CHAPTER 9

Desktop Manager

Contents

About the Desktop Database 9-4
Using the Desktop Manager 9-4
Desktop Manager Reference 9-6
    Data Structure 9-6
        The Desktop Parameter Block 9-7
    Routines 9-8
        Locating, Opening, and Closing the Desktop Database 9-9
        Reading the Desktop Database 9-12
        Adding to the Desktop Database 9-17
        Deleting Entries From the Desktop Database 9-20
        Manipulating the Desktop Database Itself 9-23
Summary of the Desktop Manager 9-27
    Pascal Summary 9-27
        Constants 9-27
        Data Types 9-27
        Routines 9-28
    C Summary 9-30
        Constants 9-30
        Data Types 9-31
        Routines 9-31
    Assembly-Language Summary 9-34
        Data Structures 9-34
        Trap Macros 9-35
    Result Codes 9-35

For quick access to the resources it needs, the Finder maintains a central **desktop database** of information about the files and directories on a volume. The Finder updates the database when applications are added, moved, renamed, or deleted.

Normally, your application won't need to use the information in the desktop database or to use Desktop Manager routines to manipulate it. Instead, your application should let the Finder manipulate the desktop database and handle such Desktop Manager tasks as maintaining user comments associated with files and managing the icons used by applications.

▲ **WARNING**
Although there may be instances where you would like to gain access to the desktop database by using Desktop Manager routines, you should never change, add to, or remove any of this information. Manipulating the desktop database is likely to wreak havoc on your users' systems. ▲

In case you should discover some important need to retrieve information from the desktop database or even to change the desktop database from within your application, Desktop Manager routines are provided for you to do so. While your application probably won't ever need to use them, for the sake of completeness they are described in this chapter.

Much of the information in the desktop database comes from the bundle resources for applications and other files on the volume. (See the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials* for a discussion on setting the bundle bit of an application so that its bundled resources get stored in the desktop database.) The desktop database contains all icon definitions and their associated file types. It lists all the file types that each application can open and all copies or versions of the application that's listed as the creator of a file. The desktop database also lists the location of each application on the disk and any comments that the user has added to the information windows for desktop objects. The Desktop Manager provides routines that let your application retrieve this information from the desktop database.

The Finder maintains a desktop database for each volume with a capacity greater than 2 MB. For most volumes, such as hard disks, the database is stored on the volume itself. For read-only volumes—such as some compact discs—that don't contain their own desktop database, the Desktop Manager creates one for it and stores it in the System Folder of the boot drive.

Note
If you distribute read-only media, it is generally a good idea to store on each volume both a desktop database (for users running System 7 or later) and a Desktop file (for users running older versions of system software). Create a desktop database on your master volume by pressing Command-Option when booting your system with System 7. Then create a Desktop file by pressing Command-Option and restarting your system with version 6.0. ◆

For compatibility with older versions of system software, the Finder keeps the information for ejectable volumes with a capacity smaller than 2 MB in a resource file instead of in a database.

Although the Desktop Manager provides tools for both reading and changing the desktop database, your application should not ordinarily change anything in the database. You can read the database to retrieve information, such as the icons defined by other applications.

Note
The desktop database doesn't store customized icons (that is, those with resource IDs of –16455 described in the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*), so your application can't retrieve them by using Desktop Manager routines. ◆

If the Translation Manager is available, the desktop database also includes information retrieved from each application's `'open'` and `'kind'` resources. For information about the information stored in these resources, see the chapter "Translation Manager" in this book.

About the Desktop Database

In earlier versions of system software, Finder information for each volume was stored in the volume's Desktop file, a resource file created and used by the Finder and invisible to the user. This strategy meets the needs of a single-user system with reasonably small volumes. The Desktop file is still used on ejectable volumes with a capacity less than 2 MB so that these floppy disks can be shared with Macintosh computers running earlier versions of system software. (Note, however, that resources can't be shared. Since the Finder is always running in System 7, it keeps each floppy disk's Desktop file open, so your application can't read or write to it.)

Because resources can't be shared, a different strategy has been used for AppleShare volumes, which are available to multiple users over a network. The Desktop Manager in System 7 uses the strategy for large local volumes that AppleShare file servers have previously used for shared volumes. When a volume is first mounted, the Finder collects the bundle information from all applications on the disk and builds the desktop database. Whenever an application is added to or removed from the disk, the Finder updates the desktop database. Through Desktop Manager routines, the database is also accessible to any other application running on the system.

Using the Desktop Manager

You can manipulate the desktop database with a set of low-level routines that follow the parameter-block conventions used by the File Manager. The desktop database functions use a desktop parameter block (see page 9-7 for the structure of the desktop parameter block).

Because you cannot use the Desktop Manager functions on a disk that does not have a desktop database, call `PBHGetVolParms` to verify that the target disk has a desktop database before calling any of the Desktop Manager functions. (For a description of the `PBHGetVolParms` function and the `bHasDesktopMgr` bit that you should check, see the chapter "File Manager" in *Inside Macintosh: Files*.)

Because the Finder uses the desktop database, the database is almost always open. When the Desktop Manager opens the database, it assigns the database a reference number that represents the access path. You use the `PBDTGetPath` function to get the reference number, which you must specify when calling most other Desktop Manager functions. If the desktop database is not open, `PBDTGetPath` opens it.

If you are manipulating the database in the absence of the Finder, you can open the database with `PBDTOpenInform`, which performs the same functions as `PBDTGetPath` and also sets a flag to tell your application whether the desktop database was empty when it was opened. Your application should never close the database.

The Desktop Manager provides different functions for manipulating different kinds of information in the database. Not all manipulations are possible with all kinds of data.

You can retrieve five kinds of information from the database:

- icon definitions
- file types and icon types supported by a known creator
- name and location of applications with a known creator
- user comments for a file or a directory
- size and parent directory of the desktop database

To retrieve an icon definition, call `PBDTGetIcon`. You must specify a file creator, file type, and icon type. The database recognizes both large and small icons, with 1, 4, or 8 bits of color encoding. (For a description of these icons, see the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.)

To step through a list of all the icon types supported by an application, make repeated calls to `PBDTGetIconInfo`. Each time you call `PBDTGetIconInfo`, you specify a creator and an index value. Set the index to 1 on the first call, and increment it on each subsequent call until `PBDTGetIconInfo` returns the result code `afpItemNotFound`. For each entry in the icon list, `PBDTGetIconInfo` reports the icon type, the file type it is associated with, and the size of its icon data.

To identify the application that can open a file with a given creator, call `PBDTGetAPPL`. In each call to `PBDTGetAPPL`, you specify a creator (which is the application's signature) and an index value. An index value of 0 retrieves the "first choice" application—that is, the one with the most recent creation date. By setting the index to 1 on the first call and incrementing it on each subsequent call until `PBDTGetAPPL` returns the result code `afpItemNotFound`, you can make multiple calls to `PBDTGetAPPL` to find information about all copies or versions of the application with this signature on the disk. `PBDTGetAPPL` returns them all in arbitrary order. `PBDTGetAPPL` returns the name, parent directory ID, and creation date of each application in the desktop database.

To retrieve the user comments for a file or directory, call `PBDTGetComment`. The user can change comments at any time by typing in the comment box of the information window for any desktop object.

Your application should not ordinarily call the functions for adding and removing data to and from the database. If your application does need to write to or delete information from the desktop database, it must call `PBDTFlush` to update the copy stored on the volume.

The following list summarizes the data manipulation functions:

Kind of data	Read	Write	Remove
Icon definitions (for a given file type and creator)	`PBDTGetIcon`	`PBDTAddIcon`	—
Icon types (associated with each file type that an application supports)	`PBDTGetIconInfo`	—	—
Applications with a given creator	`PBDTGetAPPL`	`PBDTAddAPPL`	`PBDTRemoveAPPL`
User comments	`PBDTGetComment`	`PBDTSetComment`	`PBDTRemoveComment`
Entire desktop database	`PBDTGetInfo` (returns the size and parent directory of the database)	`PBDTFlush` (updates the copy stored on the volume)	`PBDTDelete` and `PBDTReset` (neither should be called by your application)

Desktop Manager Reference

This section describes the data structure and routines that are specific to the Desktop Manager. The “Data Structure” section describes the desktop parameter block, and the “Routines” section describes the routines your application can use to retrieve information from the desktop database.

Data Structure

You can manipulate the desktop database with a set of low-level routines that follow the parameter-block conventions used by the File Manager. (For more information on parameter blocks, see the chapter “File Manager” in *Inside Macintosh: Files*.) This section describes the parameter block you pass to Desktop Manager routines.

The Desktop Parameter Block

The desktop database functions use the desktop parameter block, a data structure of type DTPBRec:

```
TYPE  DTPBRec =
      RECORD
         qLink:          QElemPtr;    {next queue entry}
         qType:          Integer;     {queue type}
         ioTrap:         Integer;     {routine trap}
         ioCmdAddr:      Ptr;         {routine address}
         ioCompletion:   ProcPtr;     {completion routine}
         ioResult:       OSErr;       {result code}
         ioNamePtr:      StringPtr;   {file, directory, or }
                                      { volume name}
         ioVRefNum:      Integer;     {volume reference number}
         ioDTRefNum:     Integer;     {desktop database reference }
                                      { number}
         ioIndex:        Integer;     {index into icon list}
         ioTagInfo:      LongInt;     {tag information}
         ioDTBuffer:     Ptr;         {data buffer}
         ioDTReqCount:   LongInt;     {requested length of data}
         ioDTActCount:   LongInt;     {actual length of data}
         filler1:        SignedByte;  {unused}
         ioIconType:     SignedByte;  {icon type}
         filler2:        Integer;     {unused}
         ioDirID:        LongInt;     {parent directory ID}
         ioFileCreator:  OSType;      {file creator}
         ioFileType:     OSType;      {file type}
         ioFiller3:      LongInt;     {unused}
         ioDTLgLen:      LongInt;     {logical length of desktop }
                                      { database}
         ioDTPyLen:      LongInt;     {physical length of desktop }
                                      { database}
         ioFiller4:                   {unused}
                         ARRAY[1..14] OF Integer;
         ioAPPLParID:    LongInt      {parent directory ID of }
                                      { application}
      END;
      DTPBPtr = ^DTPBRec;             {pointer to desktop }
                                      { parameter block}
```

For a description of the standard fields of a parameter block (`qLink`, `qType`, `ioTrap`, `ioCmdAddr`, `ioCompletion`, and `ioResult`), see the chapter "File Manager" in *Inside Macintosh: Files*. For other fields of the desktop parameter block, see the relevant routine description provided in the next section.

Routines

This section describes the low-level routines for using the desktop database.

All low-level routines exchange parameters with your application through a parameter block. When calling a low-level routine, you pass a pointer to the parameter block. See the chapter "File Manager" in *Inside Macintosh: Files* for a description of the standard fields in a parameter block.

IMPORTANT

Clear all fields (other than input fields) in the parameter block that you pass to Desktop Manager routines. ▲

Three Desktop Manager functions—namely, `PBDTGetPath`, `PBDTOpenInform`, and `PBDTCloseDown`—run synchronously only. All other Desktop Manager routines can run either asynchronously or synchronously. There are three versions of each of these routines. The first version takes two parameters: a pointer to the parameter block, and a Boolean value that determines whether the routine is run asynchronously (`TRUE`) or synchronously (`FALSE`). Here, for example, is the first version of a routine that retrieves the user's comment stored for a file or a directory:

```
FUNCTION PBDTGetComment (paramBlock: DTPBPtr;
                         async: Boolean): OSErr;
```

The second version does not take a second parameter; instead, it adds the suffix "Async" to the name of the routine.

```
FUNCTION PBDTGetCommentAsync (paramBlock: DTPBPtr): OSErr;
```

Similarly, the third version of the routine does not take a second parameter; instead, it adds the suffix "Sync" to the name of the routine.

```
FUNCTION PBDTGetCommentSync (paramBlock: DTPBPtr): OSErr;
```

All routines in this section are documented using the first version only. Note, however, that the second and third versions of these routines do not use the glue code that the first versions use and are therefore more efficient.

IMPORTANT

All of the Desktop Manager routines may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt. Your application should not call Desktop Manager routines at interrupt time. ▲

Because you cannot use the Desktop Manager functions on a disk that does not have a desktop database, call `PBHGetVolParms` to verify that the target disk has a desktop database before calling any of the Desktop Manager functions. (For a description of the `PBHGetVolParms` function and the `bHasDesktopMgr` bit that you should check, see the chapter “File Manager” in *Inside Macintosh: Files*.)

▲ **WARNING**
Although routines that set information in and get information from the desktop database are described in this section, you should never use these routines to change, add to, or remove any information from the desktop database. Manipulating the desktop database is likely to wreak havoc on your users’ systems. ▲

Assembly-Language Note
You can invoke each of the Desktop Manager routines with a macro that has the same name as the routine, preceded by an underscore. These macros, however, aren’t really trap macros. Instead, they expand to invoke the trap macro `_HFSDispatch`. The File Manager determines which routine to execute from the routine selector, an integer placed in register D0. The routine selectors appear in “Assembly-Language Summary” beginning on page 9-34. ◆

Locating, Opening, and Closing the Desktop Database

To get the access path to a database or to create a database if one doesn’t exist, use the `PBDTGetPath` or `PBDTOpenInform` function. These routines run synchronously only. System software uses the `PBDTCloseDown` function to close the desktop database; your application should never use this function, which is described in this section only for completeness.

PBDTGetPath

You can get the reference number of the desktop database using the `PBDTGetPath` function.

```
FUNCTION PBDTGetPath (paramBlock: DTPBPtr): OSErr;
```

`paramBlock` A pointer to a desktop parameter block.

Parameter block

←	`ioResult`	`OSErr`	The result code of the function.
→	`ioNamePtr`	`StringPtr`	A pointer to the volume name or full pathname of the desktop database.
→	`ioVRefNum`	`Integer`	The volume reference number of the desktop database.
←	`ioDTRefNum`	`Integer`	The desktop database reference number.

DESCRIPTION

The `PBDTGetPath` function returns the desktop database reference number in the `ioDTRefNum` field, which represents the access path to the database. You specify the volume by passing a pointer to its name in the `ioNamePtr` field or a volume reference number in the `ioVRefNum` field. If the desktop database is not already open, `PBDTGetPath` opens it and then returns the reference number. If the desktop database doesn't exist, `PBDTGetPath` creates it. If `PBDTGetPath` fails, it sets the `ioDTRefNum` field to 0.

Note

You cannot use the desktop reference number as a file reference number in any File Manager routines. ◆

▲ **WARNING**

Do not call `PBDTGetPath` at interrupt time—it allocates memory in the system heap. ▲

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`extFSErr`	–58	External file system—file system identifier is nonzero
`desktopDamagedErr`	–1305	The desktop database has become corrupted—the Finder will fix this, but if your application is not running with the Finder, use `PBDTReset` or `PBDTDelete`

PBDTOpenInform

The `PBDTOpenInform` function performs the same function as `PBDTGetPath`, but it also reports whether the desktop database was empty when it was opened.

```
FUNCTION PBDTOpenInform (paramBlock: DTPBPtr): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

Parameter block

←	`ioResult`	`OSErr`	The result code of the function.
→	`ioNamePtr`	`StringPtr`	A pointer to the volume name or full pathname of the desktop database.
→	`ioVRefNum`	`Integer`	The volume reference number of the desktop database.
←	`ioDTRefNum`	`Integer`	The desktop database reference number.
←	`ioTagInfo`	`LongInt`	The return flag (in low bit).

DESCRIPTION

If the desktop database was just created in response to `PBDTOpenInform` (and is therefore empty), `PBDTOpenInform` sets the low bit in the `ioTagInfo` field to 0. If the desktop database had been created before you called `PBDTOpenInform`, `PBDTOpenInform` sets the low bit in the `ioTagInfo` field to 1.

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`paramErr`	–50	Parameter error; use `PBDTGetPath`
`extFSErr`	–58	External file system—file system identifier is nonzero
`desktopDamagedErr`	–1305	The desktop database has become corrupted—the Finder will fix this, but if your application is not running with the Finder, use `PBDTReset` or `PBDTDelete`

PBDTCloseDown

The `PBDTCloseDown` function is used by system software to close the desktop database, though your application should never do this itself. `PBDTCloseDown` runs synchronously only, and though it will not close down the desktop databases of remote volumes, it will invalidate all local `DTRefNum` values for remote desktop databases.

```
FUNCTION PBDTCloseDown (paramBlock: DTPBPtr): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

Parameter block

←	`ioResult`	`OSErr`	The result code of the function.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.

DESCRIPTION

The `PBDTCloseDown` function closes the database specified in `ioDTRefNum` and frees all resources allocated by `PBDTOpenInform` or `PBDTGetPath`.

▲ **WARNING**
Applications should not call `PBDTCloseDown`. The system software closes the database when the volume is unmounted. ▲

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero

Reading the Desktop Database

You can get information from the desktop database, such as a specific icon that represents a file of a given type and creator or the user comments associated with a file, by using the routines described in this section.

PBDTGetIcon

To retrieve an icon definition, use the `PBDTGetIcon` function.

```
FUNCTION PBDTGetIcon (paramBlock: DTPBPtr; async: Boolean): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

`async` A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioTagInfo`	`LongInt`	Reserved; must be set to 0.
→	`ioDTBuffer`	`Ptr`	A pointer to a buffer to hold the icon's data.
→	`ioDTReqCount`	`LongInt`	The requested size of the icon's bitmap.
←	`ioDTActCount`	`LongInt`	The actual size of the icon's bitmap.
→	`ioIconType`	`SignedByte`	The icon type.
→	`ioFileCreator`	`OSType`	The icon's file creator.
→	`ioFileType`	`OSType`	The icon's file type.

DESCRIPTION

The `PBDTGetIcon` function returns the bitmap for an icon that represents a file of a given type and creator. (For example, to get the icon for a file of file type `'SFWR'` created by the application with a signature of `'WAVE'`, specify these two values in `ioFileType` and `ioFileCreator`.) You pass a pointer to a buffer for the icon bitmap in the `ioDTBuffer` field. The bitmap is returned in this buffer. You specify the desktop database in `ioDTRefNum`, the file creator in `ioFileCreator`, and the file type in `ioFileType`. For the icon type in `ioIconType`, specify a constant from the following list:

Constant	Value	Corresponding resource type	Description
`kLargeIcon`	1	`'ICN#'`	Large black-and-white icon with mask
`kLarge4BitIcon`	2	`'icl4'`	Large 4-bit color icon
`kLarge8BitIcon`	3	`'icl8'`	Large 8-bit color icon
`kSmallIcon`	4	`'ics#'`	Small black-and-white icon with mask
`kSmall4BitIcon`	5	`'ics4'`	Small 4-bit color icon
`kSmall8BitIcon`	6	`'ics8'`	Small 8-bit color icon

The value you supply in `ioDTReqCount` is the size in bytes of the buffer that you've allocated for the icon's bitmap pointed to by `ioDTBuffer`; this value depends on the icon type. Be sure to allocate enough storage for the icon data; 1024 bytes is the largest amount required for any icon in System 7. You can use constants to indicate the amount of memory you have provided for the icon's data. The following list shows these constants and, for each icon type, shows the amount of bytes you should allocate.

Constant	Value (bytes)	Corresponding resource type	Description
`kLargeIconSize`	256	`'ICN#'`	Large black-and-white icon with mask
`kLarge4BitIconSize`	512	`'icl4'`	Large 4-bit color icon
`kLarge8BitIconSize`	1024	`'icl8'`	Large 8-bit color icon
`kSmallIconSize`	64	`'ics#'`	Small black-and-white icon with mask
`kSmall4BitIconSize`	128	`'ics4'`	Small 4-bit color icon
`kSmall8BitIconSize`	256	`'ics8'`	Small 8-bit color icon

The value in `ioDTActCount` reflects the size of the bitmap actually retrieved. If `ioDTActCount` is larger than `ioDTReqCount`, only the amount of data allowed by `ioDTReqCount` is valid.

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero
`afpItemNotFound`	–5012	Information not found

PBDTGetIconInfo

You can iteratively generate a list of icon types associated with each file type supported by an application by repeatedly calling the `PBDTGetIconInfo` function.

```
FUNCTION PBDTGetIconInfo (paramBlock: DTPBPtr;
                          async: Boolean): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

`async` A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioIndex`	`Integer`	An index into the icon list.
→	`ioTagInfo`	`LongInt`	Reserved; must be set to 0.
←	`ioDTActCount`	`LongInt`	The size of the icon's bitmap.
←	`ioIconType`	`SignedByte`	The icon type.
→	`ioFileCreator`	`OSType`	The icon's file creator.
←	`ioFileType`	`OSType`	The icon's file type.

DESCRIPTION

The `PBDTGetIconInfo` function retrieves an icon type and the associated file type supported by a given creator in the database. You use it to identify the set of icons associated with each file type that is supported by a given creator. You specify the creator by placing its signature in `ioFileCreator`, and you specify the database by placing the desktop database reference number in the `ioDTRefNum` field. The `PBDTGetIconInfo` function returns the size of the bitmap in `ioDTActCount`, the file type in `ioFileType`, and the icon size and color depth in `ioIconType`.

The `PBDTGetIconInfo` function can return in the `ioIconType` field any of the values listed in the description of the `PBDTGetIcon` function on page 9-12. Ignore any values returned in `ioIconType` that are not listed there; they represent special icons and information used only by the Finder.

To step through a list of the icon types and file types supported by an application, make repeated calls to `PBDTGetIconInfo`, specifying a creator and an index value for `ioIndex` each call. Set the index to 1 on the first call, and increment it on each subsequent call until `ioResult` returns `afpItemNotFound`.

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero
`afpItemNotFound`	–5012	Information not found

SEE ALSO

To get a list of file types that an application can natively open, you can use the `GetFileTypesThatAppCanNativelyOpen` function, as described in the chapter "Translation Manager" of this book.

PBDTGetAPPL

To identify the application that can open a file with a given creator, use the `PBDTGetAPPL` function.

```
FUNCTION PBDTGetAPPL (paramBlock: DTPBPtr; async: Boolean): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

`async` A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code.
→	`ioNamePtr`	`StringPtr`	A pointer to the application's name.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioIndex`	`Integer`	An index into the application list.
←	`ioTagInfo`	`LongInt`	The application's creation date.
→	`ioFileCreator`	`OSType`	The application's signature.
←	`ioAPPLParID`	`LongInt`	The application's parent directory.

DESCRIPTION

For an application in the database specified in `ioDTRefNum` with the signature specified in `ioFileCreator`, `PBDTGetAPPL` returns the filename in `ioNamePtr`, the parent directory ID in `ioAPPLParID`, and the creation date in `ioTagInfo`. A single call, with `ioIndex` set to 0, finds the application file with the most recent creation date. If you want to retrieve information about all copies of the application with the given signature, start with `ioIndex` set to 1 and increment until `ioResult` returns `afpItemNotFound`; when called multiple times in this fashion, `PBDTGetAPPL` returns information about all the application's copies, including the file with the most recent creation date, in arbitrary order.

RESULT CODES

noErr	0	No error
ioErr	–36	I/O error
rfNumErr	–51	Reference number invalid
extFSErr	–58	External file system—file system identifier is nonzero
afpItemNotFound	–5012	Information not found

PBDTGetComment

To retrieve the user comments for a file or directory, use the `PBDTGetComment` function.

```
FUNCTION PBDTGetComment (paramBlock: DTPBPtr;
                         async: Boolean): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

`async` A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	ioCompletion	ProcPtr	A pointer to a completion routine.
←	ioResult	OSErr	The result code of the function.
→	ioNamePtr	StringPtr	A pointer to a file or directory name.
→	ioDTRefNum	Integer	The desktop database reference number.
→	ioDTBuffer	Ptr	A pointer to comment text (200 bytes).
←	ioDTActCount	LongInt	The comment size.
→	ioDirID	LongInt	The parent directory of the file or directory.

DESCRIPTION

The `PBDTGetComment` function retrieves the comment stored for a file or directory in the database specified in `ioDTRefNum`. You specify the filename or directory name and its parent directory ID through `ioNamePtr` and `ioDirID`. You allocate a buffer big enough to hold the largest comment, 200 bytes, and put a pointer to it in the `ioDTBuffer` field. The `PBDTGetComment` function places the comment in the buffer as a plain text string and places the length of the comment in `ioDTActCount`.

RESULT CODES

noErr	0	No error
ioErr	–36	I/O error
fnfErr	–43	File or directory doesn’t exist
rfNumErr	–51	Reference number invalid
extFSErr	–58	External file system—file system identifier is nonzero
afpItemNotFound	–5012	Information not found

Adding to the Desktop Database

Your application should not ordinarily call the functions for adding data to the database. If your application does need to write to or delete information from the desktop database, it must call `PBDTFlush` to update the copy stored on the volume.

PBDTAddIcon

To add an icon definition to the desktop database, use the `PBDTAddIcon` function.

```
FUNCTION PBDTAddIcon (paramBlock: DTPBPtr; async: Boolean): OSErr;
```

`paramBlock`
: A pointer to a desktop parameter block.

`async`
: A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioTagInfo`	`LongInt`	Reserved; must be set to 0.
→	`ioDTBuffer`	`Ptr`	A pointer to the icon's data.
→	`ioDTReqCount`	`LongInt`	The size of the icon's bitmap.
→	`ioIconType`	`SignedByte`	The icon type.
→	`ioFileCreator`	`OSType`	The icon's file creator.
→	`ioFileType`	`OSType`	The icon's file type.

DESCRIPTION

The `PBDTAddIcon` function adds an icon definition to the desktop database specified in `ioDTRefNum`. You specify the creator and file type that the icon is associated with in the `ioFileCreator` and `ioFileType` fields. For the icon type in `ioIconType`, specify either a constant or a value from the following list.

Constant	Value	Corresponding resource type	Description
`kLargeIcon`	1	`'ICN#'`	Large black-and-white icon with mask
`kLarge4BitIcon`	2	`'icl4'`	Large 4-bit color icon
`kLarge8BitIcon`	3	`'icl8'`	Large 8-bit color icon
`kSmallIcon`	4	`'ics#'`	Small black-and-white icon with mask
`kSmall4BitIcon`	5	`'ics4'`	Small 4-bit color icon
`kSmall8BitIcon`	6	`'ics8'`	Small 8-bit color icon

The value you supply in `ioDTReqCount` is the size in bytes of the buffer that you've allocated for the icon's bitmap pointed to by `ioDTBuffer`; this value depends on the icon type. Be sure to allocate enough storage for the icon data; 1024 bytes is the largest amount required for any icon in System 7. For the number of bytes in `ioDTReqCount`, you specify either a constant or a value from the following list.

Constant	Value (bytes in bitmap)	Corresponding resource type	Description
`kLargeIconSize`	256	`'ICN#'`	Large black-and-white icon with mask
`kLarge4BitIconSize`	512	`'icl4'`	Large 4-bit color icon
`kLarge8BitIconSize`	1024	`'icl8'`	Large 8-bit color icon
`kSmallIconSize`	64	`'ics#'`	Small black-and-white icon with mask
`kSmall4BitIconSize`	128	`'ics4'`	Small 4-bit color icon
`kSmall8BitIconSize`	256	`'ics8'`	Small 8-bit color icon

You pass a pointer to the icon bitmap in the `ioDTBuffer` field. You must initialize the `ioTagInfo` field to 0.

If the database already contains an icon definition for an icon of that type, file type, and file creator, the new definition replaces the old.

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero
`afpIconTypeError`	–5030	Sizes of new icon and one it replaces don't match

PBDTAddAPPL

To add an application to the desktop database, use the `PBDTAddAPPL` function.

```
FUNCTION PBDTAddAPPL (paramBlock: DTPBPtr; async: Boolean): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

`async` A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioNamePtr`	`StringPtr`	A pointer to the application's name.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioTagInfo`	`LongInt`	Reserved; must be set to 0.
→	`ioDirID`	`LongInt`	The application's parent directory.
→	`ioFileCreator`	`OSType`	The application's signature.

DESCRIPTION

The `PBDTAddAPPL` function adds an entry in the desktop database specified in `ioDTRefNum` for an application with the specified signature. You pass the application's signature in `ioFileCreator`, a pointer to the application's filename in `ioNamePtr`, and the application's parent directory ID in `ioDirID`. Initialize `ioTagInfo` to 0.

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`fnfErr`	–43	Application not present on volume
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero

PBDTSetComment

To add a user comment for a file or a directory to the desktop database, use the `PBDTSetComment` function.

```
FUNCTION PBDTSetComment (paramBlock: DTPBPtr;
                         async: Boolean): OSErr;
```

`paramBlock`
: A pointer to a desktop parameter block.

`async`
: A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioNamePtr`	`StringPtr`	A pointer to a file or directory name.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioDTBuffer`	`Ptr`	A pointer to the comment text.
→	`ioDTReqCount`	`LongInt`	The comment length.
→	`ioDirID`	`LongInt`	The parent directory of the file or directory.

DESCRIPTION

The `PBDTSetComment` function establishes the user comment associated with a file or directory in the database specified in `ioDTRefNum`. You specify the object name through `ioNamePtr` and the parent directory ID in `ioDirID`. You put the comment as a plain text string in a buffer pointed to by `ioDTBuffer`, and you specify the length of the buffer (in bytes) in `ioDTReqCount`. The maximum length of a comment is 200 bytes; longer comments are truncated. Since the comment is a plain text string and not a Pascal string, the Desktop Manager relies on the value in `ioDTReqCount` for determining the length of the buffer.

If the specified object already has a comment in the database, the new comment replaces the old.

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`fnfErr`	–43	File or directory doesn’t exist
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero

Deleting Entries From the Desktop Database

Your application should not ordinarily call the functions for adding and removing data to and from the database. If your application does need to write to or delete information from the desktop database, it must call `PBDTFlush` to update the copy stored on the volume.

PBDTRemoveAPPL

To remove an application from the desktop database, call the `PBDTRemoveAPPL` function.

```
FUNCTION PBDTRemoveAPPL (paramBlock: DTPBPtr;
                         async: Boolean): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

`async` A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioNamePtr`	`StringPtr`	A pointer to the application's name.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioDirID`	`LongInt`	The application's parent directory.
→	`ioFileCreator`	`OSType`	The application's signature.

DESCRIPTION

The `PBDTRemoveAPPL` function removes the mapping information for an application from the database specified in `ioDTRefNum`. You specify the application's name through `ioNamePtr`, its parent directory ID in `ioDirID`, and its signature in `ioFileCreator`.

You can call `PBDTRemoveAPPL` even if the application is not present on the volume.

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero
`afpItemNotFound`	–5012	Application not found

PBDTRemoveComment

To remove a user comment from the desktop database, call the `PBDTRemoveComment` function.

```
FUNCTION PBDTRemoveComment (paramBlock: DTPBPtr;
                            async: Boolean): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

`async` A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioNamePtr`	`StringPtr`	A pointer to the filename or directory name.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioDirID`	`LongInt`	The parent directory of the file or directory.

DESCRIPTION

The `PBDTRemoveComment` function removes the comment associated with a file or directory from the database specified in `ioDTRefNum`. You specify the file or directory name through `ioNamePtr` and the parent directory ID in `ioDirID`. You cannot remove a comment if the file or directory is not present on the volume. If no comment was stored for the file, `PBDTRemoveComment` returns an error.

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`fnfErr`	–43	File or directory doesn’t exist
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero
`afpItemNotFound`	–5012	Comment not found

Manipulating the Desktop Database Itself

If your application adds information to or removes information from the desktop database, use the `PBDTFlush` function to save your changes. To get information about the desktop database itself, use the `PBDTGetInfo` function. The `PBDTReset` function removes all information from the desktop database, and the `PBDTDelete` function removes the desktop database; you should not use these two functions unless absolutely necessary.

PBDTFlush

To save your changes to the desktop database, use the `PBDTFlush` function.

```
FUNCTION PBDTFlush (paramBlock: DTPBPtr; async: Boolean): OSErr;
```

`paramBlock`	A pointer to a desktop parameter block.
`async`	A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.

DESCRIPTION

The `PBDTFlush` function writes the contents of the desktop database specified in `ioDTRefNum` to the volume.

Note

If your application has manipulated information in the database using any of the routines described in "Adding to the Desktop Database" or "Deleting Entries From the Desktop Database" beginning on page 9-17 and page 9-20, respectively, you must call `PBDTFlush` to update the copy stored on the volume. ◆

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero

PBDTGetInfo

To determine the parent directory and the amount of space used by the desktop database on a particular volume, use the `PBDTGetInfo` function.

```
FUNCTION PBDTGetInfo (paramBlock: DTPBPtr; async: Boolean): OSErr;
```

`paramBlock`	A pointer to a desktop parameter block.
`async`	A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
←	`ioVRefNum`	`Integer`	The volume reference number where the database files are actually stored.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
←	`ioIndex`	`Integer`	The number of files comprising the desktop database on the volume.
←	`ioDirID`	`LongInt`	The parent directory of the desktop database.
←	`ioDTLgLen`	`LongInt`	The logical length of the database files.
←	`ioDTPyLen`	`LongInt`	The physical length of the database files.

DESCRIPTION

The `PBDTGetInfo` function returns information about the desktop database. You specify the volume of the desktop database in `ioDTRefNum`. The parent directory of the desktop database for the volume is returned in `ioDirID`. The sum of the logical lengths of the files that constitute the desktop database for a given volume is returned in `ioDTLgLen`; the sum of the physical lengths of the files that constitute the desktop database for a given volume is returned in `ioDTPyLen`. The number of files comprising the desktop database is returned in `ioIndex`.

RESULT CODES

`noErr`	0	No err
`nsvErr`	–35	No such volume
`ioErr`	–36	I/O error
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero

PBDTReset

The `PBDTReset` function removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function.

```
FUNCTION PBDTReset (paramBlock: DTPBPtr; async: Boolean): OSErr;
```

`paramBlock`
A pointer to a desktop parameter block.

`async` A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioDTRefNum`	`Integer`	The desktop database reference number.
→	`ioIndex`	`Integer`	Reserved; must be set to 0.

DESCRIPTION

The `PBDTReset` function removes all icons, application mappings, and comments from the desktop database specified in `ioDTRefNum`. You can call `PBDTReset` only when the database is open. It remains open after the data is cleared.

IMPORTANT
Your application should never call `PBDTReset`. ▲

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero

PBDTDelete

The `PBDTDelete` function removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function.

```
FUNCTION PBDTDelete (paramBlock: DTPBPtr; async: Boolean): OSErr;
```

`paramBlock`
: A pointer to a desktop parameter block.

`async`
: A Boolean value that specifies asynchronous (`TRUE`) or synchronous (`FALSE`) execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The result code of the function.
→	`ioVRefNum`	`Integer`	The volume reference number of the desktop database.
→	`ioIndex`	`Integer`	Reserved; must be set to 0.

DESCRIPTION

The `PBDTDelete` function removes the desktop database from a local volume. You specify the volume by passing a volume reference number in `ioVRefNum`. You can call `PBDTDelete` only when the database is closed.

IMPORTANT

Your application should never call `PBDTDelete`. ▲

RESULT CODES

`noErr`	0	No error
`ioErr`	–36	I/O error
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero

Summary of the Desktop Manager

Pascal Summary

Constants

```
CONST
      {for mapping icons to ioIconType in the desktop database}
      kLargeIcon                   = 1;        {'ICN#'}
      kLarge4BitIcon               = 2;        {'icl4'}
      kLarge8BitIcon               = 3;        {'icl8'}
      kSmallIcon                   = 4;        {'ics#'}
      kSmall4BitIcon               = 5;        {'ics4'}
      kSmall8BitIcon               = 6;        {'ics8'}

      {for allocating storage for icon data in the desktop database}
      kLargeIconSize               = 256;      {'ICN#'}
      kLarge4BitIconSize           = 512;      {'icl4'}
      kLarge8BitIconSize           = 1024;     {'icl8'}
      kSmallIconSize               = 64;       {'ics#'}
      kSmall4BitIconSize           = 128;      {'ics4'}
      kSmall8BitIconSize           = 256;      {'ics8'}
```

Data Types

```
TYPE  DTPBPtr = ^DTPBRec;
      DTPBRec =                    {parameter block for desktop database}
      RECORD
         qLink:            QElemPtr;      {next queue entry}
         qType:            Integer;       {queue type}
         ioTrap:           Integer;       {routine trap}
         ioCmdAddr:        Ptr;           {routine address}
         ioCompletion:     ProcPtr;       {completion routine}
         ioResult:         OSErr;         {result code}
         ioNamePtr:        StringPtr;     {file, directory, or volume name}
         ioVRefNum:        Integer;       {volume reference number}
         ioDTRefNum:       Integer;       {desktop database reference number}
         ioIndex:          Integer;       {index into icon list}
```

```
   ioTagInfo:        LongInt;       {tag information}
   ioDTBuffer:       Ptr;           {data buffer}
   ioDTReqCount:     LongInt;       {requested length of data}
   ioDTActCount:     LongInt;       {actual length of data}
   filler1:          SignedByte;    {unused}
   ioIconType:       SignedByte;    {icon type}
   filler2:          Integer;       {unused}
   ioDirID:          LongInt;       {parent directory ID}
   ioFileCreator:    OSType;        {file creator}
   ioFileType:       OSType;        {file type}
   ioFiller3:        LongInt;       {unused}
   ioDTLgLen:        LongInt;       {logical length of desktop }
                                    { database}
   ioDTPyLen:        LongInt;       {physical length of desktop }
                                    { database}
   ioFiller4:                       {unused}
                     ARRAY[1..14] OF Integer;
   ioAPPLParID:      LongInt        {parent directory ID of }
                                    { application}
END;
DTPBPtr = ^DTPBRec;                 {pointer to desktop }
                                    { parameter block}
```

Routines

Locating, Opening, and Closing the Desktop Database

```
FUNCTION PBDTGetPath        (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTOpenInform     (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTCloseDown      (paramBlock: DTPBPtr): OSErr;
```

Reading the Desktop Database

```
FUNCTION PBDTGetIcon        (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTGetIconAsync   (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTGetIconSync    (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTGetIconInfo    (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTGetIconInfoAsync
                            (paramBlock: DTPBPtr): OSErr;
```

```
FUNCTION PBDTGetIconInfoSync
                             (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTGetAPPL         (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTGetAPPLAsync    (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTGetAPPLSync     (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTGetComment      (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTGetCommentAsync
                             (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTGetCommentSync
                             (paramBlock: DTPBPtr): OSErr;
```

Adding to the Desktop Database

```
FUNCTION PBDTAddIcon         (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTAddIconAsync    (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTAddIconSync     (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTAddAPPL         (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTAddAPPLAsync    (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTAddAPPLSync     (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTSetComment      (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTSetCommentAsync
                             (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTSetCommentSync
                             (paramBlock: DTPBPtr): OSErr;
```

Deleting Entries From the Desktop Database

```
FUNCTION PBDTRemoveAPPL      (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTRemoveAPPLAsync
                             (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTRemoveAPPLSync
                             (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTRemoveComment   (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTRemoveCommentAsync
                             (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTRemoveCommentSync
                             (paramBlock: DTPBPtr): OSErr;
```

Manipulating the Desktop Database Itself

```
FUNCTION PBDTFlush          (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTFlushAsync     (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTFlushSync      (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTGetInfo        (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTGetInfoAsync   (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTGetInfoSync    (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTReset          (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTResetAsync     (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTResetSync      (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTDelete         (paramBlock: DTPBPtr; async: Boolean): OSErr;
FUNCTION PBDTDeleteAsync    (paramBlock: DTPBPtr): OSErr;
FUNCTION PBDTDeleteSync     (paramBlock: DTPBPtr): OSErr;
```

C Summary

Constants

```
enum {
      /*for mapping icons to ioIconType in the desktop database*/
      kLargeIcon                   = 1,     /*'ICN#'*/
      kLarge4BitIcon               = 2,     /*'icl4'*/
      kLarge8BitIcon               = 3,     /*'icl8'*/
      kSmallIcon                   = 4,     /*'ics#'*/
      kSmall4BitIcon               = 5,     /*'ics4'*/
      kSmall8BitIcon               = 6,     /*'ics8'*/

      /*for allocating storage for icon data in the desktop database*/
      kLargeIconSize               = 256,   /*'ICN#'*/
      kLarge4BitIconSize           = 512,   /*'icl4'*/
      kLarge8BitIconSize           = 1024,  /*'icl8'*/
      kSmallIconSize               = 64,    /*'ics#'*/
      kSmall4BitIconSize           = 128,   /*'ics4'*/
      kSmall8BitIconSize           = 256    /*'ics8'*/
};
```

Data Types

```
struct DTPBRec {           /*parameter block for desktop database*/
   ParamBlockHeader
   short    ioDTRefNum;                      /*desktop refnum*/
   short    ioIndex;                         /*index into icon list*/
   long     ioTagInfo;                       /*tag information*/
   Ptr      ioDTBuffer;                      /*data buffer*/
   long     ioDTReqCount;                    /*requested length of data*/
   long     ioDTActCount;                    /*actual length of data*/
   char     ioFiller1;                       /*unused*/
   char     ioIconType;                      /*icon type*/
   short    ioFiller2;                       /*unused*/
   long     ioDirID;                         /*parent directory ID*/
   OSType   ioFileCreator;                   /*file creator*/
   OSType   ioFileType;                      /*file type*/
   long     ioFiller3;                       /*unused*/
   long     ioDTLgLen;                       /*logical length of */
                                             /* desktop database*/
   long     ioDTPyLen;                       /*physical length of desktop */
                                             /* database*/
   short    ioFiller4[14];                   /*unused*/
   long     ioAPPLParID;                     /*parent directory ID of */
                                             /* application*/
};
typedef struct DTPBRec DTPBRec;
typedef DTPBRec *DTPBPtr;
```

Routines

Locating, Opening, and Closing the Desktop Database

```
pascal OSErr PBDTGetPath    (DTPBPtr paramBlock);
pascal OSErr PBDTOpenInform
                            (DTPBPtr paramBlock);
pascal OSErr PBDTCloseDown  (DTPBPtr paramBlock);
```

Reading the Desktop Database

```
pascal OSErr PBDTGetIcon    (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTGetIconAsync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTGetIconSync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTGetIconInfo
                            (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTGetIconInfoAsync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTGetIconInfoSync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTGetAPPL    (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTGetAPPLAsync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTGetAPPLSync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTGetComment
                            (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTGetCommentAsync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTGetCommentSync
                            (DTPBPtr paramBlock);
```

Adding to the Desktop Database

```
pascal OSErr PBDTAddIcon    (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTAddIconAsync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTAddIconSync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTAddAPPL    (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTAddAPPLAsync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTAddAPPLSync
                            (DTPBPtr paramBlock);
pascal OSErr PBDTSetComment
                            (DTPBPtr paramBlock, Boolean async);
```

```
pascal OSErr PBDTSetCommentAsync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTSetCommentSync
                             (DTPBPtr paramBlock);
```

Deleting Entries From the Desktop Database

```
pascal OSErr PBDTRemoveAPPL
                             (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTRemoveAPPLAsync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTRemoveAPPLSync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTRemoveComment
                             (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTRemoveCommentAsync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTRemoveCommentSync
                             (DTPBPtr paramBlock);
```

Manipulating the Desktop Database Itself

```
pascal OSErr PBDTFlush       (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTFlushAsync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTFlushSync   (DTPBPtr paramBlock);
pascal OSErr PBDTGetInfo     (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTGetInfoAsync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTGetInfoSync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTReset       (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTResetAsync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTResetSync   (DTPBPtr paramBlock);
pascal OSErr PBDTDelete      (DTPBPtr paramBlock, Boolean async);
pascal OSErr PBDTDeleteAsync
                             (DTPBPtr paramBlock);
pascal OSErr PBDTDeleteSync
                             (DTPBPtr paramBlock);
```

Assembly-Language Summary

Data Structures

Desktop Parameter Block

→	12	`ioCompletion`	long	completion routine
←	16	`ioResult`	word	result code
→	18	`ioNamePtr`	long	pointer to file, directory, or volume name
→	22	`ioVRefNum`	word	volume reference number
↔	24	`ioDTRefNum`	word	desktop database reference number
→	26	`ioIndex`	word	index into icon list; or number of files in database
↔	28	`ioTagInfo`	long	tag information
→	32	`ioDTBuffer`	long	pointer to icon data
→	36	`ioDTReqCount`	long	requested size of icon data buffer
←	40	`ioDTActCount`	long	actual size of icon definition
↔	44	`ioIconType`	byte	icon’s type
↔	48	`ioDirID`	long	parent directory
→	52	`ioFileCreator`	long	file creator
↔	56	`ioFileType`	long	file type
←	64	`ioDTLgLen`	long	logical length of database files
←	68	`ioDTPyLen`	long	physical length of database files
←	100	`ioAPPLParID`	long	application’s parent directory

Trap Macros

Trap Macros Requiring Routine Selectors

`_HFSDispatch`

Selector	Routine
$0020	`PBDTGetPath`
$0021	`PBDTCloseDown`
$0022	`PBDTAddIcon`
$0023	`PBDTGetIcon`
$0024	`PBDTGetIconInfo`
$0025	`PBDTAddAPPL`
$0026	`PBDTRemoveAPPL`
$0027	`PBDTGetAPPL`
$0028	`PBDTSetComment`
$0029	`PBDTRemoveComment`
$002A	`PBDTGetComment`
$002B	`PBDTFlush`
$002C	`PBDTReset`
$002D	`PBDTGetInfo`
$002E	`PBDTOpenInform`
$002F	`PBDTDelete`

Result Codes

`noErr`	0	No error
`nsvErr`	–35	No such volume
`ioErr`	–36	I/O error
`fnfErr`	–43	File or directory doesn’t exist
`wPrErr`	–44	Volume is locked through hardware
`vLckdErr`	–46	Volume is locked through software
`paramErr`	–50	Parameter error; use `PBDTGetPath`
`rfNumErr`	–51	Reference number invalid
`extFSErr`	–58	External file system—file system identifier is nonzero
`desktopDamagedErr`	–1305	The desktop database has become corrupted—the Finder will fix this, but if your application is not running with the Finder, use `PBDTReset` or `PBDTDelete`
`afpItemNotFound`	–5012	Information not found
`afpIconTypeError`	–5030	Sizes of new icon and one it replaces don’t match