromAllRows

Purpose Removes all matching rows from a specified list of categories in this schema database.

Applies to `PSDDatabaseAdapter` object.

Prototype

```
Sub RemoveCategoryFromAllRows(vCategoryIDList,
    ByVal MatchMode As EPSDMatchMode)
```

Parameters

→ *`vCategoryIDList`*
A `Variant` array of category IDs to match.

→ *`MatchMode`*
The category match mode that this method uses to match the specified category ID list against rows’ category memberships. Specify one of the `EPSDMatchMode` values.

Returns None.

Comments Rows whose category memberships match the categories specified in *`CategoryIDList`* are removed from those categories.

RemoveCategoryMembership

Purpose Removes a row from all of the categories on a list.

Applies to `PSDRowAdapter` object.

Prototype

```
Sub RemoveCategoryMembership(ByVal vRowID,
    CategoryIDList)
```

Parameters

→ *vRowID*
The row ID of the row.

→ *CategoryIDList*
An array of category IDs.

Returns None.

RemoveColumnCustomProperty

Purpose Removes a custom property from a table column in this schema database.

Applies to `PSDDatabaseAdapter` object.

Prototype

```
Sub RemoveColumnCustomProperty(ByVal TableName As
    String, ByVal ColumnID As Long, ByVal
    PropertyID As Integer)
```

Parameters

→ *`TableName`*
The name of the table.

→ *`ColumnID`*
The column ID of the column.

→ *`PropertyID`*
The property ID of the custom column property. Valid values range from 0x05 to 0x0A.

Returns None.

RemoveColumns

Purpose	Removes column definitions from this table given a list of column IDs.
Applies to	`PSDTable` object.
Prototype	`Sub RemoveColumns(`*`vColumnIDList`*`)`
Parameters	→ *`vColumnIDList`* A `Variant` array of column IDs.
Returns	None.

RemoveDatabase

Purpose Deletes a classic or extended database on the handheld.

Applies to `DmDatabaseQuery`, `PDDatabaseQuery` object.

Prototype `Sub` **`RemoveDatabase`**`(ByVal` *`pName`* `As String)`

Parameters → *`pName`*
Name of the database to delete.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
' Remove the Memo database
DbQuery.RemoveDatabase ("MemoDB")
```

RemoveFileFromHHQueue

Purpose Removes a file from the queue of files that are to be installed in primary storage on a user's *handheld*.

Applies to `PDInstall` object.

Prototype
```
Sub RemoveFileFromHHQueue(UserID As Long, FileName
    As String)
```

Parameters

→ *`UserID`*
: A unique ID to specify the user you want to reference.

→ *`FileName`*
: The name of the file to remove (include no path).

Returns None.

Errors

`eFailedToDelete`
: This method failed to remove the specified install file because, for example, the file does not exist.

`eInvalidPath`
: The path of the slot-install directory is longer than 256 characters and cannot be retrieved.

`eParamError`
: Parameters were not passed correctly.

Comments This method removes the specified file from the user's handheld-install directory that is associated with files of the type to remove.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetAllQueuedHHFiles()`, `GetAllQueuedHHFilesOfType()`, `InstallFileToHH()` methods

RemoveFileFromSlotQueue

Purpose Removes a file from the queue of files that are to be installed in secondary storage in an expansion *slot* of a user's handheld.

Applies to `PDInstall` object.

Prototype

```
Sub RemoveFileFromSlotQueue(UserID As Long, SlotID
    As Long, FileName As String)
```

Parameters

→ *`UserID`*
A unique ID to specify the user you want to reference.

→ *`SlotID`*
The ID of the slot from whose directory to remove a file. To get slot IDs, use `PDUserData`'s `GetSlotList()` method.

→ *`FileName`*
The name of the file to remove (include no path).

Returns None.

Errors

`eFailedToDelete`
This method failed to remove the specified install file because, for example, the file does not exist.

`eInvalidPath`
The path of the slot-install directory is longer than 256 characters and cannot be retrieved.

`eParamError`
Parameters were not passed correctly.

Comments This method removes a file on the desktop computer that was queued in a slot-install directory for a given user. This method accepts all file types.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetSlotList()`, `GetAllQueuedSlotFiles()`, `InstallFileToSlot()` methods

RemoveResource

Purpose Deletes a resource from an open resource database on the handheld.

Applies to `PDResourceAdapter` object.

Prototype `Sub RemoveResource(vType as Variant, nId as Long)`

Parameters

→ *vType*
Four-byte resource type that can be passed in either `Long` (`VT_I4`) or Little Endian form.

→ *nId*
Resource ID.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDResourceAdapter
Set Adapter = DbQuery.OpenResourceDatabase("Application")
' Remove the first resource
Dim Type as Long
Dim Id as Long
Adapter.ReadResource (0, Type, Id)
Adapter.RemoveResource (Type, Id)
```

RemoveRow

Purpose Removes a row from a schema database.

Applies to `PSDRowAdapter` object.

Prototype `Sub` **`RemoveRow`**`(ByVal` *`vRowID`*`)`

Parameters → *`vRowID`*
The row ID of the row.

Returns None.

Comments This method destroys all of the data in the specified row. Contrast this method with `DeleteRow()`.

RemoveSet

Purpose Deletes a set of records in a classic or extended database.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype

```
Sub RemoveSet(ByVal eSetType As ERemoveSetType,
   [ByVal nCategory As Long = 255])
```

Parameters

→ *`eSetType`*
The type of records to delete specified by the the `ERemoveSetType` constants.

→ *`nCategory`*
Optional category for options requiring it.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Remove all deleted records
Adapter.RemoveSet (eRemoveAllDeletedRecords)
```

See Also `ERemoveSetType` constants.

RemoveTable

Purpose	Removes a table from this schema database.
Applies to	`PSDDatabaseAdapter` object.
Prototype	`Sub` **`RemoveTable`**`(ByVal` *`TableName`* `As String)`
Parameters	→ *`TableName`* The name of the table.
Returns	None.
Comments	If any rows belong to the specified table, then this method does not remove the table but instead returns an error.

RemoveUserTemporarySyncPreferences

Purpose Removes the specified conduit's temporary synchronization preferences for the specified user ID.

Applies to `PDUserData` object.

Prototype
```
Sub RemoveUserTemporarySyncPreferences(dwUserId As
    Long, ConduitCreatorId As Long)
```

Parameters

→ *`dwUserId`*
: A unique ID to specify the user to reference in the users data store.

→ *`ConduitCreatorId`*
: The creator ID of the conduit you want to remove the preferences of.

Returns None.

Errors

`eInvalidUser`
: `dwUserId` is an invalid number.

`eNoCorePath`
: No path to find the users data store was found.

`eNoUsers`
: The users data store exists, but contains no information.

`eOtherUDErr`
: No users data store was found or another method or program is accessing the user data store.

`eUDSemaphoreError`
: Another method or program is accessing the user data store.

`eUDUnableToCreate`
: Creating a new users data store failed because of a file error.

Comments This method clears the temporary synchronization preferences for the specified conduit so that the action set in the permanent synchronization preferences will be taken during the next HotSync operation. The result is the same as if the user had never clicked HotSync Manager's **Custom** > **Change** option and altered the conduit's temporary synchronization preferences.

NOTE: `RemoveUserTemporarySyncPreferences()` clears only *one* conduit's temporary synchronization preferences. Contrast it with `DeleteUserTemporarySyncPreferences()`, which clears the temporary preferences for *all* the user's conduits.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetConduitList()`, `DeleteUserTemporarySyncPreferences()`, `GetUserTemporarySyncPreferences()`, `SetUserTemporarySyncPreferences()`, `DeleteUserPermanentSyncPreferences()`, `GetUserPermanentSyncPreferences()`, `SetUserPermanentSyncPreferences()` methods

Rename

Purpose Renames a closed file or directory on an expansion card.

Applies to `PDVFSVolumeManager` object.

Prototype

```
Sub Rename(OriginalName As String, NewName As
    String)
```

Parameters

→ *`OriginalName`*
The full path of the file or directory to rename.

→ *`NewName`*
The filename or the directory name of the new file/directory (not a full path).

Returns None.

Errors

`eCommunications`
Communications with the handheld has either not been initialized or has been lost.

`eParamError`
Parameters were not passed correctly.

`eVFSBadName`
Invalid filename or path.

`eVFSFileAlreadyExists`
A file or directory with this name exists in this location already.

`eVFSFileNotFound`
The file or directory was not found in the specified path.

`eVFSFilePermissionDenied`
Permission denied to perform requested operation because the file or directory is read-only.

`eVFSFileStillOpen`
The file is still open—for example, trying to rename an open file.

`eVFSInvalidOperation`
A file system is not present.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
: The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
: The volume reference number is invalid because, for example, the volume has not been mounted.

`eVFSVolumeFull`
: There is insufficient space left on the volume.

Comments This method cannot be used to move a file to another location within the file system because it accepts only file or directory names, not full paths. This method returns `eVFSBadName` if either `OriginalName` or `NewName` is invalid.

See Also `Close()`, `CreateFile()`, `CreateDirectory()`, `Delete()` methods.

RenameCategory

Purpose Changes the name of a category in a schema database.

Applies to `PSDCategoryAdapter` object.

Prototype

```
Sub RenameCategory(ByVal CategoryID As Long, ByVal
    NewName As String)
```

Parameters

→ *`CategoryID`*
Specifies the category ID of the category to rename.

→ *`NewName`*
Specifies a new category name.

Returns None.

ResetAllModifiedFlags

Purpose Resets the modified (dirty) flag of all records in the open classic or extended record database on the handheld.

Applies to DmRecordAdapter, PDRecordAdapter, PDAddressDbHHRecordAdapter, PDDateBookDbHHRecordAdapter2, PDDateBookDbHHRecordAdapter, PDMemoDbHHRecordAdapter, PDTodoDbHHRecordAdapter objects.

Prototype `Sub` **`ResetAllModifiedFlags`**`()`

Parameters None.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
Adapter.ResetAllModifiedFlags
```

ResetComm

Purpose	Resets the communication methods of the HotSync Manager application.
Applies to	`PDHotSyncUtility` object.
Prototype	`Sub` **`ResetComm`**`()`
Parameters	None.
Returns	None.
Errors	`eHotSyncNotFound` HotSync Manager is not running.
Comments	This method causes HotSync Manager to change stored settings so that, the next time HotSync Manager is started, only the local serial communication method is enabled; modem and network methods are disabled.
See Also	`StartHotSyncMgr()`, `RestartHotSyncMgr()`, `SetCommStatus()` methods

ResetDirtyFlags

Purpose Resets all the category `Dirty` flags to `False`.

Applies to `DmCategories`, `PDCategories` objects.

Prototype `Sub` **`ResetDirtyFlags`**`()`

Parameters None.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get a categories object
Dim Categories as PDCategories
Set Categories = Adapter.PDCategories
Categories.ResetDirtyFlags
```

RestartHotSyncMgr

Purpose Restarts the HotSync Manager application.

Applies to `PDHotSyncUtility` object.

Prototype `Sub` **`RestartHotSyncMgr`**`(`*`options`*` As Long)`

Parameters → *`options`*

A long value that is all the OR-ed flags that specify how you want to restart HotSync Manager. Use the constants described in "HotSync Manager Start Options Constants" on page 572.

Returns None.

Errors `eUnableToStart`

This method cannot start the HotSync Manager application.

Comments If HotSync Manager is running, this method closes and restarts it. If you make changes to any HotSync Manager settings (other than registering a conduit), you must restart HotSync Manager for it to recognize the change. To recognize only newly registered conduits, you can use `RefreshConduitInfo()` instead, though `RestartHotSyncMgr()` causes HotSync Manager to reread all configuration entries, including those of newly registered conduits.

NOTE: Whenever you change any of the configuration entries, HotSync Manager versions earlier than 6.0 require that you call either `RestartHotSyncMgr()` to restart HotSync Manager to recognize all but conduit configuration changes or `RefreshConduitInfo()` to recognize a conduit configuration change.

However, HotSync Manager versions 6.0 and later automatically discover changes to conduit configuration information without you calling these functions. To recognize other changes (HotSync Manager communication settings, backup conduit, and so on), you must still call `RestartHotSyncMgr()`.

See Also `RefreshConduitInfo()`, `StartHotSyncMgr()`, `TerminateHotSyncMgr()` methods

RestoreSecurityData

Purpose	Restores vault databases from the desktop to the handheld.
Applies to	`PSDDatabaseQuery`, `PSDDatabaseUtilities` objects.
Prototype	`Sub` **`RestoreSecurityData`**`(ByVal` *`SourceDir`* `As String)`
Parameters	→ *`SourceDir`* The path of the directory that holds image files of vault databases to restore. Specify a null-terminated string; do not pass in a null value. The directory must contain one or more vault databases previously backed up by a call to `BackupSecurityData()`. It can also contain other files, but only if they do not use the same filename extension as vaults.
Returns	None.
Comments	After a handheld is hard-reset, **vault**s must be restored to the handheld *before* all other databases and in a specific order so that the handheld Authorization Manager allows other secure databases to be restored afterwards. This method restores all vault databases from the specified directory and in the order mandated by the Authorization Manager. If the vault already exists on the handheld, this method deletes it from the handheld and restores the vault from the desktop.
Compatibility	Palm OS version: Palm OS Cobalt, version 6.0 or later.

Save

Purpose Writes the category information into the application info block of this database and writes the application info block to the handheld.

Applies to `DmCategories`, `PDCategories` objects.

Prototype `Sub` **`Save`**`()`

Parameters None.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get a category object
Dim Category as PDCategories
Set Category = Adapter.PDCategories
' Reset the dirty flags and save
Category.ResetDirtyFlags
Category.Save
```

Comments Merges category information into the application info block. Existing data in application info block beyond the category information is left unchanged.

Seek

Purpose Sets the position from which to read or write within an open file on an expansion card.

Applies to `PDVFSFileManager` object.

Prototype `Sub` **`Seek`**`(`*`origin`*` As EPDFileOrigin, `*`offset`*` As Long)`

Parameters

→ *`origin`*
The origin to use when calculating the new position. The `offset` parameter indicates the desired new position relative to this origin, which must be one of the `EPDFileOrigin` constants.

→ *`offset`*
The offset, either positive or negative, from the origin to which to set the current position. A value of zero positions you at the specified origin.

Returns None.

Errors

`eParamError`
Parameters were not passed correctly.

`eVFSFileBadRef`
The file reference number is invalid: it has been closed or was not obtained from `Open()`.

`eVFSInvalidOperation`
A file system is not present.

`eVFSIsADirectory`
This operation can be performed only on a regular file, not a directory.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
The volume reference number is invalid because, for example, the volume has not been mounted.

Comments This method operates only on files and cannot be used with directories.

If the resulting position of the file pointer would be beyond the end of the file, this method sets the position to the end of the file. Similarly, if the resulting position of the file pointer would be before the beginning of the file, this method sets the position to the beginning of the file.

See Also `Tell()`, `Read()`, `Write()`, `Open()` methods.
`EPDFileOrigin` constants.

SetBackupConduit

Purpose Sets the filename of the HotSync Manager **backup conduit**.

Applies to `PDCondMgr`, `PDSystemCondMgr` objects.

Prototype `Sub SetBackupConduit(BackupConduit As String)`

Parameters → *`BackupConduit`*
The full path or filename of the backup conduit.

Returns None.

Errors `eCantSetValue`
This conduit configuration entry could not be set.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments This method establishes the value of the `HotSync Manager\BackupConduit` configuration entry used by HotSync Manager for the current Windows user or for the system, depending on whether this method is called for a `PDCondMgr` or a `PDSystemCondMgr` object.

If you specify a filename that this method cannot find (or a `Null` filename), this method leaves the filename unchanged and generates no error. Before calling this method, you must check that the file exists where this method can find it.

Example

```
Dim PDCondMgr As New PDCondMgr
Call PDCondMgr.SetBackupConduit("C:\BackupCnd.dll")
```

See Also `GetBackupConduit()` method

SetCategoryMembership

Purpose Adds a row to all the categories on a list.

Applies to `PSDRowAdapter` object.

Prototype

```
Sub SetCategoryMembership(ByVal vRowID,
    CategoryIDList)
```

Parameters

→ *`vRowID`*
The row ID of the row.

→ *`CategoryIDList`*
An array of category IDs. This method adds the row to all of these categories.

Returns None.

SetColumnCustomProperty

Purpose Sets the value of a custom column property in a table.

Applies to `PSDDatabaseAdapter` object.

Prototype

```
Sub SetColumnCustomProperty(ByVal TableName As
    String, ByVal ColumnID As Long, ByVal
    PropertyID As Integer, ByVal vPropertyValue)
```

Parameters

→ *`TableName`*
The name of the table.

→ *`ColumnID`*
The column ID of the column.

→ *`PropertyID`*
The property ID of the custom column property. Valid values range from 0x05 to 0x0A.

→ *`vPropertyValue`*
A byte array containing the value of the custom column property to set.

Returns None.

SetCommStatus

Purpose Sets the status of the HotSync Manager application's communication types.

Applies to `PDHotSyncUtility` object.

Prototype
```
Sub SetCommStatus(type As EPDHSConnectionType,
   status As EPDHSConnectionStatus)
```

Parameters

→ *`type`*
The communication type of which to retrieve the status. Use one of the values defined by the `EPDHSConnectionType` constant.

→ *`status`*
The new status to set for this communication type. Use one of the values defined by the `EPDHSConnectionStatus` constant.

Returns None.

Errors

`eInvalidConnType`
The specified HotSync Manager connection type is not one defined by the `EPDHSConnectionType` constant.

`eInvalidType`
The specified HotSync Manager connection type status is not one defined by the `EPDHSConnectionStatus` constant.

Comments When HotSync Manager is running, this method requests that it change the status of the specified communication type. If HotSync Manager is not running, this method configures its stored preferences.

See Also `GetCommStatus()`, `ResetComm()`, `RestartHotSyncMgr()` methods.
`EPDHSConnectionType`, `EPDHSConnectionStatus` constants.

SetDWORDData

Purpose Sets a `DWORD` configuration entry value for the specified conduit.

Applies to `PDCondMgr`, `PDInstallConduit`, `PDSystemCondMgr` objects.

Prototype `PDCondMgr` and `PDSystemCondMgr`:

```
Sub SetDWORDData(CreatorID As Long, Name As
    String, Data As Long)
```

`PDInstallConduit`:

```
Sub SetDWORDData(UniqueId As Long, Name As String,
    Data As Long)
```

Parameters

→ *`CreatorID`*
If a `PDCondMgr` or `PDSystemCondMgr` object, this parameter is the creator ID of the conduit you want to set a value for.

→ *`UniqueId`*
If a `PDInstallConduit` object, this parameter is the unique ID of the install conduit you want to set a value for.

→ *`Name`*
The name of the `DWORD` configuration entry you want to set.

→ *`Data`*
The `DWORD` value you want to set the specified entry to.

Returns None.

Errors

`eInvalidID`
The specified conduit creator ID is not valid.

`eNoSuchConduit`
The specified install conduit does not exist.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments This is a general purpose method that allows you to create or change a configuration entry by name. If the conduit you want is a standard synchronization conduit (most are), specify the creator ID in the first parameter. If the conduit you want is an **install conduit**, specify the unique ID in the first parameter.

This method sets information about a conduit that is registered either for the current Windows user or the system, depending on whether it is called for a `PDCondMgr` or a `PDSystemCondMgr` object.

Example

```
Dim ExtraInfo As Long
Dim CreatorId As Long
Dim PCondMgr As New PDCondMgr

' Set the value for a custom field called "ExtraInfo"
' to 10.
CreatorId = PCondMgr.StringToCreatorID("memo")

Call PCondMgr.SetDWORDData(CreatorId, "ExtraInfo", 10)
ExtraInfo = PCondMgr.GetDWORDData(CreatorId, "ExtraInfo")
```

See Also `GetDWORDData()`, `GetConduitList()` methods

SetExceptionDates

Purpose	Sets the exception dates for a repeating event in Date Book.
Applies to	`PDDateBookDbHHRecord2` object.
Prototype	`Sub` **`SetExceptionDates`**`(`*`pvDates`*`)`
Parameters	→ *`pvDates`* A `Variant` array of values of type `Date`. These are the dates on which a repeating Date Book event does not occur.
Returns	None.
See Also	`GetExceptionDates()` method.

SetIntegerValue

Purpose Sets an integer value to a key in the specified user's area of the users data store.

Applies to `PDUserData` object.

Prototype

```
Sub SetIntegerValue(dwUserId As Long, Section As
    String, Key As String, value As Long)
```

Parameters

→ *`dwUserId`*
: A unique ID to specify the user to reference in the users data store.

→ *`Section`*
: The section name in the specified user's area of the users data store.

→ *`Key`*
: The key of the integer to set.

→ *`value`*
: The new integer value to set.

Returns None.

Errors

`eInvalidUser`
: `dwUserId` is an invalid number.

`eOtherUDErr`
: The attempt to write to the specified user's area of the users data store failed.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetIntegerValue()`, `DeleteKey()` methods

SetPath

Purpose Sets the value of one of the stored path variables.

Applies to `PDInstall` object.

Prototype `Sub SetPath(type As EPDPathType, value As String)`

Parameters → *`type`*
A constant of type `EPDPathType` that specifies which path name you want to set.

→ *`value`*
A `BSTR` containing the new path to store.

Returns None.

Errors `eParamError`
Parameters were not passed correctly.

Comments This method sets the value of a path variable in the HotSync Manager configuration entries (see "HotSync Manager Configuration Entries" on page 188 in the *Introduction to Conduit Development*).

See Also `GetPath()` method.
`EPDPathType` constant.

SetStringData

Purpose Sets a `String` configuration entry value for the specified conduit.

Applies to `PDCondMgr`, `PDInstallConduit`, `PDSystemCondMgr` objects.

Prototype `PDCondMgr` and `PDSystemCondMgr`:

```
Sub SetStringData(CreatorID As Long, StringName As
    String, Data As String)
```

`PDInstallConduit`:

```
Sub SetStringData(UniqueId As Long, StringName As
    String, Data As String)
```

Parameters

→ *`CreatorID`*
If a `PDCondMgr` or `PDSystemCondMgr` object, this parameter is the creator ID of the conduit you want to set a value for.

→ *`UniqueId`*
If a `PDInstallConduit` object, this parameter is the unique ID of the install conduit you want to set a value for.

→ *`StringName`*
The name of the string configuration entry you want to set.

→ *`Data`*
The string value you want to set the specified entry to.

Returns None.

Errors

`eInvalidID`
The specified conduit creator ID is not valid.

`eNoSuchConduit`
The specified install conduit does not exist.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments This is a general purpose method that allows you to create or change a configuration entry by name. If the conduit you want is a standard synchronization conduit (most are), specify the creator ID in the first parameter. If the conduit you want is an **install conduit**, specify the unique ID in the first parameter.

This method sets information about a conduit that is registered either for the current Windows user or the system, depending on whether it is called for a `PDCondMgr` or a `PDSystemCondMgr` object.

Example

```
Dim CreatorId As Long
Dim strExtra As String
Const strTestValue = "Hello World"

Dim PCondMgr As New PDCondMgr
CreatorId = PCondMgr.StringToCreatorID("memo")

' Set the value for a custom filed called "ExtraString"
Call PCondMgr.SetStringData(CreatorId, "ExtraString", _
   strTestValue)
strExtra = PCondMgr.GetStringData(CreatorId, "ExtraString")
```

See Also `GetStringData()`, `GetConduitList()` methods

SetStringValue

Purpose Sets a string value to a key in the specified user's area of the users data store.

Applies to `PDUserData` object.

Prototype `Sub **SetStringValue**(*dwUserId* As Long, *Section* As String, *Key* As String, *value* As String)`

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`Section`*
The section name in the specified user's area of the users data store.

→ *`Key`*
The key of the string to set.

→ *`value`*
The new string value to set.

Returns None.

Errors

`eInvalidUser`
`dwUserId` is an invalid number.

`eOtherUDErr`
The attempt to write to the specified user's area of the users data store failed.

Comments Specifying `value` as `Null` deletes the key.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetStringValue()`, `DeleteKey()` methods

SetUserDirectory

Purpose Sets the directory name of the specified user ID.

Applies to `PDUserData` object.

Prototype

```
Sub SetUserDirectory(dwUserId As Long, Directory
    As String)
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`Directory`*
A string containing the user directory name to set.

Returns None.

Errors

`eIDInUse`
The specified user directory name is already in use by another user.

`eInvalidUser`
`dwUserId` is an invalid number.

`eInvalidUserDir`
`Directory` parameter is invalid.

`eNoCorePath`
No path for the users data store was found.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetUserDirectory()`, `GetRootDirectory()` methods

SetUserName

Purpose Sets the user name of the specified user ID.

Applies to `PDUserData` object.

Prototype

```
Sub SetUserName(dwUserId As Long, UserName As
    String)
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`UserName`*
A string containing the user name to set. It must be no more than 20 characters long.

Returns None.

Errors

`eInvalidUser`
`dwUserId` is an invalid number.

`eNoCorePath`
No path for the users data store was found.

`eNoUsers`
The users data store exists, but contains no information.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetUserNameFromID()`, `AddNewUser()` methods

SetUserPermanentSyncPreferences

Purpose Sets a conduit's permanent synchronization preferences for the specified user ID.

Applies to `PDUserData` object.

Prototype

```
Sub SetUserPermanentSyncPreferences(dwUserId As
    Long, ConduitCreatorId As Long, SyncAction As
    EPDUserSyncAction)
```

Parameters

→ *`dwUserId`*
: A unique ID to specify the user to reference in the users data store.

→ *`ConduitCreatorId`*
: The creator ID of the conduit you want to set the preferences of.

→ *`SyncAction`*
: The user's permanent synchronization preferences you want to set for this conduit, as a `EPDUserSyncAction` value.

Returns None.

Errors

`eInvalidUser`
: `dwUserId` is an invalid number.

`eNoCorePath`
: No path for the users data store was found.

`eOtherUDErr`
: No users data store was found or another method or program is accessing the user data store.

`eParamError`
: Parameters were not passed correctly.

`eUDSemaphoreError`
: Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetConduitList()`, `GetUserPermanentSyncPreferences()`, `DeleteUserPermanentSyncPreferences()`, `GetUserTemporarySyncPreferences()`, `SetUserTemporarySyncPreferences()`, `DeleteUserTemporarySyncPreferences()`, `RemoveUserTemporarySyncPreferences()` methods.
`EPDUserSyncAction` constant.

SetUserTemporarySyncPreferences

Purpose Sets a conduit's temporary synchronization preferences for the specified user ID.

Applies to `PDUserData` object.

Prototype

```
Sub SetUserTemporarySyncPreferences(dwUserId As
    Long, ConduitCreatorId As Long, SyncAction As
    EPDUserSyncAction)
```

Parameters

→ *`dwUserId`*
A unique ID to specify the user to reference in the users data store.

→ *`ConduitCreatorId`*
The creator ID of the conduit you want to set the preferences of.

→ *`SyncAction`*
The user's temporary synchronization preferences you want to set for this conduit, as a `EPDUserSyncAction` value.

Returns None.

Errors

`eInvalidUser`
`dwUserId` is an invalid number.

`eNoCorePath`
No path for the users data store was found.

`eOtherUDErr`
No users data store was found or another method or program is accessing the user data store.

`eParamError`
Parameters were not passed correctly.

`eUDSemaphoreError`
Another method or program is accessing the user data store.

`eUDUnableToCreate`
Creating a new users data store failed because of a file error.

See Also `GetUserList()`, `GetIDFromName()`, `GetIDFromPath()`, `GetConduitList()`, `GetUserTemporarySyncPreferences()`, `DeleteUserTemporarySyncPreferences()`, `RemoveUserTemporarySyncPreferences()`, `GetUserPermanentSyncPreferences()`, `SetUserPermanentSyncPreferences()`, `DeleteUserPermanentSyncPreferences()` methods.
`EPDUserSyncAction` constant.

StartHotSyncMgr

Purpose Starts the HotSync Manager application.

Applies to `PDHotSyncUtility` object.

Prototype `Sub StartHotSyncMgr(options As Long)`

Parameters → *`options`*
A long value that is all the OR-ed flags that specify how you want to start HotSync Manager. Use the constants described in “HotSync Manager Start Options Constants” on page 572.

Returns None.

Errors `eUnableToStart`
This method cannot start the HotSync Manager application.

Comments If HotSync Manager is already running, this method ignores the `options` flags and generates no error.

See Also `RefreshConduitInfo()`, `RestartHotSyncMgr()`, `TerminateHotSyncMgr()` methods

StringToCreatorID

Purpose Converts a `String` into a `DWORD` conduit creator ID.

Applies to `PDCondMgr`, `PDSystemCondMgr` objects.

Prototype

```
Function StringToCreatorID(strID As String) As
    Long
```

Parameters

→ *`strID`*
The creator ID (as a four-character string) that you want to convert.

Returns A creator ID as a `DWORD`.

Errors

`eInvalidID`
The specified conduit creator ID is not valid.

`eParamError`
Parameters were not passed correctly.

Comments Most other methods that take a creator ID need it as a `DWORD`.

Example

```
Dim CreatorID As Long
Dim strResult As String
Const strCreator = "memo"
Dim PDcond As New PDCondMgr

' Converted the string value to a Long and back again
CreatorID = PDcond.StringToCreatorID(strCreator)
strResult = PDcond.CreatorIDToString(CreatorID)
```

See Also `CreatorIDToString()` method

StringToRecordId

Purpose Converts a string (`BSTR`) to record ID.

Applies to `PDUtility` object.

Prototype `Function StringToRecordId(strId as String) as Variant`

Parameters ← *`strId`*
Record ID in a string.

Returns A record ID. This record ID can be used by other methods including `ReadById()` and `Write()`.

Comments This method converts a `Byte` array (read from the binary file that contains your record data) into record ID. You can use the returned record ID in methods including `ReadById()` and `Write()`.

Palm OS record IDs are long integers, but this may change. PalmSource, Inc. strongly recommend that you use `PDUtility` methods like these to convert record ID from/to `Byte` array/ `String` formats.

Example

```
Private pRemoteData As PDRecordAdapter
Dim pUtil as New PDUtility
Dim DbQuery as New PDDatabaseQuery
Dim strId as String
strId = 12345
Dim vRecordId as Variant
vRecordId = pUtil.StringToRecordId(strId)

Dim nIdx As Long
Dim vRecordId As Variant
Dim nCategory As Long
Dim eAttributes As ERecordAttributes
Dim pData As Variant
fill pData, nCategory, eAttributes here
pRemoteData.Write (vRecordId, nCategory, eAttributes, pData)
```

SwapDWORD

Purpose Swaps the bytes of an unsigned `Long`.

Applies to `PDUtility` object.

Prototype `Function SwapDWORD(nDWordVal as Long) as Long`

Parameters → *`nDWordVal`*
Unsigned long to swap.

Returns The swapped unsigned `Long`.

Example

```
Dim Utility As New PDUtility
Dim DWVal as long
DwVal = &H0A0B0C0D
DwVal = Utility.SwapDWORD(DwVal)
```

SwapWORD

Purpose Swaps the bytes of an unsigned `Integer`.

Applies to `PDUtility` object.

Prototype `Function SwapWORD(nWordVal as Integer) as Integer`

Parameters → *`nWordVal`*

Unsigned `Integer` to swap.

Returns The swapped unsigned `Integer`.

Example

```
Dim Utility As New PDUtility
Dim DwVal as Integer
DwVal = &H0A0B
DwVal = Utility.SwapWORD(DWVal)
```

SyncMgrAPIVersion

Purpose Retrieves the version of the Sync Manager API that is installed on the desktop computer.

Applies to `PDSystemAdapter` object.

Prototype

```
Sub SyncMgrAPIVersion(nVMajor as Integer, nVMinor
    as Integer)
```

Parameters

← *`nVMajor`*
Major version number.

← *`nVMinor`*
Minor version number.

Returns None.

Example

```
Dim pSystem as New PDSystemAdapter
Dim Vmajor as Integer, VMinor as Integer
PSystem.SyncMgrVersion (Vmajor, VMinor)
```

Tell

Purpose Gets the current position of the file pointer within an open file on an expansion card.

Applies to `PDVFSFileManager` object.

Prototype `Sub Tell(Position As Long)`

Parameters ← *`Position`*
The current position of the file pointer.

Returns None.

Errors

`eParamError`
Parameters were not passed correctly.

`eVFSFileBadRef`
The file reference number is invalid: it has been closed or was not obtained from `Open()`.

`eVFSInvalidOperation`
A file system is not present.

`eVFSIsADirectory`
This operation can be performed only on a regular file, not a directory.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
The volume reference number is invalid because, for example, the volume has not been mounted.

Comments This method operates only on files and cannot be used with directories.

See Also `Seek()`, `Read()`, `Write()`, `Open()` methods.

TerminateHotSyncMgr

Purpose	Closes the HotSync Manager application.
Applies to	`PDHotSyncUtility` object.
Prototype	`Sub` **`TerminateHotSyncMgr`**`();`
Parameters	None.
Returns	None.
Errors	`eUnableToClose` This method cannot close HotSync Manager.
Comments	If HotSync Manager is not running, this method generates no error.
See Also	`StartHotSyncMgr()`, `RestartHotSyncMgr()`, `RefreshConduitInfo()` methods

UnregisterConduit

Purpose Unregisters a conduit with HotSync Manager.

Applies to `PDCondMgr`, `PDSystemCondMgr` objects.

Prototype `Sub` **`UnregisterConduit`**`(`*`CreatorID`*` As Long)`

Parameters → *`CreatorID`*
The creator ID of the conduit you want to unregister.

Returns None.

Errors

`eAlreadyExists`
Another conduit is already registered with this creator ID.

`eInvalidID`
The specified conduit creator ID is not valid.

`eLocalMemory`
Not enough memory on the desktop to perform the requested operation.

`eNoSuchConduit`
The specified conduit does not exist.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments This method unregisters a conduit either for the current Windows user or the system, depending on whether it is called for a `PDCondMgr` or a `PDSystemCondMgr` object. For more information, see "User- and System-registered Conduits and Notifiers" on page 78 in *Introduction to Conduit Development*.

Example See the example under `RegisterConduit()`.

See Also `GetConduitList()`, `RegisterConduit()` methods

UnregisterIC

Purpose Unregisters an install conduit with HotSync Manager.

Applies to `PDInstallConduit` object.

Prototype `Sub **UnregisterIC**(*UniqueId* As Long)`

Parameters → *`UniqueId`*
The unique ID of the install conduit you want to unregister.

Returns None.

Errors

`eInvalidInstallID`
The specified unique ID is not valid.

`eNoSuchConduit`
The specified conduit does not exist.

`eRegistryFailure`
Unable to access the conduit configuration entries.

`eValueNotFound`
The specified value could not be found in the configuration entries for this conduit.

Comments The sequence for unregistering an install conduit is to call:

1. `TerminateHotSyncMgr()` to exit HotSync Manager.
2. `UnregisterIC()` to unregister your install conduit.
3. `StartHotSyncMgr()` to launch HotSync Manager.

Example

```
Dim PInstall As New PDInstallConduit
Dim PInfo As New PDInstallConduitInfo

PInfo.Directory = "Install"
PInfo.Extension = "All Files (*.*)|*.*"
PInfo.Module = "MyInstallConduit.dll"
PInfo.Name = "Test Install"
PInfo.UniqueId = 1952805748

Call PInstall.RegisterIC(PInfo)
Call PInstall.UnregisterIC(1952805748)
```

See Also `PDInstallConduitInfo` object.
`RegisterIC()`, `TerminateHotSyncMgr()`,
`StartHotSyncMgr()` methods.

UnregisterNotifier

Purpose Unregisters a notifier with HotSync Manager.

Applies to `PDCondMgr` object.

Prototype `Sub UnregisterNotifier(NotifierPath As String)`

Parameters → *`NotifierPath`*
The full path and filename of the notifier you want to unregister.

Returns None.

Errors `eNotifierNotFound`
The specified notifier is not registered.

`eParamError`
Parameters were not passed correctly.

`eRegistryFailure`
Unable to access the conduit configuration entries.

Comments This method does not delete the notifier DLL itself, only its registration entry with HotSync Manager.

The sequence for unregistering a notifier is to call:

1. `TerminateHotSyncMgr()` to exit HotSync Manager.
2. `UnregisterNotifier()` to unregister your notifier.
3. `StartHotSyncMgr()` to launch HotSync Manager.

Example

```
Dim PDCondMgr As New PDCondMgr

Call PDCondMgr.RegisterNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll")
Call PDCondMgr.ModifyNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll", "C:\PdN20d.dll")
Call PDCondMgr.UnregisterNotifier("C:\CDK403\C++\Samples\_
   PDNotify\Debug\PdN20d.dll")
```

See Also `RegisterNotifier()`, `ModifyNotifier()` methods

WORDToByteArray

Purpose Inserts an unsigned `Integer` into a `Byte` array.

Applies to `PDUtility` object.

Prototype

```
Function WORDToByteArray(pvData As Variant,
    nOffset As Long, bSwap As Boolean, nWordVal As
    Integer) As Long
```

Parameters

→ *`pvData`*
  `Byte` array used for insertion.

→ *`nOffset`*
  Offset location where the unsigned `Integer` is inserted.

→ *`bSwap`*
  If `True`, this method swaps the bytes in *`nWordVal`* before inserting them in *`pvData`*.

→ *`nWordVal`*
  Unsigned `Integer` to insert.

Returns A hexadecimal offset to the next byte in the array.

Example

```
Sub InsertWord(Record As Variant, Value As Integer)
   Dim Utility As New PDUtility
   Dim NextOffset As Long
   ' Insert the string in the array
   NextOffset = Utility.WORDToByteArray(Record, 0, Value)
End Sub
```

Write

Purpose Writes a record in a classic or extended database.

Applies to DmRecordAdapter, PDRecordAdapter, PDAddressDbHHRecordAdapter, PDDateBookDbHHRecordAdapter2, PDDateBookDbHHRecordAdapter, PDMemoDbHHRecordAdapter, PDTodoDbHHRecordAdapter objects.

Prototype DmRecordAdapter and PDRecordAdapter:

```
Sub Write(pvUniqueId, ByVal nCategory As Long,
    ByVal eAttributes As ERecordAttributes, ByVal
    vData)
```

PD<PIM>DbHHRecordAdapter:

```
Function Write(p<PIM>DbRecord as
    PD<PIM>DbHHRecord) as Variant
```

Parameters

↔ *pvUniqueId*
Requested ID. You can set the unique ID to `VbEmpty`, which causes the handheld to create a new record, or you can set the unique ID to an existing ID and the handheld overwrites the existing record. The `Write()` method always returns the unique ID for the record.

→ *nCategory*
Category.

→ *eAttributes*
Record attributes, which are combinations of `ERecordAttributes` constants.

→ *vData*
Data. Input array for database record data.

→ *p<PIM>DbRecord*
An object representing a record in one of the four standard Palm OS application databases (denoted `PD<PIM>DbHHRecord`). This is the record to write.

Returns For a DmRecordAdapter or PDRecordAdapter object, returns no value.

For any of the objects representing classic databases used by any of four standard Palm OS applications (denoted `PD<PIM>DbHHRecordAdapter`), this method returns a unique ID as a `Variant`.

Example For a DmRecordAdapter or PDRecordAdapter object:

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Write a new record
Dim vUniqueID as Variant
vUniqueId = VbEmpty
Dim bArray() as Byte
bArray = StrConv("This is a String", vbFromUnicode)
Adapter.Write(vUniqueId, 0, 0, bArray)
```

For a `PD<PIM>DbHHRecordAdapter`:

```
Dim pDbQuery As New PDDatabaseQuery
Dim PDateRecord As New PDDateBookDbHHRecord
Dim pDateAdapter As PDDateBookDbHHRecordAdapter
Set pDateAdapter = pDbQuery.OpenRecordDatabase("DatebookDB",_
   "PDStandard.PDDatebookDbHHRecordAdapter", eRead Or eWrite_
   Or eShowSecret)
' Fill in record data.
PDateRecord.Description = "Test Record"
PDateRecord.StartTime = "07/19/2002 9:00:00 AM"
PDateRecord.EndTime = "07/19/2002 9:15:00 AM"
' Write the record.
Dim uniqueid As Variant
uniqueid = pDateAdapter.Write(PDateRecord)
```

See Also DmRecordAdapter, PDRecordAdapter, PDAddressDbHHRecord, PDDateBookDbHHRecord, PDMemoDbHHRecord, PDTodoDbHHRecord objects.

Write

Purpose Writes data to an open file on an expansion card.

Applies to `PDVFSFileManager` object.

Prototype

```
Function Write(NumBytesToWrite As Long,
    Buffer As Variant) As Long
```

Parameters

→ *`NumBytesToWrite`*
The number of bytes to write.

→ *`Buffer`*
A `Variant` containing an array of bytes to write.

Returns The number of bytes (as a `Long`) that were actually written.

Errors

`eParamError`
The `Buffer` or `NumBytesToWrite` parameter is `Null`.

`eVFSFileBadRef`
The file reference number is invalid: it has been closed or was not obtained from `Open()`.

`eVFSFilePermissionDenied`
Permission denied to perform requested operation—for example, an attempt to write to a read-only file or to read a file already opened in the `eVFSModeExclusive` mode.

`eVFSInvalidOperation`
A file system is not present.

`eVFSIsADirectory`
This operation can be performed only on a regular file, not a directory.

`eVFSNoFileSystem`
None of the file systems installed on the handheld support this operation.

`eVFSNotOpen`
The file system library on the handheld necessary for this call has not been installed or has not been opened.

`eVFSVolumeBadRef`
The volume reference number is invalid because, for example, the volume has not been mounted.

`eVFSVolumeFull`
There is insufficient space left on the volume.

Comments This method operates only on files and cannot be used with directories.

See Also `Seek()`, `Tell()`, `Read()`, `Open()` methods.

WriteAppInfoBlock

Purpose Writes an application info block to an open classic or extended database on the handheld. The database must be opened for reading and writing.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype `Sub` **`WriteAppInfoBlock`**`(ByVal` *`vAppInfo`*`)`

Parameters → *`vAppinfo`*

Application info block. `Empty` deletes the application info block.

Comments When *`vAppinfo`* is a `Byte` array, this method writes that array to the record database's application info block. If *`vAppinfo`* is `Empty` (or a `Byte` array with zero length), then this method erases the application info block.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Read, then write the AppInfo block
Dim AppInfo as Variant
AppInfo = Adapter.ReadAppInfoBlock
Adapter.WriteAppInfoBlock AppInfo
```

WriteAppPreference

Purpose Writes an application's preference block.

Applies to PDSystemAdapter object.

Prototype

```
Sub WriteAppPreference(vCreator as Variant, nId as
    Long, bBackup as Boolean, nVersion as Integer,
    vPrefs as Variant)
```

Parameters

→ *vCreator*
: Creator ID. The unique ID associated with each database and application on the device. Each conduit is associated with a specific creator ID. It is four characters that can be in either `Long` (`VT_I4`) or Little Endian form.

→ *nId*
: Preference ID.

→ *bBackup*
: When `True`, this method writes to the Saved Preferences database. When `False`, this method writes to the UnSaved Preferences database.

→ *nVersion*
: The version number, as assigned by the application.

→ *vPrefs*
: The Preference record array to write. If this parameter is `Empty` (or a `Byte` array with zero length), then this method will erase the Application Preference data.

Returns None.

Example

```
Dim pSystem as New PDSystemAdapter
Dim vAppPref as Variant
Dim Version as Integer
' Read, then write the preference
vAppPref = PSystem.ReadAppPreference ("mail", 1, True, _
   Version)
pSystem.WriteAppPreference ("mail", 1, True, Version, _
   vAppPref)
```

See Also ReadAppPreference()

WriteColumnValue

Purpose Writes the specified bytes of a single column value to a row in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Sub WriteColumnValue(ByVal vRowID, ByVal ColumnID
   As Long, ByVal DataOffset As Long, pData)
```

Parameters

→ *`vRowID`*
The row ID of the row.

→ *`ColumnID`*
The column ID of the column value to write.

→ *`DataOffset`*
An offset from the first byte in a column value from which to start writing data.

→ *`pData`*
A `Variant` byte array that contains the bytes of the column value to write.

Returns None.

WriteColumnValues

Purpose Writes a set of column values to a row in a schema database.

Applies to `PSDRowAdapter` object.

Prototype

```
Sub WriteColumnValues(ByVal vRowID, PSDRowData As
    IPSDRowData)
```

Parameters

→ *`vRowID`*
The row ID of the row.

→ *`PSDRowData`*
A `PSDRowData` object that contains the column values to write.

Returns None.

WriteResource

Purpose Writes a resource to an open resource database on the handheld.

Applies to `PDResourceAdapter` object.

Prototype

```
Sub WriteResource(vType as Variant, nId as Long,
    vData as Variant)
```

Parameters

→ *vType*
Four-byte resource type that can be passed in either `Long` (`VT_I4`) or Little Endian form.

→ *nId*
Resource ID.

→ *vData*
`Byte` array containing the resource data.

Returns None.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Write a new resource
Dim bArray() as Byte
bArray = StrConv("This is a Resource", vbFromUnicode)
Adapter.WriteResource "Res1", 1, bArray
```

WriteSortInfoBlock

Purpose Writes a sort info block to an open classic or extended database on the handheld.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Prototype `Sub` **`WriteSortInfoBlock`**`(ByVal` *`vSortInfo`*`)`

Parameters → *`vSortinfo`*
Sort info block. `Empty` deletes the sort info block.

Returns None.

Comments When *`vSortinfo`* is a `Byte` array, this method writes that array to the record database's sort info block. If *`vSortinfo`* is `Empty` (or a `Byte` array with zero length), then this method erases the sort info block.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter as PDRecordAdapter
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Read, then write the sort info block
Dim SortInfo as Variant
SortInfo = Adapter.ReadSortInfoBlock
Adapter.WriteSortInfoBlock SortInfo
```

5

Properties

This chapter describes the COM Sync properties in alphabetical order.

AccessMode

Purpose Open database access mode.

Applies to DmRecordAdapter, PDRecordAdapter, PDResourceAdapter, PDAddressDbHHRecordAdapter, PDDateBookDbHHRecordAdapter2, PDDateBookDbHHRecordAdapter, PDMemoDbHHRecordAdapter, PDTodoDbHHRecordAdapter objects.

Accessibility Read-only.

Prototype `Property **AccessMode** as EAccessModes`

Comments Can be one or more values from the EAccessModes constants.

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", _
   eRead Or eWrite)
' Get the open access mode
Dim AccessMode As EAccessModes
AccessMode = Adapter.AccessMode
```

See Also EAccessModes constants.

Address

Purpose	Content of the “Address” field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property **Address** As String`

AlarmAdvanceTime

Purpose	How long before an event to trigger the alarm.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property **AlarmAdvanceTime** As Long`
Comments	This value is a unitless number; refer to the `AlarmAdvanceUnits` property to determine whether this value is in minutes, hours, or days. For example, if this property is set to 10 and `AlarmAdvanceUnits` is set to `PD_AAU_MINUTES`, then the alarm is triggered 10 minutes before the start time of the event.

AlarmAdvanceUnits

Purpose	Time units that the `AlarmAdvanceTime` property is specified in.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property **AlarmAdvanceUnits** As EPDAlarmAdvTimeUnits`
Comments	This property has one of the values defined by the `EPDAlarmAdvTimeUnits` enum: minutes, hours, or days.

AppInfoSize

Purpose Application info block size of this database.

Applies to DmDatabaseInfo, PDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property AppInfoSize as Long`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the AppInfo block size
Dim AppInfoSize as Long
AppInfoSize = DbInfo.AppInfoSize
```

Attributes

Purpose Flags that indicate the attributes of this schema database.

Applies to PSDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property Attributes As Long`

Parameters None.

Comments The value of this property a combination of the EPSDDatabaseFlags values.

Attributes

Purpose Attributes of a volume, file, or directory on an expansion card, such as whether it is read-only.

Applies to `PDVFSVolumeManager` and `PDVFSFileManager` objects.

Accessibility For a volume: Read-only.
For a file or directory: Read/write.

Prototype `Property Attributes As Long`

Comments For volumes, this property may have a value of one or more of the constants defined in "VFS Volume Attributes" on page 576. For files and directories, this property can be set to or have one or more of the constants defined in "VFS File and Directory Attributes" on page 574.

BackupDate

Purpose Date that this database was last backed up. Last backup date of this database.

Applies to `PSDDatabaseInfo`, `DmDatabaseInfo`, `PDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property BackupDate as Date`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the last backup date
Dim BackupDate as Date
BackupDate = DbInfo.BackupDate
```

BOF

Purpose The cursor has reached the beginning of this row set.

Applies to `PSDRowSet` object.

Accessibility Read/write.

Prototype `Property` **`BOF`** `As Boolean`

Parameters → *`ColID`*
The column ID of a column in this row.

CapabilityFlags

Purpose Describes the capabilities of an expansion card, such as whether it has storage and whether it is read-only.

Applies to `PDExpansionCardInfo` object.

Accessibility Read-only.

Prototype `Property` **`CapabilityFlags`** `As Long`

Comments This property can be set to one or more of the constants defined in "Hardware Capability Flags" on page 571.

CardName

Purpose Memory card name.

Applies to `PDMemoryCardInfo` object.

Accessibility Read-only

Prototype `Property` **`CardName`** `as String`

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the card name
Dim Name as String
Name = MemCard.CardName
```

CardNum

Purpose The number of the **memory card** on which the database is stored.

Applies to DmDatabaseInfo, PDMemoryCardInfo, PDHotsyncInfo, PDDatabaseInfo objects.

Accessibility Read-only

Prototype `Property CardNum as Long`

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the card number
Dim Number as Long
Number = MemCard.CardNum
```

CardVersion

Purpose Memory card version.

Applies to PDMemoryCardInfo object.

Accessibility Read-only.

Prototype `Property CardVersion as Integer`

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the card version
Dim Version as Integer
Version = MemCard.CardVersion
```

CategoryId

Purpose Category ID specified by category index.

Applies to `DmCategories`, `PDCategories`, `PDAddressDbHHRecord`, `PDMemoDbHHRecord`, `PDTodoDbHHRecord` objects.

Accessibility Read/write.

Prototype For `DmCategories` and `PDCategories` objects:

`Property` **`CategoryId`**`(ByVal` *`nIndex`* `As Long) As Long`

For `PD<PIM>DbHHRecordAdapter` objects:

`Property` **`CategoryId`** `As Long`

Parameters → *`nIndex`*

Category index.

Comments For a `DmCategories` or `PDCategories` object, this property is the category ID corresponding to the specified category index in this database.

For any of the objects representing records used by any of four standard Palm OS® application databases (denoted `PD<PIM>DbHHRecord`), this property is the category ID assigned to this record.

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead Or _
   eWrite)
' Get the categories object
Dim Categories as PDCategories
Set Categories = Adapter.PDCategories
' Do Something with Id's
Dim Idx as Long
For Idx = 0 to 15
   If Categories.CategoryId(Idx) <> 0 then
   ' Do something here
   End If
Next
```

CategoryIDList

Purpose	List of categories to which this row belongs.
Applies to	`PSDRowData` object.
Accessibility	Read/write.
Prototype	`Property` **`CategoryIDList`** `As Variant`
Parameters	None.
Comments	This property is a `Variant` array of category IDs that specifies this row's category memberships. Writing this property removes this row's existing category memberships and adds those that this property specifies. If you want to add memberships and retain the row's existing ones, call `AddCategoryMembership()` instead.

City

Purpose	Content of the "City" field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`City`** `As String`

CloseOptions

Purpose Update database dates on close.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Accessibility Read/write.

Prototype `Property **CloseOptions** as EUpdateDbDates`

Comments Permits the last-modified and last-backup dates to be updated. COM Sync automatically closes an open database when you call the last `Release` (in C++) on the associated adapter object or set the adapter object reference to `Nothing` (in Visual Basic). The `CloseOptions` property gives you the opportunity to change one or both of the dates on the database, before it is closed.

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", _
     eRead Or eWrite)
' Reset the last modified date on closing
Adapter.CloseOptions = eModifiedDate
```

See Also `EUpdateDbDates` constants

ColumnIDFromName

Purpose Column ID specified by column name.

Applies to `PSDRowData` object.

Accessibility Read-only.

Prototype

```
Property ColumnIDFromName(ByVal ColumnName As
    String) As Long
```

Parameters → *`ColumnName`*
The name of a column in this row.

Comments

ColumnNameFromID

Purpose Column name specified by column ID.

Applies to `PSDRowData` object.

Accessibility Read-only.

Prototype

```
Property ColumnNameFromID(ByVal ColumnID As Long)
    As String
```

Parameters → *`ColumnID`*
The column ID of a column in this row.

Comments

COMClassID

Purpose	ProgID of this COM-based conduit.
Applies to	`PDConduitInfo` object.
Accessibility	Read/write.
Prototype	`Property` **`COMClassID`** `as String`
Comments	If your conduit is an ActiveX server, this value is the notification object’s ProgID (also called the Programmatic ID)—for example, `SimpleDb.CNotify`. 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`. This property is *required* for all COM-based conduits; other conduits ignore it.

Company

Purpose	Content of the “Company” field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`Company`** `As String`

ConnectionType

Purpose An EConnectionType value that indicates the type transfer medium of the current HotSync operation.

Applies to PDHotsyncInfo object.

Accessibility Read-only.

Prototype `Property **ConnectionType** as EConnectionType`

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the connection type
Dim ConType as EConnectionType
ConType = HSInfo.ConnectionType
```

See Also EConnectionType constants.

Country

Purpose Content of the “Country” field in an Address Book record.

Applies to PDAddressDbHHRecord object.

Accessibility Read/write.

Prototype `Property **Country** As String`

CreateDate

Purpose Creation date of this database.

Applies to `DmDatabaseInfo`, `PDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property CreateDate as Date`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the creation date
Dim CreateDate as Date
CreateDate = DbInfo.CreateDate
```

CreationDate

Purpose Date on which this schema database was created.

Applies to `PSDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property CreationDate As Date`

Parameters None.

CreationDate

Purpose Memory card creation date, or if on an expansion card, the creation date for a file or directory.

Applies to `PDMemoryCardInfo` and `PDVFSFileManager` objects.

Accessibility For `PDMemoryCardInfo` objects: Read-only.
For `PDVFSFileManager` objects: Read/write.

Prototype `Property CreationDate as Date`

Comments For memory cards in a handheld's primary storage, this property is the creation date of the handheld's RAM memory card, which is where Palm OS databases are stored.

For expansion cards, this property is the creation date of this file or directory.

Example For `PDMemoryCardInfo` objects:

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the card creation date
Dim CreationDate as Date
CreationDate = MemCard.CreationDate
```

Creator

Purpose The **creator ID** associated with the current conduit or database.

Applies to `DmDatabaseInfo`, `PDDatabaseInfo`, `PDHotsyncInfo` objects.

Accessibility Read-only.

Prototype `Property` **`Creator`** `as Long`

Example

```
Dim pUtil as PDUtility
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the database creator ID
Dim CreatorId as String
CreatorId = pUtil.DwordToBSTR (DbInfo.Creator, False)
```

CreatorID

Purpose Creator ID of the application on the handheld that this conduit is responsible for synchronizing.

Applies to `PDConduitInfo` object.

Accessibility Read/write.

Prototype `Property` **`CreatorID`** `as Long`

Comments 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. HotSync Manager calls your conduit during synchronization only if an application (not just a database) with this creator ID exists on the handheld, unless your conduit opts out of this requirement (see `GetConduitInfo()`).

This property is *required* for all conduits and all versions of HotSync Manager.

The value of this property is a four-byte Palm OS **creator ID**. Use `CreatorIDToString()` to convert this value to the usual four-character representation of a creator ID.

CreatorID

Purpose	Creator ID of this schema database.
Applies to	`PSDDatabaseInfo` object.
Accessibility	Read-only.
Prototype	`Property` **`CreatorID`** `As Long`
Parameters	None.
Comments	The value of this property is a four-byte Palm OS **creator ID**. Use `DWORDToBSTR()` to convert this value to the usual four-character representation of a creator ID.

Custom1

Purpose	Content of the "Custom 1" field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`Custom1`** `As String`

Custom2

Purpose	Content of the "Custom 2" field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`Custom2`** `As String`

Custom3

Purpose Content of the “Custom 3” field in an Address Book record.

Applies to `PDAddressDbHHRecord` object.

Accessibility Read/write.

Prototype `Property Custom3 As String`

Custom4

Purpose Content of the “Custom 4” field in an Address Book record.

Applies to `PDAddressDbHHRecord` object.

Accessibility Read/write.

Prototype `Property Custom4 As String`

DataBytes

Purpose Number of bytes of storage used by this database for data only, excluding overhead.

Applies to `PSDDatabaseInfo`, `DmDatabaseInfo`, `PDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property DataBytes as Long`

Comments Contrast this property with `TotalBytes`.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the database data size
Dim DataBytes as Long
DataBytes = DbInfo.DataBytes
```

DataType

Purpose	Type of data stored in a column in a schema database.
Applies to	`PSDColumnInfo` object.
Accessibility	Read/write.
Prototype	`Property` **`DataType`** `As EPSDColumnDataType`
Parameters	None.
Comments	This property has one of the values defined by the `EPSDColumnDataType` enum.

DateTime

Purpose Current date and time on the handheld.

Applies to `PDSystemAdapter` object.

Accessibility Read/write.

Prototype `Property` **`DateTime`** `as Date`

Comments This property gets and sets the current system date and time on the handheld. In general, conduits should avoid changing the system date and time, because setting this property does not notify applications on the handheld that the time has changed. Some applications, such as PalmSource's Date Book, need to know when the system time changes so that they can adjust their alarm settings. To work around this problem, you need to call `PDSystemAdapter`.`RebootSystem()`, which causes a soft-reset of the handheld after the HotSync operation completes. All applications on the handheld are notified of the reset and can make any necessary adjustments.

IMPORTANT: In HotSync Manager versions 6.0 and later (Sync Manager versions 2.4 and later), setting the system time works only if the handheld is running Palm OS Cobalt. For Palm OS versions earlier than Palm OS Cobalt, setting the time returns an `E_NOTIMPL` error.

Example

```
Dim pSystem as New PDSystemAdapter
Dim dtNow as Date
dtNow = pSystem.DateTime
```

DaysMaskForWeeklyRepeat

Purpose Mask indicating which days of the week on which a weekly repeating event occurs in Date Book.

Applies to `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.

Accessibility Read/write.

Prototype `Property` **`DaysMaskForWeeklyRepeat`** `As Integer`

Comments This value of this property is a bitfield representation from Sunday through Saturday. Each bit set to 1 indicates that the event repeats on the corresponding day each week. When all bits are set to 1, the event repeats weekly on every day of the week.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Day	Sat	Fri	Thurs	Wed	Tues	Mon	Sun	All[1]

1. Set bit 0 and all other bits to indicate that the event repeats weekly on all days of the week.

If the `IsEventRepeatable` property is false, then the value of this property is not valid.

DbFlags

Purpose Database flags that are set at creation time. You can combine the database flag constants together to specify information about a database. Each flag indicates a property or condition of the database.

Applies to `DmDatabaseInfo`, `PDDatabaseInfo` objects.
`EDbFlags` constants.

Accessibility Read-only.

Prototype

```
Property DbFlags as Long
Property DbFlags as EDbflags
```

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the database flags
Dim DbFlags as EDbflags
DbFlags = DbInfo.DbFlags
```

DbIndex

Purpose Database index in the total set of classic databases.

Applies to PDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property DbIndex as Long`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the database Index
Dim DbIndex as Long
DbIndex = DbInfo.DbIndex
```

DbName

Purpose Name of this object's associated database on the handheld.

Applies to DmCategories, DmDatabaseInfo, DmRecordAdapter, PDDatabaseInfo, PDRecordAdapter, PDCategories, PDAddressDbHHRecordAdapter, PDDateBookDbHHRecordAdapter2, PDDateBookDbHHRecordAdapter, PDMemoDbHHRecordAdapter, PDTodoDbHHRecordAdapter objects.

Accessibility Read-only.

Prototype `Property DbName as String`

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead Or _
   eWrite)
' Check the database name
Dim DbName as String
DbName = Adapter.DbName
```

DbType

Purpose The **database type**.

Applies to `DmDatabaseInfo`, `PDDatabaseInfo`, `PDHotsyncInfo` objects.

Accessibility Read-only.

Prototype `Property DbType as Long`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the database Type
Dim DbType as String
DbType = DbInfo.DbType
```

Description

Purpose Text describing a Date Book event or a To Do List item.

Applies to `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord`, `PDTodoDbHHRecord` objects.

Accessibility Read/write.

Prototype `Property Description As String`

DeskTopDataDirectory

Purpose Name of this conduit's data directory.

Applies to `PDConduitInfo` object.

Accessibility Read/write.

Prototype `Property` **`DeskTopDataDirectory`** `as String`

Comments 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 the value to which you set this property back to your conduit when it calls `IPDClientNotify`.`BeginProcess()` method.

This property is optional for all conduits and all versions of HotSync Manager.

DeskTopDataFile

Purpose	Name of the desktop data file that your conduit synchronizes with the handheld database.
Applies to	`PDConduitInfo` object.
Accessibility	Read/write.
Prototype	`Property` **`DeskTopDataFile`** `as String`
Comments	You can write your conduit to synchronize with more than the one file specified here, however. You can use the `SetStringData()` and `GetStringData()` methods to create your own configuration entries in which to store additional desktop filenames. 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 `DeskTopDataDirectory` property. HotSync Manager passes the value to which you set this property back to your conduit when it calls the `IPDClientNotify`.`BeginProcess()` method. This property is optional for all conduits and all versions of HotSync Manager.

DeviceClass

Purpose	Describes the name of the type of expansion card.
Applies to	`PDExpansionCardInfo` object.
Accessibility	Read-only.
Prototype	`Property` **`DeviceClass`** `As String`
Comments	Examples of device class names are “Backup” and “Ethernet.”

DeviceUniqueId

Purpose Unique identifier for an expansion card product.

Applies to `PDExpansionCardInfo` object.

Accessibility Read-only.

Prototype `Property **DeviceUniqueId** As String`

Comments An example of the use of this property is as a serial number for the card. This value is set to the empty string ("") if no identifier exists.

Directory

Purpose Name of the install directory associated with this **install conduit**.

Applies to `PDInstallConduitInfo` object.

Accessibility Read/write.

Prototype `Property **Directory** as String`

Comments This is a subdirectory in the user's directory on the desktop computer. The Install Aide API copies files here to be installed during the next HotSync operation. For more information, see "Install Directory Terminology" on page 60 in the *COM Sync Suite Companion*.

This property is *required* for all install conduits.

Dirty

Purpose Category dirty flag specified by category index.

Applies to DmCategories, PDCategories objects.

Accessibility Read/write.

Prototype `Property Dirty(ByVal nIndex As Long) As Boolean`

Parameters → *nIndex*
Category index.

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead Or _
   eWrite)
' Get the categories object
Dim Categories as PDCategories
Set Categories = Adapter.PDCategories
' Do Something with dirty categories
Dim Idx as Integer
For Idx = 0 to 15
   If Categories.Dirty(Idx) then
    ' Do something here
    End If
Next
```

DisplayName

Purpose User-visible name of this conduit.

Applies to PDConduitInfo object.

Accessibility Read/write.

Prototype `Property DisplayName as String`

Comments 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).

This property is optional for all conduits and all versions of HotSync Manager.

DisplayName

Purpose Display name of this schema database.

Applies to `PSDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property **DisplayName** As String`

Parameters None.

Comments Palm OS Cobalt uses the display name of a database, if it is defined; otherwise, they use the internal name (defined by the `Name` property). Database names must consist of only 7-bit ASCII characters from 0x20 through 0x7E. The maximum length of a database name is 32 characters, which includes a terminator character managed by the COM Sync module.

DisplayPhone

Purpose Contact information to display in the Address Book list view.

Applies to `PDAddressDbHHRecord` object.

Accessibility Read/write.

Prototype `Property **DisplayPhone** As EPDDisplayPhone`

Comments This property is one of the `EPDDisplayPhone` values. For example, if you specify a `DisplayPhone` value of `EPDPhoneLabel1`, then the contact information that displays in the Address Book list view is that specified by the `EPDPhoneLabels` value stored in the `PhoneLabel1` property.

See Also `EPDDisplayPhone`, `EPDPhoneLabels`

DmCategories

Purpose Returns a `DmCategories` object representing the categories in this extended database.

Applies to `DmRecordAdapter` object.

Accessibility Read-only.

Prototype

```
Property DmCategories As DmCategories
```

Example

```
Dim DbQuery As New DmDatabaseQuery
Dim Adapter As DmRecordAdapter
Dim PDCondMgr As New PDCondMgr
Dim CreatorID As Long

' Convert creator ID string to a Long.
CreatorID = PDCondMgr.StringToCreatorID("MyCr")
' Open the your database
Set Adapter = DbQuery.OpenRecordDatabase("MyExtDatabase", _
   CreatorID, "DmConduit.DmRecordAdapter", eRead Or eWrite)
' Get the categories object
Dim Categories as DmCategories
Set Categories = Adapter.DmCategories
' Add a new category
Dim NewIndex As Integer
Do While Category.Name(NewIndex) <> ""
   NewIndex = NewIndex + 1
Loop
Category.Name(NewIndex) = "New Name"
Category.Id(NewIndex) = Category.LastId
Category.LastId = Category.LastId + 1
Category.Save
```

See Also `DmCategories` object.

DmDatabaseInfo

Purpose Returns a DmDatabaseInfo object representing information about this extended database.

Applies to DmDatabaseQuery, DmRecordAdapter objects.

Accessibility Read-only.

Prototype `Property DmDatabaseInfo As DmDatabaseInfo`

Example

```
Dim DbQuery as New DmDatabaseQuery
Dim Adapter As DmRecordAdapter
Dim PDCondMgr As New PDCondMgr
Dim CreatorID As Long

' Convert creator ID string to a Long.
CreatorID = PDCondMgr.StringToCreatorID(“MyCr”)
' Open your database
Set Adapter = DbQuery.OpenRecordDatabase("MyExtDatabase", _
   CreatorID, “DmConduit.DmRecordAdapter”, eRead Or eWrite)
' Get the database information object
Dim DbInfo as DmDatabaseInfo
Set DbInfo = Adapter.DmDatabaseInfo
```

See Also DmDatabaseInfo object.

DueDate

Purpose Due date of a To Do List item.

Applies to PDTodoDbHHRecord object.

Accessibility Read/write.

Prototype `Property DueDate As Date`

Dynamic

Purpose	Flag that indicates whether a column in a schema is dynamic.
Applies to	`PSDColumnInfo` object.
Accessibility	Read/write.
Prototype	`Property` **`Dynamic`** `As Boolean`
Parameters	None.
Comments	If `True`, the column is dynamic; if `False`, it is not.

Encoding

Purpose	Type of character encoding of text data in a schema database.
Applies to	`PSDDatabaseInfo` object.
Accessibility	Read-only.
Prototype	`Property` **`Encoding`** `As EPSDEncodingType`
Parameters	None.
Comments	This property has one of the `EPSDEncodingType` values.

EndTime

Purpose	Time and date on which an event ends in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`EndTime`** `As Date`
Comments	The `PDDateBookDbHHRecord` object cannot handle event end times earlier than 12/31/1969 4:00:00 PM, which is the earliest date supported by the Date Book application.

EOF

Purpose Database iterator is at the end of the database; for a file on an expansion card, the file pointer has reached the end of the file; for a set of rows in a schema database, the cursor is at the end of the row set.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDResourceAdapter`, `PDVFSFileManager`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter`, `PSDRowSet` objects.

Accessibility Read-only.

Prototype `Property` **`EOF`** `as Boolean`

Comments For a database in a handheld's primary storage, this property is `True` when the database iterator has reached the end of the database. For a file on expansion cards, this property is `True` when the file pointer has reached the end of the file (this property is not valid for directories). For a set of rows in a schema database, this property is `True` when the cursor has reached the end of the row set in a schema database.

Example For `DmRecordAdapter`, `PDRecordAdapter`, and `PDResourceAdapter` objects:

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter

' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead _
   Or eWrite)

Dim Index As Long
Dim UniqueId As Long
Dim Category As Byte
Dim Attributes As Byte
Dim Data As Variant

' Set the iteration index.
Adapter.IterationIndex = 10
```

```
' Read the next unfiled record
Category = 0
Data = Adapter.ReadNextInCategory(Index, _
   UniqueId, Category, Attributes)

' Read another if it's not at EOF
If Not Adapter.EOF Then
    Data = Adapter.ReadNextInCategory(Index, _
        UniqueId, Category, Attributes)
End If
```

ExcludeFromSync

Purpose Determines whether this database is excluded from synchronization.

Applies to `DmDatabaseInfo`, `PDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property **ExcludeFromSync** as Boolean`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Has this database been excluded from sync?
Dim Exclude as Boolean
Exclude = DbInfo.ExcludeFromSync
' Do something if its not excluded
If not Exclude then
Endif
```

Extension

Purpose The file type extensions of the files that this install conduit can install.

Applies to `PDInstallConduitInfo` object.

Accessibility Read/write.

Prototype `Property **Extension** as String`

Comments This string is in the standard Windows `CFileDialog` format—for example,

```
Palm Applications(*.prc)|*.prc|Palm Databases
(*.pdb)|*.pdb|Palm Query Application (*.pqa)|*.pqa
```

This property is *required* for all install conduits.

FileName

Purpose	Filename of this conduit DLL.
Applies to	`PDConduitInfo` object.
Accessibility	Read/write.
Prototype	`Property **FileName** as String`
Comments	This is the filename of the DLL that HotSync Manager loads to run this conduit. If this entry is only a filename, the DLL must be in the HotSync Manager directory or in the current Windows `PATH`. If it is a path and filename, you can put the DLL in any directory.

If this conduit is a C API-based conduit, you must set this property to the filename of the conduit DLL you created with the C/C++ Sync Suite.

If this conduit is a COM-based conduit, you must set this property to `COMConduit.dll` so that the COM Sync module is loaded when your conduit needs to run.

If this conduit is a Java-based conduit, you must set this property 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. For more information, see the *JSync Suite Companion*.

FileSystemType

Purpose	Type of file system on this volume on an expansion card.
Applies to	`PDVFSVolumeManager` object.
Accessibility	Read-only.
Prototype	`Property **FileSystemType** As EPDVFSFileSystemType`
Comments	This property is one of the `EPDVFSFileSystemType` constants defined in "EPDVFSFileSystemType" on page 554.

FirstName

Purpose Content of the “First name” field in an Address Book record.

Applies to `PDAddressDbHHRecord` object.

Accessibility Read/write.

Prototype `Property FirstName As String`

FirstSync

Purpose An `EFirstSync` value that indicates whether the current HotSync operation is the first for the handheld, the first with the current desktop, or the first for neither.

Applies to `PDHotsyncInfo` object.

Accessibility Read-only.

Prototype `Property FirstSync as EFirstSync`

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the first sync flag
Dim FirstSync as EFirstSync
FirstSync = HSInfo.FirstSync
```

Comments When a synchronization is initiated, HotSync manager will detect a *first sync* on the handheld or desktop and set this property prior to calling a conduit.

See Also `EFirstSync` constants.

Flags

Purpose Flags that indicate whether this schema database is excluded from HotSync operations and whether it is in RAM on the handheld.

Applies to `PSDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property **Flags** As Long`

Parameters None.

Comments This property can have one or more of the values defined in "Database Information Flags" on page 534.

FreeRamSize

Purpose Amount of available RAM on the card in bytes.

Applies to `PDMemoryCardInfo` object.

Accessibility Read-only.

Prototype `Property **FreeRamSize** as Long`

Example

```
Dim pSystem as New PDSystemAdapter
Dim memCard as PDMemoryCardInfo
Set memCard = pSystem.PDMemoryCardInfo
' Get the free RAM size
Dim FreeRam as Long
FreeRam = MemCard.FreeRamSize
```

HandHeldDB

Purpose	Name of the database on the handheld that this conduit accesses.
Applies to	`PDConduitInfo` object.
Accessibility	Read/write.
Prototype	`Property **HandHeldDB** as String`
Comments	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 the value to which you set this property back to your conduit when it calls the `IPDClientNotify`.`BeginProcess()` method.

This property is optional for all conduits and all versions of HotSync Manager.

ID

Purpose	Column ID of a column in a schema.
Applies to	`PSDColumnInfo` object.
Accessibility	Read/write.
Prototype	`Property **ID** As Long`
Parameters	None.

Index

Purpose Position of this record in its PIM database.

Applies to `PDAddressDbHHRecord`, `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord`, `PDMemoDbHHRecord`, `PDTodoDbHHRecord` objects.

Accessibility Read-only.

Prototype `Property **Index** As Long`

Comments Though this property is read/write, you should not write this property. Always let the handheld control its value.

This index also indicates the current sort order of this record in the database.

InputBufferSize

Purpose Size of the buffer to allocate to read classic record or resource database data or extended database data. All methods that read classic or extended databases use this property.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDResourceAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Accessibility Read/write.

Prototype `Property **InputBufferSize** as Long`

Comments This property specifies the maximum record size in bytes to be read. This property is used by all of the methods that read classic or extended database records.

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead Or _
   eWrite)
Dim Index As Long
Dim UniqueId As Variant
Dim Category As Long
Dim Attributes As Long
Dim Data As Variant
' Read a very large record
Adapter.InputBufferSize = 64000
Data = Adapter.ReadByIndex(0, UniqueId, Category, Attributes)
```

IsAlarmSet

Purpose	Indicates whether the alarm is set for this event in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property **IsAlarmSet** As Boolean`
Comments	If `True`, the alarm is set; if `False`, it is not set.

IsArchived

Purpose	Indicates whether a `PD<PIM>DbHHRecord` record is marked to be archived.
Applies to	`PDAddressDbHHRecord`, `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord`, `PDMemoDbHHRecord`, `PDTodoDbHHRecord` objects.
Accessibility	Read/write.
Prototype	`Property **IsArchived** As Boolean`
Comments	If `True`, this record's archive bit is set. If `False`, the archive bit is clear.

IsCompleted

Purpose	Indicates whether a To Do List item is completed.
Applies to	`PDTodoDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property **IsCompleted** As Long`
Comments	If `True`, this item is completed. If `False`, it is not. Note that this property is stored as a `Long`, not a `Boolean`.

IsDataPresent

Purpose	Flag that indicates whether a column in this row contains valid data.
Applies to	`PSDRowData` object.
Accessibility	Read-only.
Prototype	`Property **IsDataPresent**(ByVal *ColumnID* As Long) As Boolean`
Parameters	→ *`ColumnID`* The column ID of a column in this row.
Comments	If `True`, the specified column in this row contains data; if `False`, it does not.

IsDeleted

Purpose	Indicates whether a `PD<PIM>DbHHRecord` record is marked to be deleted.
Applies to	`PDAddressDbHHRecord`, `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord`, `PDMemoDbHHRecord`, `PDTodoDbHHRecord` objects.
Accessibility	Read/write.
Prototype	`Property **IsDeleted** As Boolean`
Comments	If `True`, this record's delete bit is set. If `False`, the delete bit is clear.

IsDirty

Purpose	Indicates whether a `PD<PIM>DbHHRecord` record is has been modified since the last synchronization.
Applies to	`PDAddressDbHHRecord`, `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord`, `PDMemoDbHHRecord`, `PDTodoDbHHRecord` objects.
Accessibility	Read/write.
Prototype	`Property **IsDirty** As Boolean`
Comments	If `True`, this record's dirty bit is set, indicating that it has been modified. If `False`, the dirty bit is clear.

IsEventNotTimed

Purpose	Indicates whether a time is specified for this event in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`IsEventNotTimed`** `As Boolean`
Comments	If `True`, no time is specified (Date Book ignores the values of the `StartTime` and `EndTime` properties); if `False`, the values of the those properties are used to time the event.

IsEventRepeatable

Purpose	Indicates whether this event repeats in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`IsEventRepeatable`** `As Boolean`
Comments	If `True`, this is a repeating event (DateBook uses the values of the `Repeat`* properties); if `False`, the values of these properties are invalid.
See Also	`RepeatDay`, `RepeatEndDate`, `RepeatFrequency`, `RepeatType` properties.

IsPrivate

Purpose	Flag that indicates whether this row is marked private.
Applies to	`PSDRowData` object.
Accessibility	Read/write.
Prototype	`Property` **`IsPrivate`** `As Boolean`
Parameters	None.
Comments	If `True`, the this row is marked private; if `False`, it is not.

IsPrivate

Purpose Indicates whether a `PD<PIM>DbHHRecord` record is marked as private.

Applies to `PDAddressDbHHRecord`, `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord`, `PDMemoDbHHRecord`, `PDTodoDbHHRecord` objects.

Accessibility Read/write.

Prototype `Property IsPrivate As Boolean`

Comments If `True`, this record's secret bit is set, indicating that it is private. If `False`, the secret bit is clear.

IsRam

Purpose Determines whether a database is stored in RAM or ROM.

Applies to `DmDatabaseInfo`, `PDDatabaseInfo` object.

Accessibility Read-only.

Comments Returns `True` if database is stored in RAM, returns `False` if stored in ROM.

Prototype `Property IsRam as Boolean`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' RAM or ROM database
if DbInfo.IsRam Then
    ' do something
EndIf
```

IsReadOnly

Purpose	Flag that indicates whether this row is marked read-only.
Applies to	`PSDRowData` object.
Accessibility	Read/write.
Prototype	`Property` **`IsPrivate`** `As Boolean`
Parameters	None.
Comments	If `True`, this row is marked read-only; if `False`, it is not. However, if a column's `WritableExceptionInReadOnlyRows` property is `True`, then its column value can be modified even if the row's `IsReadOnly` property is `True`.

IsReadOnlyDatabase

Purpose	Flag that indicates whether this schema database is read-only.
Applies to	`PSDDatabaseInfo` object.
Accessibility	Read-only.
Prototype	`Property` **`IsReadOnlyDatabase`** `As Boolean`
Parameters	None.
Comments	If `True`, this schema database is read-only. If `False`, it is read-write.

IsSecureDatabase

Purpose	Flag that indicates whether this schema database is secure.
Applies to	`PSDDatabaseInfo` object.
Accessibility	Read-only.
Prototype	`Property` **`IsSecureDatabase`** `As Boolean`
Parameters	None.
Comments	If `True`, this schema database is secure. If `False`, it is not.

IterationIndex

Purpose Current starting index for the record/resource data iteration methods.

Applies to `DmRecordAdapter`, `PDRecordAdapter`, `PDResourceAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Accessibility Read/write.

Prototype `Property IterationIndex as Long`

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase ("MemoDB", _
   eRead Or eWrite)
Dim Index As Long
Dim UniqueId As Variant
Dim Category As Long
Dim Attributes As Long
Dim Data As Variant
' Set the iteration index to the first record
Adapter.IterationIndex = 0
' Read the next modified record
Data = Adapter.ReadNextModified(Index, UniqueId, Category, _
   Attributes)
```

JavaClassName

Purpose Full name of the Java-based conduit class (including package).

Applies to `PDConduitInfo` object.

Accessibility Read/write.

Prototype `Property JavaClassName as String`

Comments This property is *required* for all Java-based conduits; other conduits (including all COM-based conduits) ignore it.

JavaClassPath

Purpose Directory that contains all the classes used by this Java-based conduit.

Applies to `PDConduitInfo` object.

Accessibility Read/write.

Prototype `Property **JavaClassPath** as String`

Comments This is the value of `CLASSPATH` required to find all the classes invoked by a 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. (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`).

This property is *required* for all Java-based conduits; other conduits (including all COM-based conduits) ignore it.

Label

Purpose Label of this volume on an expansion card.

Applies to `PDVFSVolumeManager` object.

Accessibility Read/write.

Prototype `Property` **`Label`** `As String`

Comments Volume reference numbers can change each time the handheld mounts a given volume. To keep track of a particular volume from one HotSync operation to the next, save the volume's label rather than its reference number. Volume labels can be up to 255 characters long. They can contain any normal character, including spaces and lowercase characters, in any character set as well as the following special characters: $ % ' - _ @ ~ ` ! ( ) ^ # & + , ; = [ ]. See "Naming Volumes" on page 95 in the *COM Sync Suite Companion* for guidelines on naming.

NOTE: Most conduits should not need to set this property. Setting this property may create or delete a file in the root directory, which would invalidate any current calls to `GetFileList()`.

LastAccessedDate

Purpose Last accessed date of a file or a directory on an expansion card.

Applies to `PDVFSFileManager` object.

Accessibility Read/write.

Prototype `Property` **`LastAccessedDate`** `As Date`

LastId

Purpose Category ID of the last new category.

Applies to DmCategories, PDCategories objects.

Accessibility Read/write.

Prototype `Property` **`LastId`** `as Byte`

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead Or _
   eWrite)
' Get the categories object
Dim Categories as PDCategories
Set Categories = Adapter.PDCategories
' Add a new category
Dim Idx as Integer
For Idx = 0 to 15
   If Categories.Name(Idx) = "" then
      Categories.Name(Idx) = "New Category"
      Categories.Dirty(Idx) = True
      Categories.Id(Idx) = Categories.LastId
      Categories.LastId = Categories.LastId + 1
      Categories.Save
      ' All done
      Exit For
   End If
Next
```

LastModificationDate

Purpose Last modification date of a file or a directory on an expansion card.

Applies to PDVFSFileManager object.

Accessibility Read/write.

Prototype `Property` **`LastModificationDate`** `As Date`

LastName

Purpose	Content of the "Last name" field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property LastName As String`

LastSyncDate

Purpose	Last synchronization date.
Applies to	`PDUserInfo` object.
Accessibility	Read-only.
Prototype	`Property LastSyncDate as Date`

Example

```
Dim pSystem as PDSystemAdapter
Dim UserInfo as PDUserInfo
' Get the user info object
Set UserInfo = pSystem.PDUserInfo
' Get the Last sync date
Dim LastSyncDate as Date
LastSyncDate = UserInfo.LastSyncDate
```

LastSyncPC

Purpose	ID assigned by HotSync Manager of the last PC that was synchronized with this handheld.
Applies to	`PDUserInfo` object.
Accessibility	Read-only.
Prototype	`Property LastSyncPC as Long`

Example

```
Dim pSystem as PDSystemAdapter
Dim UserInfo as PDUserInfo
' Get the user info object
Set UserInfo = pSystem.PDUserInfo
' Get the Last PC to sync with this handheld
Dim LastSyncPC as Long
LastSyncPC = UserInfo.LastSyncPC
```

LocalizationId

Purpose Localization ID, currently unused.

Applies to PDSystemAdapter object.

Accessibility Read-only.

Prototype Property **LocalizationId** as Long

Example

```
Dim pSystem as New PDSystemAdapter
Dim nLocalId as Long
nLocalId = pSystem.LocalizationId
```

LocalName

Purpose The desktop file that the conduit synchronizes with. This value is set in the conduit's File configuration entry.

Applies to PDHotsyncInfo object.

Accessibility Read-only.

Prototype Property **LocalName** as String

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the local (PC) filename
Dim LocalName as String
LocalName = HSInfo.LocalName
```

ManufacturerName

Purpose Name of the manufacturer of the expansion card.

Applies to PDExpansionCardInfo object.

Accessibility Read-only.

Prototype Property **ManufacturerName** As String

ManufName

Purpose Memory card manufacturer's name.

Applies to PDMemoryCardInfo object.

Accessibility Read-only.

Prototype `Property **ManufName** as String`

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the card manufacturer name
Dim Name as String
Name = MemCard.ManufName
```

Mask

Purpose Unique bit mask value associated with this **install conduit**.

Applies to PDInstallConduitInfo object.

Accessibility Read/write.

Prototype `Property **Mask** as String`

Comments HotSync Manager uses this mask value to identify your install conduit. Your installer is responsible for ensuring that this value is unique. See "Registering an Install Conduit" on page 66 in the *COM Sync Suite Companion*.

This property is *required* for all install conduits.

MaxAllowedRecordSize

Purpose Size in bytes of the largest record allowed in a classic or extended database on the handheld.

Applies to `DmDatabaseQuery`, `PDDatabaseQuery` object.

Accessibility Read-only.

Prototype `Property` **`MaxAllowedRecordSize`** `as Long`

Comments Table 5.1 lists the maximum record size supported by versions of Palm OS.

Table 5.1 Maximum record size for non-schema databases

Database Type	Maximum Record Size (bytes)
Extended	2^{26} – 16 (~64 MB)
Classic	65,505 for Palm OS versions 3.0 and later
	64,720 for Palm OS versions earlier than 3.0

NOTE: Sync Manager versions 2.4 and later return only the value 65,505 for classic databases regardless of the version of Palm OS on the handheld.

Example

```
Dim DbQuery as New PDDatabaseQuery
' Get the Max allowed record size
Dim MaxRecSize as Long
MaxRecSize = DbQuery.MaxAllowedRecordSize
```

MaxRecordSize

Purpose Size of the largest record in this database.

Applies to DmDatabaseInfo, PDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property **MaxRecordSize** as Long`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the largest record size
Dim MaxRecordSize as Long
MaxRecordSize = DbInfo.MaxRecordSize
```

MaxSize

Purpose Maximum size of a column in a schema.

Applies to PSDColumnInfo object.

Accessibility Read/write.

Prototype `Property **MaxSize** As Long`

Parameters None.

Comments The value of this property is not valid for columns of fixed-size data types—for example, `PSDInt32`.

MediaType

Purpose Type of media supported by the expansion card.

Applies to `PDExpansionCardInfo` and `PDVFSVolumeManager` objects.

Accessibility Read-only.

Prototype `Property **MediaType** As Long`

Comments This property may have a value of one of the constants defined in "VFS Manager and Expansion Manager Media Type Constants" on page 575. Because this read-only property returns a value of type `Long`, you can use the `PDCondMgr`.`CreatorIDToString()` method to convert it to one of the string values defined in that section.

These values specify whether the supported media type is any or only one of several, such as Secure Digital, CompactFlash, or others.

Example

```
Private Function GetMediaType() As String

   Dim ExpansionSlot As New PDExpansionManager
   Dim CardInfo As PDExpansionCardInfo
   Dim IsCardPresent As Boolean
   Dim IsVolumeMounted As Boolean
   Dim VolumeRef As Long
   Dim SlotNumbers As Variant
   Dim Convert As New PDCondMgr
   Dim i As Integer
   Dim strMediaType As String

   If ExpansionSlot.IsExpansionSlotPresent Then
      ' Get a list of all available slots
      SlotNumbers = ExpansionSlot.GetSlotReferenceNumbers

      For i = 0 To UBound(SlotNumbers)
         Call ExpansionSlot.GetSlotInfo(SlotNumbers(i), _
            IsCardPresent, IsVolumeMounted, VolumeRef)
         If IsCardPresent Then Exit For
      Next i

      ' Inform the user that none of the found slots
      ' contained a card.
      If Not IsCardPresent Then
         MsgBox "No expansion card present.", vbInformation,_
            "Information"
         Exit Function
```

```
            End If

            ' Retrieve card information.
            Set CardInfo = _
               ExpansionSlot.GetCardInfo(SlotNumbers(i))

            ' Convert the MediaType to string.
            strMediaType = _
               Convert.CreatorIDToString(CardInfo.MediaType)

            Select Case strMediaType
               Case "sdig"
                  strMediaType = "Secure Digital"
               Case "mstk"
                  strMediaType = "Memory Stick"
               Case "cfsh"
                  strMediaType = "Compact Flash"
               Case "mmcd"
                  strMediaType = "Multimedia Card"
               Case "smed"
                  strMediaType = "Smart Media"
               Case "ramd"
                  strMediaType = "RAM Disk"
               Case "pose"
                  strMediaType = "Palm OS Emulator"
               Case "pnps"
                  strMediaType = "Universal PnP"
               Case Else
                  strMediaType = "Unknown"
            End Select

            ' Assign return value.
            GetMediaType = strMediaType
         End If

      End Function
```

Memo

Purpose	Content of a Memo Pad record.
Applies to	`PDMemoDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`Memo`** `As String`

ModCount

Purpose	Database modification count.
Applies to	`DmDatabaseInfo`, `PDDatabaseInfo` object.
Accessibility	Read-only.
Prototype	`Property` **`ModCount`** `as Long`
Comments	This value is incremented every time a record in the database is added, modified, or deleted on the handheld.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the modification count
Dim ModCount as Long
ModCount = DbInfo.ModCount
```

ModDate

Purpose Last modification date.

Applies to DmDatabaseInfo, PDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property **ModDate** as Date`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the last modification date
Dim ModDate as Date
ModDate = DbInfo.ModDate
```

ModifyDate

Purpose Date on which this schema database was most recently modified.

Applies to PSDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property **ModifyDate** As Date`

Parameters None.

ModifyNumber

Purpose The database modification number, which is incremented every time a row in this schema database is added, modified, or deleted on the handheld.

Applies to PSDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property **ModifyNumber** As Long`

Parameters None.

Module

Purpose Filename of this install conduit.

Applies to `PDInstallConduitInfo` object.

Accessibility Read/write.

Prototype `Property **Module** as String`

Comments This property specifies the filename of this 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. This property is *required* for all install conduits and all versions of HotSync Manager.

mountClass

Purpose Mount class of the file system driver that mounted this volume on an expansion card.

Applies to `PDVFSVolumeManager` object.

Accessibility Read-only.

Prototype `Property **mountClass** As Long`

Comments This property is set to one of the constants described in "VFS Volume Mount Class Constants" on page 577. Because this read-only property returns a value of type `Long`, you can use the `PDCondMgr`.`CreatorIDToString()` method to convert it to one of the string values defined in that section.

Name

Purpose	User-visible name of this install conduit.
Applies to	`PDInstallConduitInfo` object.
Accessibility	Read/write.
Prototype	`Property` **`Name`** `as String`
Comments	HotSync Manager displays this string as the name of an install conduit. If you do not set this entry, HotSync Manager shows the name your conduit provides when called (by `GetConduitInfo` or `GetConduitName` in C/C++ Sync, `Conduit::name()` in JSync, or `IPDClientNotify->GetConduitInfo()` in COM Sync). This property is optional for all install conduits and all versions of HotSync Manager.

Name

Purpose Category name specified by category index.

Applies to `DmCategories`, `PDCategories` objects.

Accessibility Read/write.

Prototype `Property` **`Name`**`(ByVal` *`nIndex`* `As Long) As String`

Parameters → *`nIndex`*

Category index.

Errors `eDuplicateName`

The specified category name already exists.

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead Or _
      eWrite)
' Get the categories object
Dim Categories as PDCategories
Set Categories = Adapter.PDCategories
' Add a new category
Dim Idx as Integer
For Idx = 0 to 15
    If Categories.Name(Idx) = "" then
        Categories.Name(Idx) = "New Category"
        Categories.Dirty(Idx) = True
        Categories.Id(Idx) = Categories.LastId
        Categories.LastId = Categories.LastId + 1
        Categories.Save
        ' All done
        Exit For
    End If
Next
```

Name

Purpose Category name specified by category ID in a schema database.

Applies to `PSDCategoryAdapter` object.

Accessibility Read-only.

Prototype `Property Name(ByVal CategoryID As Long) As String`

Parameters → *`CategoryID`*
Specifies the category ID of the category.

Name

Purpose Name of a column in a schema.

Applies to `PSDColumnInfo` object.

Accessibility Read/write.

Prototype `Property Name As String`

Parameters None.

Name

Purpose Internal name of this schema database.

Applies to `PSDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property Name As String`

Parameters None.

Comments Palm OS Cobalt uses the display name of a schema database (defined by the `DisplayName` property), if it is defined; otherwise, they use the internal name defined by this property. Database names must consist of only 7-bit ASCII characters from 0x20 through 0x7E. The maximum length of a database name is 32 characters, which includes a terminator character managed by the COM Sync module.

Name

Purpose	Name of this table in a schema database.
Applies to	`PSDTable` object.
Accessibility	Read/write.
Prototype	`Property Name As String`
Parameters	None.

NameList

Purpose	List of the handheld databases that have the same creator ID as the current conduit. The number of items in the array is specified by the `RemoteNameCount` property.
Applies to	`PDHotsyncInfo` object.
Accessibility	Read-only.
Prototype	`Property NameList as Variant`

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the database name list to synchronize
Dim NameList as Variant
NameList = HSInfo.NameList
' Loop and process
Dim Idx as Integer
For Idx = 0 to HSInfo.RemoteNameCount
    ' Do something with NameList(Idx)
Next
```

NonSyncable

Purpose	Flag that indicates whether the data in a column is to be synchronized.
Applies to	`PSDColumnInfo` object.
Accessibility	Read/write.
Prototype	`Property **NonSyncable** As Boolean`
Parameters	None.
Comments	If `True`, the column data is *not* to be synchronized; if `False`, it is to be synchronized. The Data Manager on the handheld does not track changes to data in a nonsyncable column in a schema database.

Notes

Purpose	Content of the note in an Address Book, Date Book, or To Do List record.
Applies to	`PDAddressDbHHRecord`, `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord`, `PDTodoDbHHRecord` objects.
Accessibility	Read/write.
Prototype	`Property **Notes** As String`

Password

Purpose Encrypted handheld password.

Applies to PDUserInfo object.

Accessibility Read-only.

Prototype `Property Password as String`

Example

```
Dim pSystem as PDSystemAdapter
Dim UserInfo as PDUserInfo
' Get the user info object
Set UserInfo = pSystem.PDUserInfo
' Get the password
Dim password as String
Password = UserInfo.Password
```

PathName

Purpose The conduit's directory name. This value is set in the conduit's Directory configuration entry.

Applies to PDHotsyncInfo object.

Accessibility Read-only.

Prototype `Property PathName as String`

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the local (PC) path name
Dim PathName as String
PathName = HSInfo.PathName
```

PDCategories

Purpose Returns a PDCategories object representing the categories in this database.

Applies to PDRecordAdapter, PDAddressDbHHRecordAdapter, PDDateBookDbHHRecordAdapter2, PDDateBookDbHHRecordAdapter, PDMemoDbHHRecordAdapter, PDTodoDbHHRecordAdapter objects.

Accessibility Read-only.

Prototype `Property **PDCategories** as PDCategories`

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead Or _
   eWrite)
' Get the categories object
Dim Categories as PDCategories
Set Categories = Adapter.PDCategories
' Add a new category
Dim NewIndex As Integer
Do While Category.Name(NewIndex) <> ""
   NewIndex = NewIndex + 1
Loop
Category.Name(NewIndex) = "New Name"
Category.Id(NewIndex) = Category.LastId
Category.LastId = Category.LastId + 1
Category.Save
```

See Also PDCategories object.

PDDatabaseInfo

Purpose Returns a `PDDatabaseInfo` object representing information about this database.

Applies to `PDDatabaseQuery`, `PDRecordAdapter`, `PDResourceAdapter`, `PDAddressDbHHRecordAdapter`, `PDDateBookDbHHRecordAdapter2`, `PDDateBookDbHHRecordAdapter`, `PDMemoDbHHRecordAdapter`, `PDTodoDbHHRecordAdapter` objects.

Accessibility Read-only.

Prototype `Property PDDatabaseInfo as PDDatabaseInfo`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
```

See Also `PDDatabaseInfo` object.

PDHotsyncInfo

Purpose Returns a `PDHotsyncInfo` object representing information about the current HotSync session.

Applies to `PDSystemAdapter` object.

Accessibility Read-only.

Prototype `Property PDHotsyncInfo as PDHotsyncInfo`

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
' Get the HotsyncInfo object
Set HSInfo = pSystem.PDHotsyncInfo
```

See Also `PDHotsyncInfo` object.

PDMemoryCardInfo

Purpose Returns a `PDMemoryCardInfo` object representing information about the handheld's primary storage (called a "memory card").

Applies to `PDSystemAdapter` object.

Accessibility Read-only.

Prototype `Property` **`PDMemoryCardInfo`**`([`*`nCard`*` as Long = 0]) as PDMemoryCardInfo`

Parameters ← *`nCard`*
The memory card number.

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
' Get the MemoryCardInfo object
Set MemCard = pSystem.PDMemoryCardInfo
```

See Also `PDMemoryCardInfo` object.

PDUserInfo

Purpose Returns a `PDUserInfo` object representing information about the current handheld user.

Applies to `PDSystemAdapter` object.

Accessibility Read-only.

Prototype `Property` **`PDUserInfo`**` as PDUserInfo`

Example

```
Dim pSystem as New PDSystemAdapter
Dim UserInfo as PDUserInfo
' Get the UserInfo object
Set UserInfo = pSystem.PDUserInfo
```

See Also `PDUserInfo` object.

Phone1

Purpose	Content of the Phone 1 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`Phone1`** `As String`

Phone2

Purpose	Content of the Phone 2 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`Phone2`** `As String`

Phone3

Purpose	Content of the Phone 3 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`Phone3`** `As String`

Phone4

Purpose	Content of the Phone 4 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`Phone4`** `As String`

Phone5

Purpose	Content of the Phone 5 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property Phone5 As String`

PhoneLabel1

Purpose	Name of the Phone 1 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property PhoneLabel1 As EPDPhoneLabels`
Comments	This property can be set to any of the `EPDPhoneLabels` values.
See Also	`Phone1` property.

PhoneLabel2

Purpose	Name of the Phone 2 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property PhoneLabel2 As EPDPhoneLabels`
Comments	This property can be set to any of the `EPDPhoneLabels` values.
See Also	`Phone2` property.

PhoneLabel3

Purpose	Name of the Phone 3 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`PhoneLabel3`** `As EPDPhoneLabels`
Comments	This property can be set to any of the `EPDPhoneLabels` values.
See Also	`Phone3` property.

PhoneLabel4

Purpose	Name of the Phone 4 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`PhoneLabel4`** `As EPDPhoneLabels`
Comments	This property can be set to any of the `EPDPhoneLabels` values.
See Also	`Phone4` property.

PhoneLabel5

Purpose	Name of the Phone 5 field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`PhoneLabel5`** `As EPDPhoneLabels`
Comments	This property can be set to any of the `EPDPhoneLabels` values.
See Also	`Phone5` property.

Priority

Purpose For PDConduitInfo objects, execution priority for this conduit. For PDTodoDbHHRecord objects, the priority of this To Do List item.

Applies to PDConduitInfo, PDTodoDbHHRecord objects.

Accessibility Read/write.

Prototype `Property Priority as Long`

Comments For PDConduitInfo objects:

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.

This property is optional for all conduits and all versions of HotSync Manager.

For PDTodoDbHHRecord objects:

This property represents the priority of a To Do List item. It can be set to values from 1 to 5.

ProductId

Purpose Handheld product ID.

Applies to PDSystemAdapter object.

Accessibility Read-only.

Prototype `Property ProductId as String`

Example

```
Dim pSystem as New PDSystemAdapter
Dim ProductId as String
' Read the product ID
ProductId = pSystem.ProductId
```

ProductName

Purpose Name of the expansion card product.

Applies to PDExpansionCardInfo object.

Accessibility Read-only.

Prototype `Property **ProductName** As String`

Comments An example value of this property is “SafeBackup 32 MB.”

RamDbCount

Purpose Number of databases in primary storage RAM on the handheld.

Applies to PDMemoryCardInfo, PDDatabaseQuery objects.

Accessibility Read-only.

Prototype `Property **RamDbCount** as Long`

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the RAM database count
Dim DbCount as Long
DbCount = MemCard.RamDbCount
```

RamSize

Purpose Total amount of RAM on the memory card in bytes.

Applies to PDMemoryCardInfo object.

Accessibility Read-only.

Prototype `Property **RamSize** as Long`

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the RAM size
Dim Size as Long
Size = MemCard.RamSize
```

RecordCount

Purpose Number of records in this database.

Applies to DmDatabaseInfo, DmRecordAdapter, PDDatabaseInfo, PDRecordAdapter, PDResourceAdapter, PDAddressDbHHRecordAdapter, PDDateBookDbHHRecordAdapter2, PDDateBookDbHHRecordAdapter, PDMemoDbHHRecordAdapter, PDTodoDbHHRecordAdapter objects.

Accessibility Read-only.

Prototype `Property RecordCount as Long`

Example

```
Dim DbQuery As New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB", eRead Or _
   eWrite)
' How many records in the database??
Dim recordCount as Long
recordCount = Adapter.RecordCount
```

RegistryKey

Purpose The full Windows registry path of the current conduit. Do not use this property; use the PDConduitInfo object to access conduit configuration entries instead.

Applies to PDHotsyncInfo object.

Accessibility Read-only.

Prototype `Property RegistryKey as Long`

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the base registry key
Dim RegKey as Long
RegKey = HSInfo.RegistryKey
```

RegistryPath

Purpose The full Windows registry path of the current conduit. Do not use this property; use the `PDConduitInfo` object to access conduit configuration entries instead.

Applies to `PDHotsyncInfo` object.

Accessibility Read-only.

Prototype `Property` **`RegistryPath`** `as String`

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the full registry path
Dim RegPath as String
RegPath = HSInfo.RegistryPath
```

RemoteNameCount

Purpose The number of entries in the conduit's database `NameList` property.

Applies to `PDHotsyncInfo` object.

Accessibility Read-only.

Prototype `Property` **`RemoteNameCount`** `as Integer`

Comments On entry to a conduit, specifies the number of databases in the DataBaseNameList property.

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the remote name count
Dim nameCount As Integer
nameCount = HSInfo.RemoteNameCount
```

RepeatDay

Purpose	Day on which to repeat this event each month in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`RepeatDay`** `As EPDDayIndex`
Comments	This property can be set to any of the `EPDDayIndex` values and is valid only when the `RepeatType` property is set to `EPDMonthlyByDay`. If the `IsEventRepeatable` property is false, then the value of this property is not valid.
See Also	`IsEventRepeatable` property.

RepeatEndDate

Purpose	Date on which to end this repeating event in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`RepeatEndDate`** `As Date`
Comments	The `PDDateBookDbHHRecord` object cannot handle event end dates earlier than 12/31/1969 4:00:00 PM, which is the earliest date supported by the Date Book application. If the `IsEventRepeatable` property is false, then the value of this property is not valid.
See Also	`IsEventRepeatable` property.

RepeatFrequency

Purpose	How many cycles between instances of this repeating event in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`RepeatFrequency`** `As Integer`
Comments	For example, if this event repeats by day and this property is set to 1, this event repeats every day; if set to 2, it repeats every other day; and so on. If the `IsEventRepeatable` property is false, then the value of this property is not valid.
See Also	`IsEventRepeatable` property.

RepeatType

Purpose	Cycle on which this event repeats in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`RepeatType`** `As EPDRepeatType`
Comments	This property is set to any of the values defined by the `EPDRepeatType` enum. If the `IsEventRepeatable` property is false, then the value of this property is not valid.
See Also	`IsEventRepeatable` property.

RomDbCount

Purpose Number of databases in ROM on the handheld.

Applies to `PDMemoryCardInfo`, `PDDatabaseQuery` objects.

Accessibility Read-only.

Prototype `Property RomDbCount as Long`

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the ROM database count
Dim DbCount as Long
DbCount = MemCard.RomDbCount
```

RomSize

Purpose Total amount of ROM on the memory card in bytes.

Applies to `PDMemoryCardInfo` object.

Accessibility Read-only.

Prototype `Property RomSize as Long`

Example

```
Dim pSystem as New PDSystemAdapter
Dim MemCard as PDMemoryCardInfo
Set MemCard = pSystem.PDMemoryCardInfo
' Get the ROM size
Dim Size as Long
Size = MemCard.RomSize
```

RomSoftwareVersion

Purpose	Palm OS® software version on the handheld.
Applies to	`PDSystemAdapter` object.
Accessibility	Read-only.
Prototype	`Property RomSoftwareVersion as Long`

Example

```
Dim pSystem as New PDSystemAdapter
Dim ROMSoftwareVersion as Long
' Read the ROM software Version
RomSoftwareVersion = pSystem.RomSoftwareVersion
```

RowCount

Purpose	Number of rows in this schema database.
Applies to	`PSDDatabaseInfo` object.
Accessibility	Read-only.
Prototype	`Property RowCount As Long`
Parameters	None.
Comments	This count includes all rows in all tables in this schema database.

RowID

Purpose	The unique row ID of this row.
Applies to	`PSDRowData` object.
Accessibility	Read-only.
Prototype	`Property RowID As Variant`
Parameters	None.

Size

Purpose Size of a file on an expansion card or what to resize a file to.

Applies to `PDVFSFileManager` object.

Accessibility Read/write.

Prototype `Property **Size** As Long`

Comments This property is valid only for files, not directories.

SlotLibRefNumber

Purpose Reference number for the slot driver shared library on the handheld that is allocated to the slot number on which this volume is mounted.

Applies to `PDVFSVolumeManager` object.

Accessibility Read-only.

Prototype `Property **SlotLibRefNumber** As Integer`

Comments This property is valid only when the `mountClass` property is `vfsMountClass_SlotDriver`.

SlotReferenceNumber

Purpose Reference number for the expansion slot that holds this volume.

Applies to `PDVFSVolumeManager` object.

Accessibility Read-only.

Prototype `Property **SlotReferenceNumber** As Integer`

Comments This property is valid only when the `mountClass` property is set to `vfsMountClass_SlotDriver`.

SortInfoSize

Purpose Size of database sort info block in bytes.

Applies to `DmDatabaseInfo`, `PDDatabaseInfo` object.

Accessibility Read-only.

Prototype `Property SortInfoSize as Long`

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim dbInfo as PDDatabaseInfo
Set dbInfo = Adapter.PDDatabaseInfo
' Get the SortInfo block size
Dim SortInfoSize as Long
sortInfoSize = DbInfo.SortInfoSize
```

StartTime

Purpose Time and date on which an event starts in Date Book.

Applies to `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.

Accessibility Read/write.

Prototype `Property StartTime As Date`

Comments The `PDDateBookDbHHRecord` object cannot handle event start times earlier than 12/31/1969 4:00:00 PM, which is the earliest date supported by the Date Book application.

State

Purpose Content of the “State” field in an Address Book record.

Applies to `PDAddressDbHHRecord` object.

Accessibility Read/write.

Prototype `Property **State** As String`

SyncType

Purpose Synchronization type, which is one of the `ESyncTypes` constants.

Applies to `PDHotsyncInfo` object.

Prototype `Property **SyncType** as ESyncTypes`

Comments This property tells a conduit the type of synchronization operation to perform based on the user’s saved preference (synchronize, handheld overwrites desktop, desktop overwrites handheld, do nothing), and if “synchronize” then whether the conduit should perform a fast or slow sync.

IMPORTANT: The `eFast` or `eSlow` values are based only on whether the 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. Instead, call `GetSyncTypeInfo()` to take full advantage of the extra change tracking information that is available only in schema databases and not in classic and extended databases.

Conduits synchronizing schema databases must still rely on this field for the other values (`eHHtoPC`, `ePCtoHH`, etc.) it receives, which are equally valid for all database types.

Example

```
Dim pSystem as New PDSystemAdapter
Dim HSInfo as PDHotsyncInfo
Set HSInfo = pSystem.PDHotsyncInfo
' Get the sync type
Dim SyncType As ESyncTypes
SyncType = HSInfo.SyncType
```

TableCount

Purpose	Number of tables in this schema database.
Applies to	`PSDDatabaseInfo` object.
Accessibility	Read-only.
Prototype	`Property TableCount As Long`
Parameters	None.

TableName

Purpose	The name of the table that this row is in.
Applies to	`PSDRowData` object.
Accessibility	Read-only.
Prototype	`Property TableName As String`
Parameters	None.

Title

Purpose	Content of the “Title” field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property Title As String`

TotalBytes

Purpose Total number of bytes of storage used by this database, including overhead.

Applies to PSDDatabaseInfo, DmDatabaseInfo, PDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property TotalBytes as long`

Comments Contrast this property with DataBytes.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the database size
Dim TotalBytes as Long
TotalBytes = DbInfo.TotalBytes
```

TotalCapacity

Purpose Total capacity, in bytes, of this volume on an expansion card.

Applies to PDVFSVolumeManager object.

Accessibility Read-only.

Prototype `Property TotalCapacity As Long`

Comments This is the maximum formatted space available for the VFS Manager to use on this volume.

Type

Purpose	Database type of this schema database.
Applies to	`PSDDatabaseInfo` object.
Accessibility	Read-only.
Prototype	`Property` **`Type`** `As Long`
Parameters	None.
Comments	The value of this property is a four-byte Palm OS **database type**. Use `DWORDToBSTR()` to convert this value to the usual four-character representation of a database type.

UniqueId

Purpose For a `PDInstallConduitInfo` object, a unique ID associated with this install conduit. For a `PD<PIM>DbHHRecord` object, the record ID of this record.

Applies to `PDInstallConduitInfo`, `PDAddressDbHHRecord`, `PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord`, `PDMemoDbHHRecord`, `PDTodoDbHHRecord` objects.

Accessibility Read/write.

Prototype `PDInstallConduitInfo`:

```
Property UniqueId as Long
```

`PD<PIM>DbHHRecord`:

```
Property UniqueId As Variant
```

Comments For `PDInstallConduitInfo` objects:

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 your installer to ensure that this `UniqueId` 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. This property is *required* for all install conduits.

For `PD<PIM>DbHHRecord` objects:

This value is the record ID assigned to this record.

UsedSpace

Purpose Amount of space, in bytes, already in use on this volume on an expansion card.

Applies to `PDVFSVolumeManager` object.

Accessibility Read-only.

Prototype

```
Property UsedSpace As Long
```

UserId

Purpose User ID, which specifies the user to reference in the users data file.

Applies to PDUserInfo object.

Accessibility Read-only.

Prototype

```
Property UserId as Long
```

Example

```
Dim pSystem as PDSystemAdapter
Dim UserInfo as PDUserInfo
' Get the user info object
Set UserInfo = pSystem.PDUserInfo
' Get the User ID
Dim Id as Long
Id = UserInfo.UserId
```

UserName

Purpose Name of the handheld user in the user data store to synchronize with.

Applies to PDHotsyncInfo, PDUserInfo objects.

Accessibility Read-only.

Prototype

```
Property UserName as String
```

Example

```
Dim pSystem as PDSystemAdapter
Dim UserInfo as PDUserInfo
' Get the user info object
Set UserInfo = pSystem.PDUserInfo
' Get the user name
Dim UserName as String
UserName = UserInfo.UserName
```

Value

Purpose The value of a column in this row that is specified by column name.

Applies to `PSDRowData` object.

Accessibility Read/write.

Prototype

```
Property Value(ByVal ColumnName As String) As
    Variant
```

Parameters → *`ColumnName`*
The name of a column in this row.

Comments This property contains the column value of a column in this row specified by column name. If no data exists for this column, then this propery has a default value, but it does not generate an error. The default value depends on the type you read it into: if a Variant, then it is the `VT_EMPTY` value; if an integer or long value, then it is zero; if it is a string, then it is an empty string; if it is a boolean, then the default is `False`.

If you want a list of only the columns that contain data in this row, then call `GetColumnsWithData()`.

ValueByID

Purpose The value of a column in this row that is specified by column ID.

Applies to `PSDRowData` object.

Accessibility Read/write.

Prototype

```
Property ValueByID(ByVal ColumnID As Long) As
    Variant
```

Parameters → *`ColumnID`*
The column ID of a column in this row.

Comments This property contains the column value of a column in this row specified by column ID. If no data exists for this column, this propery has a default value, which depends on the column's data type, but it does not generate an error. To determine whether a given column contains data, call `IsDataPresent()` or call `GetColumnsWithData()` to get a list of all the columns that contain data in this row.

Version

Purpose An application-specific version number of this database.

Applies to PSDDatabaseInfo, DmDatabaseInfo, PDDatabaseInfo object.

Accessibility Read-only.

Prototype `Property **Version** as Long`

Comments The developer defines this version number for the database, which Palm OS can use to determine whether a newer version of a database can overwrite an older one.

Example

```
Dim DbQuery as New PDDatabaseQuery
Dim Adapter As PDRecordAdapter
' Open the Memo Pad database
Set Adapter = DbQuery.OpenRecordDatabase("MemoDB")
' Get the database information object
Dim DbInfo as PDDatabaseInfo
Set DbInfo = Adapter.PDDatabaseInfo
' Get the database version
Dim Version as Long
Version = DbInfo.Version
```

ViewerId

Purpose ID of the handheld. Not currently used.

Applies to PDUserInfo object.

Accessibility Read-only.

Prototype `Property **ViewerId** as Long`

Example

```
Dim pSystem as PDSystemAdapter
Dim UserInfo as PDUserInfo
' Get the user info object
Set UserInfo = pSystem.PDUserInfo
' Get the Viewer Id
Dim ViewerId as Long
ViewerId = UserInfo.ViewerId
```

WeekIndexForMonthlyRepeatByDay

Purpose	Week on which to repeat this event if it repeats monthly by day in Date Book.
Applies to	`PDDateBookDbHHRecord2`, `PDDateBookDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`WeekIndexForMonthlyRepeatByDay`** `As EPDWeekIndex`
Comments	This property is set to any of the values defined by the `EPDWeekIndex` enum. It is valid only if the `RepeatType` property is set to `EPDMonthlyByDay` and the `IsEventRepeatable` property is set to `True`.

WritableExceptionInReadOnlyRows

Purpose	Flag that indicates whether the data in a column is writable.
Applies to	`PSDColumnInfo` object.
Accessibility	Read/write.
Prototype	`Property` **`WritableExceptionInReadOnlyRows`** `As Boolean`
Parameters	None.
Comments	If `True`, the column data is writable; if `False`, it is not writable.

ZipCode

Purpose	Content of the “Zip Code” field in an Address Book record.
Applies to	`PDAddressDbHHRecord` object.
Accessibility	Read/write.
Prototype	`Property` **`ZipCode`** `As String`

6 Constants

This chapter describes the COM Sync constants. The COM Sync module presents many of these constants as enum values, but for some it only passes the constants defined in the underlying C APIs' header file. The enums are presented first, in alphabetical order, followed by other groups of related constants.

Database Information Flags

Purpose Indicate whether a schema database is excluded from HotSync operations and whether it is in RAM on the handheld

Applies to `PSDDatabaseInfo` object.
`Flags` property.

Constants `eMiscDbFlagExcludeFromSync = 0x0080`
Indicates that the database is to be excluded from the synchronization operations. This is typically the result of the user disabling synchronization for the application associated with the database on the handheld (accessible from the HotSync client's **Options** > **Conduit Setup** menu item). This feature is supported in Palm OS versions 2.0 or later.

`eMiscDbFlagRamBased = 0x0040`
Indicates that the database is located in RAM. If this flag is not set, the database is stored in ROM. This flag is available with Palm OS versions 3.0 or later.

EAccessModes

Purpose Defines the access modes in which you can create or open a database.

Applies to `CreateRecordDatabase()`, `CreateResourceDatabase()`, `OpenRecordDatabase()`, `OpenResourceDatabase()` methods.
`AccessMode` property.

Constants `Constant eExclusive = 32 (&H20)`
Not used.

`Constant eRead = 128 (&H80)`
Read permission.

`Constant eShowSecret = 16 (&H10)`
Open the database with full access to the user's secret records.

`Constant eWrite = 64 (&H40)`
Write permission.

EConnectionType

Purpose Defines two HotSync connection types.

Applies to `ConnectionType` property.

Constants `Constant eCable = 0`
Handheld is connected to a cable.

`Constant eModem = 1`
Handheld is connected to a modem.

EDbFlags

Purpose Defines the attributes of a non-schema database.

Applies to `CreateRecordDatabase()`, `CreateResourceDatabase()`, `OpenRecordDatabase()`, `OpenResourceDatabase()` methods. `DbFlags` property.

Constants `Constant eAppInfoDirty = 4`
The application info block has been modified.

`Constant eBackupDb = 8`
The database should be backed up to the desktop computer if no application-specific conduit is available.

`Constant eBundle = 2048 (&H800)`
The database is bundled with its application during a beam. That is, if the user chooses to beam the application from the Launcher, the Launcher beams this database along with the application's resource database and overlay database.

This attribute applies to Palm OS® versions 4.0 and later. Note that overlay databases are automatically beamed with the application database. You do not need to set this bit in overlay databases.

`Constant eCopyPrevention = 64 (&H40)`
Prevents the database from being copied by methods such as IR beaming.

`Constant eHidden = 256 (&H100)`
This database should be hidden from view. For example, this attribute is set to hide some applications in the Launcher's main view. You can set it on non-resource databases to have

the Launcher disregard the database's records when when it shows a count of records in its **Info** dialog.

`Constant eLaunchableData = 512 (&H200)`
This database (not applicable for executables) can be "launched" from the Launcher, which passes the database's name to its owner application (`'appl'` database with same creator ID) using the `sysAppLaunchCmdOpenNamedDB` action code.

`Constant eOkToInstallNewer = 16 (&H10)`
A backup conduit can install a newer version of this database with a different name if the current database is open.

`Constant eOpenDb = 32768 (&H8000)`
The database is open. Only Palm OS can set this attribute.

`Constant eReadOnly = 2`
The database is a read-only database.

`Constant eRecord = 0`
The database is a record database.

`Constant eRecyclable = 1024 (&H400)`
The database is recyclable. Recyclable databases are deleted when they are closed or upon a system reset.

`Constant eResetAfterInstall = 32 (&H20)`
Reset the handheld after installation.

`Constant eResource = 1`
The database is a resource database. Only Palm OS can set this attribute.

`Constant eStream = 128 (&H80)`
The database is a file stream.

EFirstSync

Purpose Identifies whether the handheld or desktop is performing its first synchronization.

Applies to `FirstSync` property.

Constants `Constant eHH = 2`
First synchronization for the handheld.

`Constant eNeither = 0`
Not first synchronization for either the handheld or the desktop.

`Constant ePC = 1`
First synchronization for the desktop.

EGetConduitInfo

Purpose Indicates the type of information that HotSync Manager is requesting when it calls a conduit's `GetConduitInfo()` entry point.

Applies to `GetConduitInfo()` method.

Constants `Constant eGetConduitInfoDoNotUse = -1 (&Hffffffff)`
Reserved. Do not use.

`Constant eGetConduitName = 0`
HotSync Manager requests the display name of your conduit. HotSync Manager uses the name a conduit passes back only when the conduit is not registered with a name—that is, if the `Name` conduit configuration entry is not set.

`Constant eGetConduitVersion = 3`
HotSync Manager requests the version number of your conduit. Your implementation must pack your major version number into the high byte of the low word in the result, and must pack your minor version number into the low byte of the low word in the result.

`Constant eGetDefaultAction = 2`
HotSync Manager requests the type of default action performed by your conduit. A conduit must pass back one of the `ESyncTypes` enum values.

`Constant eGetMfcVersion = 1`

HotSync Manager requests whether your conduit uses MFC or not, and if so, what version. This version number is actually the version of Visual C++ that MFC shipped with, not necessarily the version number of MFC itself.

Note that this enum value is deprecated in Sync Manager versions 2.4 and later. The corresponding versions of HotSync Manager do not query conduits for an MFC version. A conduit must return one of the `EMfcVersion` enum values only if version 2.3 or earlier of Sync Manager is present.

`Constant ePDDoNotDisplayInConduitListForUser = 4`

HotSync Manager requests whether your conduit should be displayed in the HotSync Manager **Custom** dialog box. If your conduit passes back a zero value or no value, then your conduit's name appears in the **Custom** dialog box. If your conduit passes back any nonzero value, then its name does not appear.

Note that HotSync Manager can pass in this enum value only if Sync Manager version 2.4 or later is present.

`Constant ePDDoNotDisplayProgress = 6`

HotSync Manager requests whether it should display your conduit's name in the **HotSync Progress** dialog box during a HotSync operation. If your conduit passes back a zero value or no value, then HotSync Manager displays your conduit's name. If your conduit passes back any nonzero value, then it does not display your conduit's name.

Note that HotSync Manager can pass in this enum value only if Sync Manager version 2.4 or later is present.

`Constant ePDRunAlways = 5`

HotSync Manager, versions 6.0 and later, requests whether it should run your conduit regardless of whether an application with the same creator ID is on the handheld. If your conduit passes back a zero value or no value, then HotSync Manager runs your conduit only if an application with the same creator ID is on the handheld. If your conduit passes back any nonzero value, then it runs your conduit always.

Note that HotSync Manager can pass in this enum value only if Sync Manager version 2.4 or later is present.

Comments HotSync Manager can pass one of these values to your conduit via its implementation of the `IPDClientNotify` interface's `GetConduitInfo()` method.

ELogActivity

Purpose Defines the desktop HotSync log message types.

Applies to `AddLogEntry()` method.

Constants

`Constant eArchiveFailed = 16 (&H10)`
The archive operation failed.

`Constant eCategoryDeleted = 4`
A category was deleted.

`Constant eChangeCatFailed = 7`
Changing a category failed.

`Constant eCustomLabel = 6`
A custom label was changed.

`Constant eDateChanged = 5`
The date was changed.

`Constant eDoubleModify = 0`
A record has been modified on both the desktop computer and handheld.

`Constant eDoubleModifyArchive = 1`
A record that has been modified on both the desktop and handheld has been archived.

`Constant eDoubleModifySubsc = 23 (&H17)`
A file link record was modified on the desktop. This value is deprecated because the file link feature has been removed in HotSync Manager 6.0.1 and later.

`Constant eError = 30 (&H1e)`
An error occurred. You can use this value only with HotSync Manager version 6.0.1 or later.

`Constant eFileLinkCompleted = 24 (&H18)`
Processing of a file link completed. This value is deprecated because the file link feature has been removed in HotSync Manager 6.0.1 and later.

`Constant eFileLinkDeleted = 25 (&H19)`
A file link was deleted. This value is deprecated because the file link feature has been removed in HotSync Manager 6.0.1 and later.

`Constant eHTMLText = 32 (&H20)`
: This type of log entry contains HTML tags or characters. Using this value passes the string to the log unchanged so that its HTML formatting will be rendered. All other `Activity` values cause the HotSync Log API to replace HTML control characters (<, >, &) with their HTML equivalents (&lt;, &gt;, &amp;). You can use this value only with HotSync Manager version 6.0.1 or later.

`Constant eLocalAddFailed = 13`
: Adding a record on the desktop computer failed.

`Constant eLocalSaveFailed = 17 (&H11)`
: Saving data on the desktop computer failed.

`Constant eRecCountMismatch = 14`
: Record counts did not match.

`Constant eRecommendation = 31 (&H1f)`
: Recommendation that the user do something—for example, resolve conflicts that a conduit could not. You can use this value only with HotSync Manager version 6.0.1 or later.

`Constant eRemoteAddFailed = 9`
: Adding a record on the handheld failed.

`Constant eRemoteChangeFailed = 11`
: Changing a record on the handheld failed.

`Constant eRemoteDeleteFailed = 12`
: Deleting a record on the handheld failed.

`Constant eRemotePurgeFailed = 10`
: Purging a record on the handheld failed.

`Constant eRemoteReadFailed = 8`
: Reading a record failed on the handheld.

`Constant eResetFlagsFailed = 18 (&H12)`
: Resetting of the synchronization flags failed.

`Constant eReverseDelete = 2`
: A record that was deleted on one side has been restored, because the same record was modified on the other side.

`Constant eSyncAborted = 21 (&H15)`
: The synchronization operation was aborted.

`Constant eSyncDidNothing = 26 (&H1a)`
: The user specified that the conduit should not perform any operations during synchronization. You can use this value only with HotSync Manager version 6.0 or later.

`Constant eSyncFinished = 20 (&H14)`
: A conduit completed its synchronization operations successfully.

`Constant eSyncSessionCancelled = 29 (&H1d)`
: The user clicked the **Cancel** button on the **HotSync Progress** dialog box. Only HotSync Manager uses this value; conduits must not. You can use this value only with HotSync Manager version 6.0.1 or later.

`Constant eSyncSessionEnd = 28 (&H1c)`
: The HotSync operation completed. Only HotSync Manager uses this value; conduits must not. You can use this value only with HotSync Manager version 6.0.1 or later.

`Constant eSyncSessionStart = 27 (&H1b)`
: The HotSync operation started. Only HotSync Manager uses this value; conduits must not. You can use this value only with HotSync Manager version 6.0.1 or later.

`Constant eSyncStarted = 19 (&H13)`
: A conduit started its synchronization operations.

`Constant eText = -1 (&Hffffffff)`
: Allows a conduit to add text to the log without incrementing the warning counter.

`Constant eTooManyCategories = 3`
: No more categories can be added.

`Constant eWarning = 22 (&H16)`
: This constant lets a conduit record a warning that doesn't fit any of the other activity codes provided.

`Constant eXMapFailed = 15`
: The position cross-map operation failed.

Comments Your conduit passes one of these values to `AddLogEntry()` to indicate what type of entry it is adding to the log.

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

See Also `AddLogEntry()` method.

EMfcVersion

Purpose Indicates whether your conduit uses MFC or not, and if so, what version.

Applies to `GetConduitInfo()` method.

Constants `Constant ePDMFC_NOT_USED = 268435456 (&H10000000)`
This conduit does not use MFC.

`Constant ePDMFC_VERSION_41 = 1040 (&H410)`
Version 4.1.

`Constant ePDMFC_VERSION_50 = 1280 (&H500)`
Version 5.0.

`Constant ePDMFC_VERSION_60 = 1536 (&H600)`
Version 6.0.

`Constant ePDMFC_VERSION_70 = 1792 (&H700)`
Version 7.0.

Comments Your conduit's implementation of the `IPDClientNotify` interface's `GetConduitInfo()` method must return one of these values when HotSync Manager passes in the *`infoType`* parameter a `EGetConduitInfo` enum value of `eGetMfcVersion`. Note that the version numbers actually correspond to the version of Visual C++ that MFC shipped with.

EPDAlarmAdvTimeUnits

Purpose Defines the time units that the `AlarmAdvanceTime` property is specified in.

Applies to `AlarmAdvanceUnits` property.

Constants `Constant PD_AAU_DAYS = 2`
Days.

`Constant PD_AAU_HOURS = 1`
Hours.

`Constant PD_AAU_MINUTES = 0`
Minutes.

EPDDayIndex

Purpose Defines the days of the week on which a repeating event can occur in Date Book.

Applies to `RepeatDay` property.

Constants

`Constant EPDFriday = 5`
Friday.

`Constant EPDMonday = 1`
Monday.

`Constant EPDSaturday = 6`
Saturday.

`Constant EPDSunday = 0`
Sunday.

`Constant EPDThursday = 4`
Thursday.

`Constant EPDTuesday = 2`
Tuesday.

`Constant EPDWednesday = 3`
Wednesday.

Comments The `PDDateBookDbHHRecord` object's `RepeatDay` property can be set to any of these values.

EPDDisplayPhone

Purpose Defines which contact information to display in the Address Book list view.

Applies to `PDAddressDbHHRecord` object.
`DisplayPhone` property.

Constants `Constant EPDPhoneLabel1 = 0`
Display the information specified by the `PhoneLabel1` property.

`Constant EPDPhoneLabel2 = 1`
Display the information specified by the `PhoneLabel2` property.

`Constant EPDPhoneLabel3 = 2`
Display the information specified by the `PhoneLabel3` property.

`Constant EPDPhoneLabel4 = 3`
Display the information specified by the `PhoneLabel4` property.

`Constant EPDPhoneLabel5 = 4`
Display the information specified by the `PhoneLabel5` property.

See Also `EPDPhoneLabels`, `DisplayPhone`

EPDFileOrigin

Purpose Defines the origins of relative offsets passed to the `Seek()` method.

Applies to `Seek()` method.

Constants `Constant eBeginning = 0`
From the beginning (first data byte of file).

`Constant eCurrent = 1`
From the current position.

`Constant eEnd = 2`
From the end of the file (one position beyond last data byte). Only negative offsets are legal from this origin.

EPDHSConnectionStatus

Purpose Defines the status of a HotSync Manager connection type.

Applies to `GetCommStatus()` method.

Constants

`Constant EPDDisbleConnection = 0`
Disabled.

`Constant EPDEnableConnection = 1`
Enabled.

EPDHSConnectionType

Purpose Defines the types of connection that HotSync Manager can make with the handheld.

Applies to `GetCommStatus()`, `SetCommStatus()` methods.

Constants

`Constant EPDHSInfraRedPort = 4`
Infrared.

`Constant EPDHSModem = 1`
Modem.

`Constant EPDHSNetwork = 2`
Network.

`Constant EPDHSSerialPort = 0`
Serial.

`Constant EPDHSUSBPort = 3`
USB.

EPDPathType

Purpose Defines the path of HotSync user directories and the path of the HotSync Manager executable. HotSync Manager and other desktop software use these paths.

Applies to `GetPath()`, `SetPath()` methods.

Constants `Constant EPDPathHome = 0`

The path to the user directories on the desktop computer—for example, `C:\Documents and Settings\<WinUsername>\My Documents\Palm OS Desktop`. This corresponds to the `Core\Path` configuration entry.

`Constant EPDPathHotSync = 1`

The full path and filename of the HotSync Manager executable—for example, `C:\Program Files\PalmSource\Desktop\hotsync.exe`. This corresponds to the `Core\HotSyncPath` configuration entry.

EPDPhoneLabels

Purpose Defines the values that a `PDAddressDbHHRecord` object's `PhoneLabel<n>` properties can take.

Applies to `Phone1`, `Phone2`, `Phone3`, `Phone4`, `Phone5` properties.

Constants

`Constant PHONE_LABEL_EMAIL = 4`
Email address.

`Constant PHONE_LABEL_FAX = 2`
Fax number.

`Constant PHONE_LABEL_HOME = 1`
Home phone number.

`Constant PHONE_LABEL_MAIN = 5`
Main phone number.

`Constant PHONE_LABEL_MOBILE = 7`
Mobile phone number.

`Constant PHONE_LABEL_OTHER = 3`
Other phone number or email address.

`Constant PHONE_LABEL_PAGER = 6`
Pager number.

`Constant PHONE_LABEL_WORK = 0`
Work phone number.

EPDRepeatType

Purpose Defines the values that a `PDDateBookDbHHRecord` object's `RepeatType` property can take. These specify the cycle on which an event repeats in Date Book.

Applies to `RepeatType` property.

Constants

`Constant EPDDaily = 1`
Repeat every x days.

`Constant EPDMonthlyByDate = 4`
Repeat every x months by date of the month.

`Constant EPDMonthlyByDay = 3`
Repeat every x months by day of the week.

`Constant EPDNoRepeat = 0`
Does not repeat.

`Constant EPDWeekly = 2`
Repeat every x weeks by day of the week.

`Constant EPDYearlyByDate = 5`
Repeat every year by date of the month.

EPDRunOptions

Purpose Defines the values that `GetConduitInfo()` should return to indicate whether HotSync Manager should run the conduit only if a matching application exists on the handheld.

Applies to `GetConduitInfo()` method.

Constants

`Constant ePDRunConduitAlways = 2`
HotSync Manager runs the conduit always, regardless of whether an application with the same creator ID is on the handheld.

`Constant ePDRunOnlyWhenAppExists = 3`
HotSync Manager runs the conduit only if an application with the same creator ID is on the handheld.

EPDSlotMediaType

Purpose Defines the media types supported by the Expansion Manager.

Applies to `GetSlotMediaType()` method.

Constants

`Constant EPDMediaTypeAny = 0`
Matches all media types when looking up a default directory.

`Constant EPDMediaTypeCompactFlash = 2`
CompactFlash.

`Constant EPDMediaTypeMemoryStick = 1`
Memory Stick.

`Constant EPDMediaTypeMultiMediaCard = 4`
MultiMediaCard.

`Constant EPDMediaTypePlugAndPlay = 8`
Universal "plug and play" (PnP) connector.

`Constant EPDMediaTypePoserHost = 7`
Host file system emulated by Palm OS® Emulator.

`Constant EPDMediaTypeRAMDisk = 6`
A RAM disk based media.

`Constant EPDMediaTypeSecureDigital = 3`
Secure Digital.

`Constant EPDMediaTypeSmartMedia = 5`
SmartMedia.

Comments Note that the `MediaType` property of the `PDExpansionCardInfo` and `PDVFSVolumeManager` objects does not use this enum; for its values, see "VFS Manager and Expansion Manager Media Type Constants" on page 575.

EPDUserSyncAction

Purpose Defines a user's preferences for the type of synchronization operation to perform for a specified conduit.

Applies to `GetUserPermanentSyncPreferences()`, `GetUserTemporarySyncPreferences()`, `SetUserPermanentSyncPreferences()`, `GetUserTemporarySyncPreferences()` methods.

Constants

`Constant EPDCustom = 4`
Perform any custom actions implemented in the conduit. HotSync Manager passes only this flag to the conduit, which must determine what action to take.

`Constant EPDDoNothing = 3`
Do not exchange data between the handheld and the desktop computer; the conduit does, however, load and can set flags or log messages.

`Constant EPDHHToPC = 2`
Perform a restore from the handheld: overwrite the desktop database with the database on the handheld.

`Constant EPDPCToHH = 1`
Perform a restore from the desktop computer: overwrite the database on the handheld with the database on the desktop computer.

`Constant EPDSynchronize = 0`
Perform a mirror-image synchronization between the desktop computer and the handheld.

EPDVFSFileOpenAttr

Purpose Defines the mode in which a file or directory is opened by the VFS Manager.

Applies to `Open()` method.

Constants

`Constant eVFSModeCreate = 8`
Create the file if it doesn't already exist.

`Constant eVFSModeExclusive = 1`
Open and lock the file or directory. This mode excludes anyone else from using the file or directory until it is closed.

`Constant eVFSModeRead = 2`
Open for read access.

`Constant eVFSModeReadWrite = 7`
Open for read/write access.

`Constant eVFSModeTruncate = 16 (&H10)`
Truncate the file to zero bytes after opening, removing all existing data.

`Constant eVFSModeWrite = 5`
Open for exclusive write access. This mode excludes anyone else from using the file or directory until it is closed.

EPDVFSFileSystemType

Purpose Defines the file system that is present on a volume on an expansion card.

Applies to `FileSystemType` property.

Constants

`Constant PDvfsFilesystemType_AFS = 10`
Unix Andrew file system.

`Constant PDvfsFilesystemType_EXT2 = 7`
Linux file system.

`Constant PDvfsFilesystemType_FAT = 2`
FAT12 and FAT16, which handles only 8.3 filenames.

`Constant PDvfsFilesystemType_FFS = 8`
Unix Berkeley block based file system.

`Constant PDvfsFilesystemType_HFS = 5`
Macintosh standard hierarchical file system.

`Constant PDvfsFilesystemType_HFSPlus = 4`
Macintosh extended hierarchical file system.

`Constant PDvfsFilesystemType_HPFS = 12`
OS/2 High Performance file system.

`Constant PDvfsFilesystemType_MFS = 6`
Macintosh original file system.

`Constant PDvfsFilesystemType_NFS = 9`
Unix Networked file system.

`Constant PDvfsFilesystemType_Novell = 11`
Novell file system.

`Constant PDvfsFilesystemType_NTFS = 3`
Windows NT file system.

`Constant PDvfsFilesystemType_VFAT = 1`
FAT12 and FAT16, extended to handle long filenames.

EPDWeekIndex

Purpose Defines the week of the month on which an event occurs when it is set up to repeat monthly on a particular day of the week (`RepeatType` property is set to `EPDMonthlyByDay`).

Applies to `WeekIndexForMonthlyRepeatByDay` property.

Constants

`Constant EPDFirst = 0`
First week.

`Constant EPDFourth = 3`
Fourth week.

`Constant EPDLast = 4`
Last week.

`Constant EPDSecond = 1`
Second week.

`Constant EPDThird = 2`
Third week.

EPSDCloseOptions

Purpose Indicates optional actions for `CloseDatabase()` to take when it closes a schema database.

Applies to `PSDDatabaseQuery` objects.
`CloseDatabase()` method.

Constants

`Constant ePSDNone = 0`

`Constant ePSDUpdateBackupDate = 128 (&H80)`

`Constant ePSDUpdateBothDates = 256 (&H100)`

`Constant ePSDUpdateModifiedDate = 64 (&H40)`

EPSDColumnDataType

Purpose Defines the permissible data types that columns can be defined as in a schema.

Applies to `PSDColumnInfo`, `PSDRowData` objects.
`GetDataType()` method.
`DataType` property.

Constants

```
Constant PSDdmInt8 = 5

Constant PSDdmBoolean = 11

Constant PSDdmChar = 15

Constant PSDdmDate = 13

Constant PSDdmDateTime = 12

Constant PSDdmDateTimeSecs = 18 (&H12)

Constant PSDdmDouble = 10

Constant PSDdmFloat = 9

Constant PSDdmInt16 = 6

Constant PSDdmInt32 = 7

Constant PSDdmInt64 = 8

Constant PSDdmInt8 = 5

Constant PSDdmString = 16 (&H10)

Constant PSDdmStringVector = 192 (&Hc0)
Constant PSDdmTime = 14

Constant PSDdmUInt16 = 2

Constant PSDdmUInt32 = 3

Constant PSDdmUInt64 = 4

Constant PSDdmUInt8 = 1

Constant PSDdmVector = 128 (&H80)
```

EPSDDatabaseFlags

Purpose Defines the attributes of a schema database.

Applies to `PSDDatabaseInfo` object.
`Attributes` property.

Constants `Constant ePSDBackupDb = 8`

The database should be backed up to the desktop computer if no application-specific conduit is available.

`Constant ePSDBundle = 2048 (&H800)`

The database is bundled with its application during a beam. That is, if the user chooses to beam the application from the Launcher, the Launcher beams this database along with the application's resource database and overlay database.

This attribute applies to Palm OS® versions 4.0 and later. Note that overlay databases are automatically beamed with the application database. You do not need to set this bit in overlay databases.

`Constant ePSDCopyPrevention = 64 (&H40)`

Prevents the database from being copied by methods such as IR beaming.

`Constant ePSDDbReadOnly = 2`

The database is a read-only database.

`Constant ePSDFixedUp = 16384 (&H4000)`

The Palm OS loader had to fix up an application for relocation. Only Palm OS can set this attribute.

`Constant ePSDHidden = 256 (&H100)`

This database should be hidden from view. For example, this attribute is set to hide some applications in the Launcher's main view. You can set it on non-resource databases to have the Launcher disregard the database's rows when when it shows a count of rows in its **Info** dialog.

`Constant ePSDLaunchableData = 512 (&H200)`

This database (not applicable for executables) can be "launched" from the Launcher, which passes the database's name to its owner application (`'appl'` database with same creator ID) using the `sysAppLaunchCmdOpenNamedDB` action code.

`Constant ePSDOkToInstallNewer = 16 (&H10)`
: A backup conduit can install a newer version of this database with a different name if the current database is open.

`Constant ePSDOpenDb = 32768 (&H8000)`
: The database is open. Only Palm OS can set this attribute.

`Constant ePSDRecord = 0`
: The database contains row data, not resources.

`Constant ePSDRecyclable = 1024 (&H400)`
: The database is recyclable. Recyclable databases are deleted when they are closed or upon a system reset.

`Constant ePSDResetAfterInstall = 32 (&H20)`
: The handheld must be reset after this database is installed. That is, the HotSync application on the handheld forces a reset after installing this database.

`Constant ePSDResource = 1`
: The database is a resource database. Only Palm OS can set this attribute.

`Constant ePSDSchema = 4096 (&H1000)`
: The database is a schema database. Only Palm OS can set this attribute.

`Constant ePSDSecure = 8192 (&H2000)`
: The database is a secure database. Only Palm OS can set this attribute.

`Constant ePSDStream = 128 (&H80)`
: The database is a file stream.

Comments For a description of the different types of databases indicated by some of these attributes, see Chapter 8, “Palm OS Databases,” on page 113 in *Introduction to Conduit Development*.

EPSDDBAttribute

Purpose Defines whether a database is a schema, extended or classic database.

Applies to `PSDDatabaseUtilities` object.

Constants

```
Constant ePSDClassicDBType = 0
```
A **classic database**.

```
Constant ePSDExtendedDBType = 8192 (&H2000)
```
A **extended database**.

```
Constant ePSDSchemaDBType = 4096 (&H1000)
```
A **schema database**.

Comments For a description of the different types of databases indicated by these attributes, see Chapter 8, “Palm OS Databases,” on page 113 in *Introduction to Conduit Development*.

EPSDDesktopTrustStatus

Purpose Indicates the desktop trust status of the HotSync operation that is in progress.

Applies to `PSDDatabaseQuery` objects.
`GetDeskTopTrustStatus()` method.

Constants

```
Constant ePSDDesktopNotTrusted = 1

Constant ePSDDesktopTrusted = 2

Constant ePSDDesktopTrustNotVerified = 3
```

EPSDEncodingType

Purpose Define the character encoding types for the `Encoding` property of a `PSDDatabaseInfo` object.

Applies to `PSDDatabaseInfo` object.
`Encoding` property.

Constants

```
Constant ePSDEncodingUnknown = 0

Constant ePSDEncodingPalmGSM = 78

Constant ePSDEncodingPalmLatin = 3

Constant ePSDEncodingCP1252 = 7

Constant ePSDEncodingISO8859_1 = 2

Constant ePSDEncodingAscii = 1

Constant ePSDEncodingPalmSJIS = 5

Constant ePSDEncodingCP932 = 8

Constant ePSDEncodingShiftJIS = 4

Constant ePSDEncodingUCS2 = 9

Constant ePSDEncodingUTF8 = 6

Constant ePSDEncodingUTF7 = 24

Constant ePSDEncodingUTF16 = 75

Constant ePSDEncodingUTF16BE = 76

Constant ePSDEncodingUTF16LE = 77
```

Latin character encodings

```
Constant ePSDEncodingCP850 = 12

Constant ePSDEncodingCP437 = 13

Constant ePSDEncodingCP865 = 14

Constant ePSDEncodingCP860 = 15

Constant ePSDEncodingCP861 = 16

Constant ePSDEncodingCP863 = 17

Constant ePSDEncodingCP775 = 18

Constant ePSDEncodingMacIslande = 19
Constant ePSDEncodingMacintosh  = 20
Constant ePSDEncodingCP1257 = 21
Constant ePSDEncodingISO8859_3 = 22
Constant ePSDEncodingISO8859_4 = 23
```

Extended Latin character encodings

```
Constant ePSDEncodingISO8859_2 = 26
Constant ePSDEncodingCP1250 = 27
Constant ePSDEncodingCP852 = 28
Constant ePSDEncodingXKamenicky = 29
Constant ePSDEncodingMacXCroate = 30
Constant ePSDEncodingMacXLat2 = 31
Constant ePSDEncodingMacXRomania = 32
```

Japanese character encodings

```
Constant ePSDEncodingEucJp = 25
Constant ePSDEncodingISO2022Jp = 10
Constant ePSDEncodingXAutoJp = 11
```

Greek character encodings

```
Constant ePSDEncodingISO8859_7 = 33
Constant ePSDEncodingCP1253 = 34
Constant ePSDEncodingCP869 = 35
Constant ePSDEncodingCP737 = 36
Constant ePSDEncodingMacXGr = 37
```

Cyrillic character encodings

```
Constant ePSDEncodingCP1251 = 38
Constant ePSDEncodingISO8859_5 = 39
Constant ePSDEncodingKoi8R = 40
Constant ePSDEncodingKoi8 = 41
Constant ePSDEncodingCP855 = 42
Constant ePSDEncodingCP866 = 43
Constant ePSDEncodingMacCyr = 44
Constant ePSDEncodingMacUkraine = 45
```

Turkish character encodings

```
Constant ePSDEncodingCP1254 = 46
Constant ePSDEncodingISO8859_9 = 47
Constant ePSDEncodingCP857 = 48
Constant ePSDEncodingMacTurc = 49
Constant ePSDEncodingCP853 = 50
```

Arabic character encodings

```
Constant ePSDEncodingISO8859_6 = 51
Constant ePSDEncodingAsmo708 = 52
Constant ePSDEncodingCP1256 = 53
Constant ePSDEncodingCP864 = 54
Constant ePSDEncodingAsmo708Plus = 55
Constant ePSDEncodingAsmo708Fr  = 56
Constant ePSDEncodingMacAra = 57
```

Simplified Chinese character encodings

```
Constant ePSDEncodingGB2312 = 58
Constant ePSDEncodingHZ = 59
Constant ePSDEncodingGBK = 82
Constant ePSDEncodingPalmGB = 83
```

Traditional Chinese character encodings

```
Constant ePSDEncodingBig5 = 60
Constant ePSDEncodingBig5_HKSCS = 79
Constant ePSDEncodingBig5Plus = 80
Constant ePSDEncodingPalmBig5 = 81
```

Vietnamese character encodings

```
Constant ePSDEncodingViscii = 61
Constant ePSDEncodingViqr = 62
Constant ePSDEncodingVncii = 63
Constant ePSDEncodingVietnet = 65
Constant ePSDEncodingCP1258 = 66
```

Korean character encodings

```
Constant ePSDEncodingKsc5601 = 67
Constant ePSDEncodingCP949 = 68
Constant ePSDEncodingISO2022Kr  = 69
```

Hebrew character encodings

```
Constant ePSDEncodingISO8859_8I  = 70
Constant ePSDEncodingISO8859_8  = 71
Constant ePSDEncodingCP1255 = 72
Constant ePSDEncodingCP1255V = 73
```

Thai character encodings

```
Constant ePSDEncodingTis620 = 74
Constant ePSDEncodingCP874 = 64
```

EPSDMatchMode

Purpose Defines how a specified list of category IDs is to be matched against the category memberships of rows in a schema database.

Applies to `PSDDatabaseAdapter` objects.
`DeleteRowsInCategory()` method.

Constants

```
Constant eDbMatchAll = 2

Constant eDbMatchAny = 1

Constant eDbMatchExact = 3
```

EPSDOpenMode

Purpose Defines the access modes in which a schema database can be opened.

Applies to `PSDDatabaseQuery` objects.
`OpenDatabase()` method.

Constants

```
Constant ePSDReadOnly = 1

Constant ePSDReadWrite = 3

Constant ePSDShowSecret = 16 (&H10)
```

EPSDSearch

Purpose	Specifies how `PSDDatabaseQuery`.`ReadDatabaseNameList()` performs a search operation.
Applies to	`PSDDatabaseQuery` objects. `ReadDatabaseNameList()` method.
Constants	`Constant eDbLatest = 64 (&H40)`
	`Constant eDbNew = 128 (&H80)`
	`Constant eDbNone = 0`

EPSDShareMode

Purpose	Defines the shared access modes in which a schema database can be opened.
Applies to	`PSDDatabaseQuery` objects. `OpenDatabase()` method.
Constants	`Constant EPSDShareNone = 0`
	`Constant EPSDShareRead = 1`
	`Constant EPSDShareReadWrite = 3`

EPSDSyncAtom

Purpose Indicates the type of sync atom for which you call `GetSyncTypeInfo()` to get the synchronization mode.

Applies to `PSDDatabaseAdapter` objects.
`GetSyncTypeInfo()` method.

Constants

`Constant PSDSyncAtomCategory = 2`
Indicates the type of synchronization for categories.

`Constant PSDSyncAtomRow = 3`
Indicates the type of synchronization for records.

`Constant PSDSyncAtomSchema = 1`
Indicates the type of synchronization for schemas.

EPSDSyncType

Purpose Indicates whether a conduit needs to perform a fast, slow, or no synchronization on all sync atoms of a particular type in a schema database.

Applies to `PSDDatabaseAdapter` objects.
`GetSyncTypeInfo()` method.

Constants

`Constant PSDSyncTypeFastSync = 1`
Indicates that the sync atom has changed and that all the change flags are valid.

`Constant PSDSyncTypeNoChange = 0`
Indicates that the sync atom has not changed.

`Constant PSDSyncTypeSlowSync = 2`
Indicates that the change flags (or lack of change flags) for the sync atom cannot be trusted and only an object-by-object comparison can determine whether the sync atom has changed.

ERecordAttributes

Purpose Defines a database record's attribute flags.

Applies to `ReadById()`, `ReadByIndex()`, `ReadNext()`, `ReadNextInCategory()`, `ReadNextModified()`, `ReadNextModifiedInCategory()` methods.

Constants

`Constant eArchive = 8`
Record has been archived.

`Constant eDelete = 128 (&H80)`
Record has been deleted.

`Constant eDirty = 64 (&H40)`
Record has been modified.

`Constant eSecret = 16 (&H10)`
Record is private.

ERemoveSetType

Purpose Defines the multirecord removal flags.

Applies to `RemoveSet()` method.

Constants

`Constant eRemoveAllDeletedRecords = 1`
Remove all the records in the database that are marked deleted.

`Constant eRemoveAllRecords = 0`
Remove all records in the database.

`Constant eRemoveAllRecordsInCategory = 2`
Remove all records in the given category.

ESyncPref

Purpose	Defines the conduit configuration flags.
Applies to	`CfgConduit()` method.

Constants	`Constant eNoPreference = 0`
	Not specified.
	`Constant ePermanentPreference = 1`
	Preferences are permanent.
	`Constant eTemporaryPreference = 2`
	Preferences are for the next HotSync operation only.

ESyncTypes

Purpose	Defines the HotSync synchronization types.
Applies to	`SyncType` property.

Constants	`Constant eBackup = 5`
	Backup handheld database to the desktop.
	`Constant eDoNothing = 6`
	Do not do anything.
	`Constant eFast = 0`
	Perform a fast synchronization.
	`Constant eHHtoPC = 2`
	Copy handheld database to the desktop, overwrite all old records.
	`Constant eInstall = 4`
	Install new application to the handheld.
	`Constant ePCtoHH = 3`
	Copy desktop database to the handheld, overwrite all old records.
	`Constant eProfileInstall = 7`
	Perform a profile download.
	`Constant eSlow = 1`
	Perform a slow synchronization.

EUpdateDbDates

Purpose Defines which database dates to update when a database is closed.

Applies to `CloseOptions` property.

Constants

`Constant eBackupDate = 128 (&H80)`
Update the last-backup date.

`Constant eBothDates = 192 (&Hc0)`
Update both last-backup and last-modified dates.

`Constant eModifiedDate = 64 (&H40)`
Update the last-modified date.

`Constant eNone = 0`
Do not change dates.

Hardware Capability Flags

Purpose Indicates the capabilities of an expansion card.

Applies to `CapabilityFlags` property.

Constants

`#define expCapabilityHasStorage 0x00000001`
Indicates that the card has data storage. The `expCapabilityReadOnly` flag indicates whether the card can be written or only read, though.

`#define expCapabilityReadOnly 0x00000002`
Indicates that the card is read-only.

Comments These constants are defined in the `ExpansionMgr.h` file in the C/C++ Sync Suite.

HotSync Manager Start Options Constants

Purpose Defines options for starting the HotSync Manager application with `RestartHotSyncMgr()`.

Applies to `RestartHotSyncMgr()` method.

Constants `#define HSFLAG_DEVICE_SYNC_CHECK 0x00010000`
Run HotSync Manager in start/stop sync mode. In this mode, during a HotSync operation with the handheld, HotSync Manager runs no conduits; it only validates the connection. This is the same as starting HotSync Manager from the command line with the `-c` option.

`#define HSFLAG_INSPECT_CONDUIT 0x00001000`
Run HotSync Manager and launch the Conduit Inspector utility. This is the same as starting HotSync Manager from the command line with the `-ic` option. For more information on Conduit Inspector, see Chapter 6, "Conduit Inspector Utility," on page 29 in the *Conduit Development Utilities Guide*. This value is defined only for HotSync Manager API versions 2 and later.

`#define HSFLAG_LOG_DEBUG_LEVEL_1 0x00000100`
Run HotSync Manager in debug log mode 1. This is the same as starting HotSync Manager from the command line with the `-L1` option.

`#define HSFLAG_LOG_DEBUG_LEVEL_2 0x00000200`
Run HotSync Manager in debug log mode 2. This is the same as starting HotSync Manager from the command line with the `-L2` option.

`#define HSFLAG_NONE 0x00000000`
Set no flags. This is the same as starting HotSync Manager from the command line with no options.

`#define HSFLAG_RESTORE_REGISTRY 0x00000001`
Restore any missing configuration entries. This is the same as starting HotSync Manager from the command line with the `-r` option.

`#define HSFLAG_RESTORE_REGISTRY_DEFAULT 0x00000002`
Restore configuration entries to defaults. This is the same as starting HotSync Manager from the command line with the `-d` option.

```
#define HSFLAG_VERBOSE 0x00000010
```

Run HotSync Manager in verbose log mode. This is the same as starting HotSync Manager from the command line with the `-v` option.

Comments These constants are defined in the `HSAPI.h` file in the C/C++ Sync Suite. Each of these options corresponds to a HotSync Manager command-line option described in more detail in "Using Command-line Options for HotSync Manager" on page 24 in *Conduit Development Utilities Guide*.

VFS File and Directory Attributes

Purpose Defines the bits that can be used individually or in combination when setting or interpreting the attributes for a given file or directory.

Applies to `Attributes` property.

Constants

`#define vfsFileAttrArchive (0x00000020UL)`
Archived file or directory.

`#define vfsFileAttrDirectory (0x00000010UL)`
A directory, not a file.

`#define vfsFileAttrHidden (0x00000002UL)`
Hidden file or directory.

`#define vfsFileAttrLink (0x00000040UL)`
Link to another file or directory.

`#define vfsFileAttrReadOnly (0x00000001UL)`
Read-only file or directory.

`#define vfsFileAttrSystem (0x00000004UL)`
System file or directory.

`#define vfsFileAttrVolumeLabel (0x00000008UL)`
Volume label.

Comments These constants are defined in the `VFSMgr.h` file in the C/C++ Sync Suite.

VFS Manager and Expansion Manager Media Type Constants

Purpose Defines the media types supported by the Expansion Manager and VFS Manager.

Applies to `MediaType` property.

Constants `#define ExpMediaType_Any 'wild'`
Matches all media types when looking up a default directory.

`#define ExpMediaType_CompactFlash 'cfsh'`
CompactFlash.

`#define ExpMediaType_MemoryStick 'mstk'`
Memory Stick.

`#define ExpMediaType_MultiMediaCard 'mmcd'`
MultiMediaCard.

`#define ExpMediaType_PlugNPlay 'pnps'`
Universal "plug and play" (PnP) connector.

`#define ExpMediaType_PoserHost 'pose'`
Host file system emulated by Palm OS® Emulator.

`#define ExpMediaType_RAMDisk 'ramd'`
A RAM-disk-based media.

`#define ExpMediaType_SecureDigital 'sdig'`
Secure Digital.

`#define ExpMediaType_SmartMedia 'smed'`
SmartMedia.

Comments These constants are defined in the `ExpansionMgr.h` file in the C/C++ Sync Suite. They are the values that the `MediaType` property of the `PDExpansionCardInfo` and `PDVFSVolumeManager` objects can be set to. They correspond exactly to the values of the `EPDSlotMediaType` enum defined in the `PDStandard` library; however, this enum is not defined in the `PDDirect` library, so this enum is not available to the Expansion and VFS Manager objects.

Because the `MediaType` property returns a value of type `Long`, you can use the `PDCondMgr`.`CreatorIDToString()` method to convert it to one of the string values above.

VFS Volume Attributes

Purpose Defines the bits that can be used individually or in combination when interpreting the attributes for a given volume.

Applies to `Attributes` property.

Constants `#define vfsVolumeAttrHidden (0x00000002UL)`

The volume should not be visible to the user. For more information, see "Hidden Volumes" on page 93 in the *COM Sync Suite Companion*.

`#define vfsVolumeAttrReadOnly (0x00000002UL)`

The volume is read only.

`#define vfsVolumeAttrSlotBased (0x00000001UL)`

The volume is associated with a slot driver as opposed to the Palm OS® Emulator.

Comments These constants are defined in the `VFSMgr.h` file in the C/C++ Sync Suite.

VFS Volume Mount Class Constants

Purpose Defines how a given volume is mounted.

Applies to `Format()` method.
`mountClass` property.

Constants

`#define vfsMountClass_Simulator sysFileTSimulator`
: Mount the volume through the 68K Palm Simulator. This is used for testing.

`#define vfsMountClass_SlotDriver sysFileTSlotDriver`
: Mount the volume with a slot driver shared library.

`#define sysFileTSimulator '\?\?\?\?'`
: File type for 68K Palm Simulator files (`app.tres`, `sys.tres`). `vfsMountClass_Simulator` is defined as this value.

`#define sysFileTSlotDriver 'libs'`
: File type for slot driver libraries. `vfsMountClass_SlotDriver` is defined as this value.

Comments These constants are defined in the `VFSMgr.h` file in the C/C++ Sync Suite. They are used in both the *`mountClass`* input parameter in the `Format()` method and in the `mountClass` property of a `PDVFSVolumeManager` object.

Because the `PDVFSVolumeManager`.`mountClass` property returns a value of type `Long`, you can use the `PDCondMgr`.`CreatorIDToString()` method to convert it to one of the string values below.

7 Errors

This chapter describes the COM Sync error codes, implemented as constants defined in the `EPDDirectErrors` enum. The COM Sync Suite implements the standard-COM rich error handling protocol.

Visual Basic implements this error handling protocol and wraps it with the `Err` object. Because errors are returned using the standard Visual Basic B methodology, the Visual Basic developer and the COM-based conduit developer can find detailed usage information in the Microsoft Visual Studio online documentation.

This chapter describes the COM Sync Error Codes sorted alphabetically in Table 7.1.

COM Sync Error Codes

Table 7.1 COM Sync error codes

Error	Description
`eAlreadyExists`	The creator ID that you specified to use as a new creator ID in the configuration entry is already in use. If an install conduit, the unique ID that you specified is already in use.
`eAlreadyInstalled`	The specified conduit or notifier is already installed.
`eBadAdapterName`	Returned if the Adapter name (ProgID) doesn't resolve to a registered CLSID. The following `PDDatabaseQuery` methods can cause this error: `OpenRecordDatabase()`, `OpenResourceDatabase()`, `CreateRecordDatabase()`, and `CreateResourceDatabase()`.
`eBadOperation`	Returned by the Sync Manager API. The requested operation is not supported on the given database type (record or resource).
`eBufferTooSmall`	Returned by the Sync Manager API. The passed buffer is too small for the reply data.
`eCantCreateConduit`	The conduit could not be registered with HotSync® Manager.
`eCantSetValue`	The specified conduit configuration entry could not be set.
`eCommunications`	Communications with the handheld has either not been initialized or has been lost.
`eDatabaseMismatch`	Attempts to open a database of the wrong type, record vs. resource, or use a mismatched database adapter to return this error. The following `PDDatabaseQuery` methods can cause this error: `OpenRecordDatabase()` and `OpenResourceDatabase()`.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`eDuplicateName`	Specified name already exists.
`eFailedToDelete`	Failed to delete the configuration entries for the specified conduit. Or failed to remove the specified install file because, for example, the file does not exist.
`eFileExists`	Attempts to create a database that exists causes this error. Some `PDDatabaseQuery` methods that cause this error are `CreateRecordDatabase()` and `CreateResourceDatabase()`.
`eFileInUse`	Specified file already exists.
`eFileIsOpen`	Attempting to open or remove an open database returns this error. The following `PDDatabaseQuery` methods can cause this error: `OpenRecordDatabase()` and `RemoveDatabase()`.
`eFileNotOpen`	Returned by the Sync Manager API. The attempt to open the database failed.
`eHotsyncLogError`	Returned by `AddLogEntry()` method.
`eHotSyncNotFound`	HotSync Manager is not running.
`eHSAPIFailure`	`PDHotSyncUtility` cannot communicate with HotSync Manager.
`eIDInUse`	The specified user ID is already in use.
`eIndexOutOfRange`	The specified index value is out of range.
`eInvalidConnType`	The specified HotSync Manager connection type is not one defined by `EPDHSConnectionType`.
`eInvalidID`	The specified conduit creator ID is not valid.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`eInvalidInstallID`	The specified unique ID for an install conduit is not valid.
`eInvalidPath`	The specified path is longer than 256 characters. Methods that generate this error check only the length of the path, not whether it is invalid for any other reason (invalid characters, file does not exist, and so on).
`eInvalidType`	The specified HotSync Manager connection type status is not one defined by `EPDHSConnectionStatus`.
`eInvalidUser`	If supplied a user ID, it is `Null` or not for an available user. If supplied a user name, it is `Null` or more than 20 characters long.
`eInvalidUserDir`	The specified user directory is invalid.
`eLocalMemory`	Not enough memory on the desktop to perform the requested operation.
`eMoveFailed`	Failed to move the specified install file because, for example, the file does not exist
`eNoCorePath`	A value for the `Core\Path` configuration entry does not exist. See `GetRootDirectory()`.
`eNoHSPath`	`PDHotSyncUtility` could not find the HotSync Manager path.
`eNoServerObject`	Unused.
`eNoSuchConduit`	The specified conduit does not exist.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`eNotAttached`	If you create a `PSDRowData` object without attching it to any table and try to set some values, this error is returned by all `PSDRowData` methods if they are called before calling `AttachToTable()`. This is not true when a `PSDRowData` object is returned by `ReadRow()`.
`eNotFound`	Returned by the Sync Manager API. Returned by `PDRecordAdapter` or `PDResourceAdapter` `ReadXXX` methods when the Index or UniqueId does not resolve to an existing record. This includes attempts to read past EOF.
`eNotifierNotFound`	The specified notifier is not registered.
`eNoUsers`	The users data file does not contain any user information.
`eObjectCreation`	Returned if the COM Sync module is unable to create the adapter object. The following `PDDatabaseQuery` methods can cause this error: `OpenRecordDatabase()`, `OpenResourceDatabase()`, `CreateRecordDatabase()`, and `CreateResourceDatabase()`.
`eOsVersion`	Unused.
`eOtherError`	An unspecified error occurred.
`eOtherUDErr`	Either the specified directory or filename is bad, the user data store could not be accessed, or another method or program is accessing the user data store (only one process can access the user data store at a time).

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`ePalmLogFull`	Returned by the Sync Manager API. A data limit has been exceeded on the handheld. For example, this happens when the HotSync log size limit has been exceeded on the handheld.
`eParamError`	Returned by all methods and properties to indicate any parameter error detected.
`ePathBig`	The path or string is more than 256 characters long.
`ePSDAccessDenied`	The Authorization Manager on the handheld denied access to the database or the database cannot be unencrypted.
`ePSDBackupBitNotSet`	Backup failed because the database's backup bit was not set.
`ePSDBuiltinProperty`	Cannot modify or remove a built-in column property.
`ePSDCategoryNameNotSpecified`	A category name has not been specified.
`ePSDColumnIDExists`	The column with the specified ID already exists.
`ePSDDeviceNotConnected`	The operation cannot complete without the device connected.
`ePSDDiskFull`	A write to the desktop disk failed because the disk is full.
`ePSDInvalidCategoryID`	The specified category ID is invalid.
`ePSDInvalidColumnID`	The specified column ID is invalid.
`ePSDInvalidColumnName`	The specified column name is invalid—for example, the name contains spaces.
`ePSDInvalidColumnSize`	The specified column data size for a fixed-sized column is invalid.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`ePSDInvalidColumnSpec`	One or more of the column attributes is invalid.
`ePSDInvalidColumnType`	The specified column type is invalid.
`ePSDInvalidHandle`	The specified database handle is 0, which is an invalid value.
`ePSDInvalidID`	The specified ID is invalid.
`ePSDInvalidImage`	The image being installed is not a valid PalmOS database.
`ePSDInvalidMatcMode`	An invalid match mode is specified.
`ePSDInvalidPropID`	The specified property ID is invalid.
`ePSDInvalidTableDefn`	The specified table definition is invalid.
`ePSDInvalidTableName`	The specified table name is invalid—for example, the name contains spaces.
`ePSDMaxCategoryLimit`	Cannot add another category, because the maximum number of categories already exists.
`ePSDNamesAlreadyExists`	The specified column or table name already exists.
`ePSDNoData`	No column data is present.
`ePSDNoSecureDatabaseCreationFromDeskTop`	Cannot create a secure database on the handheld from the desktop.
`ePSDNotRecordDB`	A method that operates only on classic *record* databases attempted to operate on a classic *resource* database.
`ePSDNotSchemaDB`	A method that operates only on *schema* databases attempted to operate on a *nonschema* database.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`ePSDNotSecureDB`	A method that operates only on *secure* databases attempted to operate on a *nonsecure* database.
`ePSDOperationAborted`	The writing of column values has aborted because of one or more invalid inputs.
`ePSDOptionCantBeUsedAlone`	When opening a database, the `eShowSecret` open mode cannot be specified by itself.
`ePSDPathNotFound`	The directory or file path is not valid.
`ePSDRemotePassword`	Backup or restore of security data failed because of an invalid password.
`ePSDRetrievalFailure`	Retrieval of one or more column definitions, column values, or column property values failed.
`ePSDRowsExist`	Cannot remove the specified table because rows that belong to it still exist.
`ePSDStreamError`	An error while streaming the data to the handheld.
`ePSDTableNameNotSpecified`	A table name has not been specified.
`eReadOnlyMode`	Returned by the Sync Manager API. Returned by `PDRecordAdapter` or `PDResourceAdapter` methods when an attempt has been made to write to or change a database opened in read-only mode.
eRecordDeleted	A record specified by ID has been deleted. Can be returned by `ReadRow()`.
`eRegistryFailure`	Unable to access the conduit configuration entries.
`eRemoteMemory`	Returned by the Sync Manager API. There is insufficient memory on the handheld to receive or complete the request.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`eSaveErr`	Saving changes was not successfully completed.
`eSyncApiError`	Unused.
`eSyncApiVersion`	The following `PDSystemAdapter` methods are unavailable for Sync Manager versions prior to version 2.2: `CallRemoteModule()` and `ReadFeature()`.
`eSyncCanceled`	Returned by the Sync Manager API. The HotSync operation was cancelled by the desktop computer user.
`eUDSemaphoreError`	Another method or program is accessing the user data store.
`eUDUnableToCreate`	`PDUserData` could not create a new file.
`eUnableToClose`	`PDHotSyncUtility` cannot close HotSync Manager.
`eUnableToStart`	`PDHotSyncUtility` cannot start the HotSync Manager application.
`eUnknownRequest`	Returned by the Sync Manager API. The handheld is running an unsupported version of the HotSync protocol—for example, when a function that works only with Palm OS Cobalt is called against a handheld running an earlier version of Palm OS.
`eUserExists`	No such user exists or no users exist.
`eValueNotFound`	The specified value could not be found in the configuration entries for this conduit.
`eVFSBadData`	The operation could not be completed because of invalid data—for example, importing a database from a corrupted PRC file.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`eVFSBadName`	Invalid filename, path, or volume label. See "Naming Files" on page 98, "Directory Paths" on page 100, or "Naming Volumes" on page 95 in the *COM Sync Suite Companion*.
`eVFSBufferOverflow`	The supplied buffer is too small.
`eVFSCardNotPresent`	No expansion card is present in the given slot.
`eVFSDirectoryNotFound`	The full path, excluding filename or new directory name, does not exist or no default directory is registered for this file type.
`eVFSDirNotEmpty`	The directory is not empty and therefore cannot be deleted.
`eVFSDiskFileAccess`	Failed to create or open the disk file on the desktop.
`eVFSDiskFull`	Not enough space on the desktop's disk.
`eVFSEnumerationEmpty`	No volumes are present to enumerate or none remain to enumerate.
`eVFSFileAccessOther`	Generic desktop file access error. If returned by `CopyFileToDeskTop()`, it could not access or map the desktop file—for example, because of insufficient memory on the desktop.
`eVFSFileAlreadyExists`	A file or a directory with this name exists in this location already.
`eVFSFileBadRef`	The file reference number is invalid: it has been closed or was not obtained from `Open()`.
`eVFSFileEOF`	Reaching the end of a file is not treated as an error by the COM Sync module, but as a state change for the open file. See the `EOF` property.
`eVFSFileGeneric`	Generic VFS Manager file error.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`eVFSFileNotFound`	The file was not found in the specified path.
`eVFSFilePermissionDenied`	Permission denied to perform requested operation—for example, an attempt to write to a read-only file or to read a file already opened in the `eVFSModeExclusive` mode.
`eVFSFileStillOpen`	The file is still open—for example, trying to delete or rename an open file.
`eVFSInvalidOperation`	A file system is not present or the VFS Manager method is not valid.
`eVFSInvalidSlotNumber`	The slot reference number is not valid.
`eVFSIsADirectory`	This operation can be performed only on a regular file, not a directory.
`eVFSNameShortened`	A volume name or filename was automatically shortened to conform to the file system specification.
`eVFSNoFileSystem`	None of the file systems installed on the handheld support this operation.
`eVFSNoSectorReadWrite`	The expansion card does not support the slot driver block read/write API.
`eVFSNotADirectory`	This operation can be performed only on a directory.
`eVFSNotEnoughPower`	Insufficient battery power on the handheld to perform the operation.
`eVFSNotOpen`	The file system library on the handheld necessary for this call has not been installed or has not been opened.
`eVFSSlotDeallocated`	The slot reference number is within the valid range, but the Expansion Manager has unloaded the slot driver on the handheld.

Table 7.1 COM Sync error codes *(continued)*

Error	Description
`eVFSUnimplemented`	This call is not implemented.
`eVFSUnsupportedOperation`	Either virtual file systems are not present on the handheld or the handheld does not have an expansion slot.
`eVFSVolumeBadRef`	The volume reference number is invalid because, for example, the volume has not been mounted.
`eVFSVolumeFull`	There is insufficient space left on the volume.
`eVFSVolumeStillMounted`	The volume is still mounted.
`S_OK`	Returned by all methods and properties when there are no errors.

A

Revision History

This appendix lists significant additions and changes in each release of the COM Sync Suite, part of the Palm OS® CDK for Windows:

For a summary of the enhancements and new features available in the last few releases of the CDK, see Chapter 1, "What's New in the Palm OS CDK," on page 1 in a *Introduction to Conduit Development*.

Changes in COM Sync Suite 6.0.1

This section lists the COM Sync Suite APIs that are new or changed in version 6.0.1 of the CDK:

- Added the following values to the `ELogActivity` enum to match those available in the C API:
  - `eDoubleModifySubsc`
  - `eFileLinkCompleted`
  - `eFileLinkDeleted`
  - `eSyncDidNothing`
  - `eSyncSessionStart`
  - `eSyncSessionEnd`
  - `eSyncSessionCancelled`
  - `eError`
  - `eRecommendation`
  - `eHTMLText`

- Added the `PSDDatabaseUtilities` object to provide the following new methods:
  - `BackupDatabase()`
  - `CallDeviceApplication()`
  - `InstallAndBackupDatabase()`
  - `IsDatabaseBackupNeeded()`

  The remaining methods provided in this object are duplicated for convenience from the `PSDDatabaseQuery` object.
- Added the `PDSystemCondMgr` object, which provides methods for registering and managing conduits that are registered for the system. These methods are similar to those defined in `PDCondMgr` for conduits that are registered for the current Windows user.
- Added the following values to the `EGetConduitInfo` enum to enable your conduit to opt out of certain default HotSync Manager behaviors:
  - `ePDDoNotDisplayInConduitListForUser`
  - `ePDDoNotDisplayProgress`
  - `ePDRunAlways`

  Only HotSync Manager versions 6.0.1 or later query your conduit's `GetConduitInfo()` entry point for these options.
- Added the `EPDRunOptions` enum to specify a `GetConduitInfo()` return value when the input parameter *`infoType`* `= ePDRunAlways`.
- Added the `EPSDDBAttribute` enum to specify whether a database is a schema, extended, or classic database in certain methods that work with all types of databases.
- The following have been deprecated because the file link feature has been removed from HotSync Manager 6.0.1:
  - `LaunchFileLinkDlg()` method
  - `ELogActivity` enum values `eDoubleModifySubsc`, `eFileLinkCompleted`, and `eFileLinkDeleted`

Changes in COM Sync Suite 6.0

This section lists the COM Sync Suite objects that are new in version 6.0 of the CDK:

Extended Database Objects

The extended database objects enable conduits to access **extended databases** on handhelds running Palm OS® Cobalt. The object model for extended databases mirors that of **classic databases**, except that extended databases do not support resources and must be identified by both name and creator ID. These objects are detailed in this reference:

- "DmCategories" on page 8
- "DmDatabaseInfo" on page 10
- "DmDatabaseQuery" on page 12
- "DmRecordAdapter" on page 13

Schema Database Objects

The schema database objects enable conduits to access **schema databases** on handhelds running Palm OS Cobalt. The object model for schema databases is quite different from that of **classic databases** and **extended databases**. These objects are detailed in this reference:

- "PSDCategoryAdapter" on page 94
- "PSDColumnInfo" on page 95
- "PSDDatabaseAdapter" on page 96
- "PSDDatabaseInfo" on page 98
- "PSDDatabaseQuery" on page 100
- "PSDRowAdapter" on page 104
- "PSDRowData" on page 106
- "PSDRowSet" on page 108
- "PSDTable" on page 109

Miscellaneous Changes

This version of the COM Sync Suite includes the following changes:

- Added the `PDAddressDbHHRecord`.`DisplayPhone` property so that you can specify which contact information (phone number, email address, etc.) to display in the Address Book list view.
- Fixed a problem with `ReadUniqueIdList()` in which it did not pass back a correct value for the number of record IDs returned (*`nRead`*) when the start index (*`nFirstIndex`*) is nonzero. In this version, this method correctly passes back via *`nRead`* the number of record IDs it returns.
- In HotSync Manager versions 6.0 and later (Sync Manager versions 2.4 and later), setting the system time via the `PDSystemAdapter`.`DateTime` property works only if the handheld is running Palm OS Cobalt.
- Removed the `vfsMountClass_POSE` constant from "VFS Volume Mount Class Constants" on page 577, because it cannot be used from the desktop VFS Manager.

Changes in COM Sync Suite 4.03

This section lists the COM Sync Suite objects and other features that are new in version 4.03 of the CDK:

- Installation and Support Objects
- Expansion Manager and VFS Manager Objects
- PIM Database and Record Objects

Installation and Support Objects

These objects enable installers and desktop applications to register conduits and install conduits with HotSync® Manager and retrieve information about them, queue files for installation on the handheld, access information about handheld users on the desktop, and control the HotSync Manager application. These objects are detailed in this reference:

- "PDCondMgr" on page 26
- "PDConduitInfo" on page 28
- "PDInstallConduit" on page 58
- "PDInstallConduitInfo" on page 59
- "PDHotSyncUtility" on page 54
- "PDInstall" on page 56
- "PDUserData" on page 82

See Chapter 4, "Writing an Installer," on page 51 in the *COM Sync Suite Companion* for further discussion on using these objects.

Expansion Manager and VFS Manager Objects

These objects enable conduits and desktop applications to access expansion slots on a handheld and the virtual file systems on cards in these slots. These objects are detailed in this reference:

- "PDExpansionManager" on page 51
- "PDExpansionCardInfo" on page 50
- "PDVFSManager" on page 90
- "PDVFSVolumeManager" on page 91

- "PDVFSManager" on page 90

See Chapter 5, "Using Expansion Technology," on page 71 in the *COM Sync Suite Companion* for a full discussion on using these objects.

PIM Database and Record Objects

These objects enable conduits to easily access the databases of four standard Palm OS® applications: Address Book, Date Book, To Do List, and Memo Pad. For each of these personal information management (PIM) applications, the COM Sync module includes a database adapter object to access its database and a record object to represent the fields in each record. These objects are detailed in this reference:

- "PDAddressDbHHRecordAdapter" on page 19
- "PDAddressDbHHRecord" on page 16
- "PDDateBookDbHHRecordAdapter" on page 40
- "PDDateBookDbHHRecord" on page 34
- "PDTodoDbHHRecordAdapter" on page 77
- "PDTodoDbHHRecord" on page 75
- "PDMemoDbHHRecordAdapter" on page 61
- "PDTodoDbHHRecord" on page 75

Changes in COM Sync Suite 4.01/4.02/4.02a

The COM Sync Suite for Windows was first released in CDK 4.01 and remained unchanged in CDK 4.02/4.02a.

B

Private Methods and Properties

Some methods and properties visible in the COM Sync objects are *not* supported for use in your conduit. This list is documented here only because you may see these names in your IDE's object browser and may not recognize the implications of using them.

The following private methods are in the `PD<PIM>DbHHRecord` objects:

- `ReadFromByteStream`
- `WriteToByteStream`

The following private property is in the `PDVFSVolumeManager` object:

- `CreatorCode`

Index