Contents

For additional assistance, contact Technical Support

Paradox overview

The topics below provide step-by-step instructions for working with the Paradox DLL.

- General
- Tables
- Records and fields
- Indexing and searching
- BLOBs
- Multi-user support

Paradox overview

The topics below provide step-by-step instructions for working with the Paradox DLL.

Paradox overview

The topics below provide step-by-step instructions for working with the Paradox DLL.

Paradox overview

The topics below provide step-by-step instructions for working with the Paradox DLL.

Paradox overview

The topics below provide step-by-step instructions for working with the Paradox DLL.

Paradox overview

The topics below provide step-by-step instructions for working with the Paradox DLL.

Paradox overview

The topics below provide step-by-step instructions for working with the Paradox DLL.

Database DLL reference

- TB30PDX.DLL
- TB30DB3.DLL

Database DLL reference

Database DLL reference

- **TB30PDX.DLL**
- **TB30DB3.DLL**

`checkDBIndex()`
`closeAllDBFiles()`
`closeDBFile()`
`closeDBIndexFile()`
`createDBFieldTag()`
`createDBFile()`
`createDBIndexFile()`
`deleteDBFile()`
`deselectDBIndexFile()`
`findDBKey()`
`firstDBKey()`
`firstDBRecord()`
`freeDBFieldTag()`
`getDBDateFormat()`
`getDBErrorString()`
`getDBFieldCount()`
`getDBFieldName()`
`getDBFieldPrecision()`
`getDBFieldType()`
`getDBFieldValue()`
`getDBFieldWidth()`
`getDBFileName()`
`getDBIndexExpression()`
`getDBIndexFileName()`
`getDBKeyType()`
`getDBKeyValue()`
`getDBNavigateToDeleted()`
`getDBRecordCount()`
`getDBRecordDeleted()`
`getDBRecordNumber()`
`gotoDBRecord()`
`lastDBKey()`
`lastDBRecord()`
`nextDBKey()`
`nextDBRecord()`
`openDBFile()`
`openDBIndexFile()`
`packDBFile()`
`previousDBKey()`
`previousDBRecord()`
`reindexDBFile()`
`removeDBRecords()`
`selectDBFile()`
`selectDBIndexFile()`
`setDBDateFormat()`
`setDBFieldTag()`
`setDBFieldValue()`
`setDBNavigateToDeleted()`
`setDBRecordDeleted()`
`writeDBRecord()`

Technical support contact information

Telephone support

Contact Asymetrix at the telephone numbers listed below for information on telephone support contracts.

Australia/Asia Pacific	(61+3) 5255471
Europe (except France and Germany), Middle East, Africa, Russia	44-923-208-433
UK	0800-716-957 (freephone)
France	05-90-83-19 (freephone)
Germany	01-30-81-27-07 (freephone)
USA and rest of world	206-637-1600

Online services

Asymetrix provides complimentary support via fax, Asymetrix BBS, CompuServe, America Online, and Internet to registered users. Technical support responds to online queries within 48 hours (Monday to Friday).

Technical support fax

- Australia/Asia Pacific (61+3) 5255-482
- Europe 44-923-208-419
- USA 206-454-0672

Asymetrix BBS

- Line 1 (1200-2400 baud/9600 baud, 206-451-1173

US Robotics HST mode)

- Line 2 (9600-14,400 baud v.32bis)   206-451-8290

America Online

- Find Asymetrix in the Industry Connection,
a subset of the Computing and Software area.

CompuServe

Windows Third Party Developer A forum, section 1	*go asymetrix* or *go winapa*
Multimedia Vendors forum, Section 15	*go multiven*
IBM Ultimedia Tools A forum, Section 5	*go ultiatools*

Internet

- techsup@asymetrix.com
- support@asymetrix.com

Initialization and finalization functions

TB30PDX.DLL

`exitPX()`	Closes the Paradox environment
`getPXMaxFiles()`	Gets the maximum number of file handles that can be used by the Engine
`getPXMaxFilesFromINI()`	Gets the maximum number of files from the WIN.INI file
`getPXMaxLocks()`	Gets the maximum number of record locks that can be used by the Engine
`getPXMaxLocksFromINI()`	Gets the maximum number of locks from the WIN.INI file
`getPXMaxTables()`	Gets the maximum number of Table handles that can be used by the Engine
`getPXMaxTablesFromINI()`	Gets the maximum number of tables from the WIN.INI file
`getPXSwapSize()`	Gets the internal swap buffer size
`getPXSwapSizeFromINI()`	Gets the swap size from the WIN.INI file
`getPXUserInfo()`	Gets user information from the WIN.INI file
`initializePX()`	Initializes the Paradox Engine for concurrency operations
`savePX()`	Saves swap buffer to disk
`setPXINIMaxFiles()`	Sets the MaxFiles entry in the [Paradox Engine] section of the WIN.INI file.
`setPXINIMaxLocks()`	Sets the MaxLocks entry in the [Paradox Engine] section of the WIN.INI file.
`setPXINIMaxTables()`	Sets the MaxTables entry in the [Paradox Engine] section of the WIN.INI file.
`setPXINISwapSize()`	Sets the SwapSize entry in the [Paradox Engine] section of the WIN.INI file.
`setPXSortOrder()`	Sets the sort order
`setPXTableCreateMode()`	Sets the mode for creating tables
`setPXTableMaxSize()`	Sets the maximum block size for new tables
`setPXUserInfo()`	Sets user information in the WIN.INI file

Table functions

TB30PDX.DLL

`addPXTable()`	Copies records from one table to another
`closePXTable()`	Closes a table
`copyPXTable()`	Copies one table family to another
`createPXTable()`	Creates a table
`decryptPXTable()`	Decrypts a table
`deletePXTable()`	Deletes a table family
`doesPXTableExist()`	Tests if a table exists
`emptyPXTable()`	Removes all records from a table. Fails if table is open or preemptive locks exist that prevent a full lock
`encryptPXTable()`	Encrypts a table. Fails if table is open or preemptive locks exist that prevent a full lock
`getPXErrorString()`	Gets the error message for an error number
`getPXFieldCount()`	Returns the number of fields in a table
`getPXKeyFieldCount()`	Returns the number of key fields in a table
`getPXMaxLocks()`	Gets the maximum number of record locks per table

getPXMaxTables()	Gets the maximum number of tables that can be open at one time
getPXRecordCount()	Gets the number of records in a table
openPXTable()	Opens a table
packPXTable()	Packs the database, and removes any space occupied by deleted records.
renamePXTable()	Changes the base name of a table family
setPXTableCreateMode()	Sets the mode for creating tables
setPXTableMaxSize()	Sets the maximum block size for new tables
upgradePXTable()	Upgrades an older Paradox table (Paradox 3.5 or later) to the latest table format

Record functions
TB30PDX.DLL

appendPXRecord()	Adds an empty record to a table
deletePXRecord()	Deletes the current record from a table
emptyPXRecord()	Clears the current record of a table
firstPXRecord()	Moves to the first record of a table
getPXRaw()	Gets raw data from the current record of a table
getPXRawDataSize()	Gets the size of the raw data in a record
getPXRecordNumber()	Gets the current record number of a table
gotoPXRecord()	Moves to a specified record of a table
insertPXRecord()	Inserts an empty record into a table
lastPXRecord()	Moves to the last record of a table
nextPXRecord()	Moves to the next record of a table
previousPXRecord()	Moves to the previous record of a table
setPXRaw()	Writes raw data to the current record of a table
updatePXRecord()	Updates the current record in a table

Field functions
TB30PDX.DLL

emptyPXField()	Empties a field of the current record of a table
getPXFieldNames()	Gets a list of field names in a table
getPXFieldType()	Gets the field type for a field
getPXFieldValue()	Gets the contents of a field in the current record of a table
setPXFieldValue()	Sets the contents of a field in the current record of a table

BLOB functions
TB30PDX.DLL

clonePXBlob()	Creates private BLOBs from public BLOBs in the current record
closePXBitmapWindow()	Closes a bitmap window
closePXBlob()	Closes a BLOB
dropPXBlob()	Drops (releases) a BLOB from the current record of a table
freePXBlobMemory()	Frees the global memory referenced by a BLOB handle
freePXGraphicBlob()	Frees the GDI memory referenced by a bitmap handle
freePXGraphicBlobPalette()	Frees the GDI memory referenced by the Windows color palette

`getPXBitmapSize()`	Gets the size of a bitmap
`getPXBlob()`	Reads data from a BLOB
`getPXBlobQuick()`	Reads BLOB leader directly from the current record of a table
`getPXBlobSize()`	Gets the size of a BLOB
`getPXFileSize()`	Gets the size of a file for BLOB operations
`getPXGraphicBlob()`	Returns a handle to a Windows bitmap
`getPXGraphicBlobPalette()`	Returns a handle to the standard Windows color palette
`getPXMemoBlob()`	Returns a memo BLOB as a ToolBook string
`openPXBitmapWindow()`	Opens a child window for displaying a bitmap
`openPXBlobRead()`	Opens a BLOB for reading operations
`openPXBlobWrite()`	Opens a BLOB for writing operations
`setPXBitmapWindowInfo()`	Sets display attributes for bitmap window
`setPXBlob()`	Writes data to a BLOB
`setPXBlobFromFile()`	Converts a file into BLOB format
`setPXGraphicBlob()`	Copies a handle to a Windows bitmap into a graphic BLOB
`setPXGraphicBlobFromFile()`	Converts a bitmap file into BLOB format
`setPXMemoBlob()`	Copies a string into a memo BLOB
`writePXBlobToFile()`	Writes a BLOB to a file
`writePXGraphicBlobToFile()`	Writes a graphic BLOB as a bitmap file

Key and index functions

TB30PDX.DLL

`addPXKey()`	Creates a primary or secondary index on a table
`dropPXKey()`	Deletes a primary or secondary index
`mapPXKey()`	Obtains a field handle for a composite or case-insensitive, single-field index
`queryPXKey()`	Gets information about an index
`searchPXField()`	Searches a table on a specified field
`searchPXKey()`	Searches a table for a key match

Search functions

TB30PDX.DLL

`searchPXField()`	Searches a table on a specified field
`searchPXKey()`	Searches a table for a key match

Password and security functions

TB30PDX.DLL

`addPXPassword()`	Enters a password
`decryptPXTable()`	Decrypts a table
`deletePXPassword()`	Removes a password
`encryptPXTable()`	Encrypts a table. Fails if table is open or preemptive locks exist that prevent a full lock
`isPXTableProtected()`	Tests if a table is encrypted

Sharing and concurrency functions

TB30PDX.DLL

`getPXNetErrorUser()`	Reports the name of the user causing a locking error

`getPXNetUserName()`	Obtains the name of a user
`gotoPXNetRecordLock()`	Returns to a previously locked record
`isPXNetRecordLocked()`	Determines if the current record has been locked
`isPXNetTableChanged()`	Tests if a table has been changed
`lockPXNetFile()`	Locks a file
`lockPXNetRecord()`	Locks the current record of a table
`lockPXNetTable()`	Locks a table
`refreshPXNetTable()`	Resynchronizes a shared table
`unlockPXNetFile()`	Unlocks a file
`unlockPXNetRecord()`	Unlocks the current record of a table
`unlockPXNetTable()`	Unlocks a table

Error messages

TB30PDX.DLL

1	Drive not ready	99	Table name is invalid
2	Directory not found	101	End of table
3	File is busy	102	Start of table
4	File is locked	103	No more record handles available
5	Could not find file	104	Invalid record handle
6	Table is corrupted	105	Operation on empty table
7	Primary index is corrupted	106	Invalid lock code
8	Primary index is out of date	107	Engine not initialized with PXNetInit
9	Record is locked	108	Invalid file name
10	Sharing violation	109	Invalid unlock
11	Sharing violation	110	Invalid lock handle
12	No access to directory	111	Too many locks on table
13	Sort for index different from table	112	Invalid sort order table
14	Single user but directory is shared	113	Invalid net type
15	Multiple Paradox net files found	114	Invalid directory name
16	Directory is in use by Paradox 3.5	115	Too many passwords specified
21	Insufficient password rights	116	Invalid password
22	Table is write-protected	117	Buffer is too small for result
30	Data type mismatch	118	Table is busy
31	Argument is out of range	119	Table is locked
33	Invalid argument	120	Table was not found
40	Not enough memory to complete operation	121	Secondary index was not found
41	Not enough disk space to complete operation	122	Secondary index is corrupted
50	Another user deleted record	124	Disk is write-protected
51	Operations N/A for BLOB open mode	125	Record too big
52	BLOB already open	126	General system error
53	Invalid offset into BLOB	127	Not enough stack space to complete operation
54	Invalid size for BLOB	128	Table is full
55	Another user modified BLOB	129	Not enough swap buffer space to complete operation
56	BLOB file corrupted	130	Table is SQL replica
57	Cannot index on a BLOB	131	Too many clients for the engine DLL
59	Invalid BLOB handle	132	Exceeds limits specified in WIN.INI
60	Cannot search on a BLOB field	133	No more slots for file handle remapping
70	No more file handles available	134	Can't share Paradox net file -- is SHARE.EXE loaded?
72	No more table handles available	135	Can't run engine in Windows real mode
73	Invalid date given	136	Can't modify table opened on non-maintained seconda
74	Invalid field name	137	Timed out trying to achieve a lock
75	Invalid field handle	251	Unable to lock or unlock memory
76	Invalid table handle	252	Undefined error code
78	Engine not initialized	253	Too many items in list parameter

79	Previous fatal error, cannot proceed
81	Table structures are different
82	Engine already initialized
83	Unable to perform operation on open table
86	No more temporary names available
89	Record was not found
92	Table cannot be upgraded
93	Feature not available for tables older than Paradox 4.0
94	Table is indexed
95	Table is not indexed
96	Secondary index is out of date
97	Key violation
98	Could not login on network
254	Missing items in list parameter
255	Internal error no:
256	File I/O error
257	Invalid graphic BLOB format
258	Windows GDI failure
259	Invalid window handle
260	Invalid bitmap handle
261	BLOB window registration failed
262	BLOB window creation failed
263	Invalid BLOB window color
264	Invalid BLOB window display mode
265	Invalid BLOB window position/size
266	Unable to re-open table after dropPXKey
267	Invalid Entry in [Paradox Engine] section of WIN.INI

addPXKey( )

TB30PDX.DLL

Engine Function **PXKeyAdd()**

Description Creates a primary or secondary index on a table

Syntax `addPXKey(<table alias>,<field names>,<mode>)`

Declaration `INT addPXKey(STRING,STRING,INT)`

Parameters `<table alias>` Alias for table to add key on. Table must be open.

`<field names>` Comma-separated list of names of fields used to create key

`<mode>` Mode for creating index

0	Primary index (key)
1	Secondary index (maintained only when open)
2	Incremental Secondary index (maintained even if closed)

Returns

0	Successful operation
< 0	Error number

Example

```
--Add an index to table and open table on new index
fieldNumber = mapPXKey("myDatabase", "Last Name, First Name", "Last name
First", 1)
get addPXKey("myDatabase", "Last name First", 2)
get closePXTable("myDatabase")                    -- close on old index
get openPXTable("myDatabase", "c:\data\mydata.db", fieldNumber,savemode)
```

addPXPassword( )

TB30PDX.DLL

Engine Function	**PXPswAdd()**
Description	Enters a password
Syntax	`addPXPassword(<password>)`
Declaration	`INT addPXPassword(STRING)`
Parameters	`<password>` Password to use
Returns	0 Successful operation < 0 Error number

Example

```
--enter password to encrypt table
get addPXPassword(text of field "Password")
if it >= 0
  get openPXTable("myDatabase", "C:\data\mydb", 0, 0)
else
  request "Incorrect password"
end
```

addPXTable( )

TB30PDX.DLL

Engine Function	**PXTblAdd()**
Description	Adds records from one table to another
Syntax	`addPXTable(<source table>,<destination table>)`
Declaration	`INT addPXTable(STRING,STRING)`

Parameters	`<source table>`	Filename of source table providing records to add
	`<destination table>`	Filename of destination table

Returns	0	Successful operation
	< 0	Error number

Example

```
--Merge the table specified in field "database" to the current database
get addPXTable("c:\data\newData.db", "c:\data\currData.db")
if it = 0
  request "merge successful"
else
  request getPXErrorString(it)
end
```

appendPXRecord( )

TB30PDX.DLL

Engine Function	**PXRecAppend()**
Description	Adds a record to a table
Syntax	`appendPXRecord(<table alias>)`
Declaration	`INT appendPXRecord(STRING)`
Parameters	`<table alias>` Alias of table to append record to
Returns	0 Successful operation < 0 Error number

Example

```
--Add a new record to the database with the contents of the recordBuffer
if name of target = "add"
   get appendPXRecord("myDatabase")
end
```

clonePXBlob( )

TB30PDX.DLL

Engine Function	**PXBlobClone()**
Description	Creates private BLOBs from public BLOBs in the current record
Syntax	`clonePXBlob(<table alias>,<field name>)`
Declaration	`INT clonePXBlob(STRING,STRING)`

Parameters	`<table alias>`	Alias of table to use
	`<field name>`	Name of field from which to create BLOB
	`NULL`	Create private BLOBs from all BLOBs in record

Returns	0	Successful operation
	< 0	Error number

Example

```
--Get a private copy of the BLOB to modify
--other users can view the public BLOB.

lockHandle = lockPXNetRecord("myDatabase")
if lockHandle >= 0
  hBlobClone = clonePXBlob("myDatabase", "graphic")
  if it < 0
    request getPXErrorString(it)
    break
  end
  get unlockPXNetRecord("myDatabase", lockHandle)
else
  request getPXErrorString(lockHandle)
end
```

closePXBitmapWindow( )

TB30PDX.DLL

Engine Function	**None**
Description	Closes a bitmap window and destroys the associated bitmap
Syntax	`closePXBitmapWindow(<window handle>)`
Declaration	`INT closePXBitmapWindow(WORD)`
Parameters	`<window handle>` Window handle returned by `openPXBitmapWindow()`
Returns	0 Successful operation < 0 Error number

Example

```
to handle closeBitmap
   system hBMP,hPal,hWndBMP
-- When we're finished with the bitmap and the window:
-- close the window
   get closePXBitmapWindow(hWndBMP)
-- free the Windows handle to the bitmap
   get freePXGraphicBlob(hBMP)
-- ditto for the palette
   get freePXGraphicBlobPalette(hPal)
end closeBitmap
```

closePXBlob( )

TB30PDX.DLL

Engine Function	**PXBlobClose()**	
Description	Closes a BLOB	
Syntax	`closePXBlob(<BLOB handle>,<accept>)`	
Declaration	`INT closePXBlob(INT,INT)`	
Parameters	`<BLOB handle>`	Handle to BLOB to close
	`<accept>`	Accept or reject changes 0 Reject 1 Accept
Returns	0	Successful operation
	< 0	Error number

Example

```
--Close the private BLOB and post the changes to the database
lockHandle = lockPXNetRecord("myDatabase")
if lockHandle >= 0
  get closePXBlob(hModifiedBlob, 1)
  get unlockPXNetRecord("myDatabase", lockHandle)
else
  request getPXErrorString(lockHandle)
end
```

closePXTable( )

TB30PDX.DLL

Engine Function	**PXTblClose()**	
Description	Closes a table	
Syntax	`closePXTable(<table alias>)`	
Declaration	`INT closePXTable(STRING)`	
Parameters	`<table alias>`	Alias of table to close
Returns	0	Successful operation
	< 0	Error number

Example

```
get closePXTable("myTable")
if it < 0
  request getPXErrorString(it)
end if
```

copyPXTable( )

TB30PDX.DLL

Engine Function	**PXTblCopy()**	
Description	Copies one table family to another	
Syntax	`copyPXTable(<source table>,<destination table>)`	
Declaration	`INT copyPXTable(STRING,STRING)`	
Parameters	`<source table>`	Base name of source table family
	`<destination table>`	Base name of destination table family
Returns	0	Successful operation
	< 0	Error number

Example

```
--Allow the user to install a local copy of the database
ask "Enter path for local database"
if it <> null
  get copyPXTable("d:\data\netDB", it)
  if it < 0
    request getPXErrorString(it)
  end
end
```

createPXTable( )

TB30PDX.DLL

Engine Function	**PXTblCreate()**
Description	Creates a table
Syntax	`createPXTable(<table name>,<field names>,<field types>)`
Declaration	`INT createPXTable(STRING,STRING,STRING)`
Parameters	`<table name>` Name of table (Engine assumes .DB extension.)
	`<field names>` List of field names
	`<field types>` List of field types
Returns	0 Successful operation
	< 0 Error number

Example

```
--Create a database of CD names, artists, songs, and album covers
get createPXTable("c:\data\CDbase", "Title, artist, songs, cover", \
   "A25, A25, M40, G10")
```

decryptPXTable( )
TB30PDX.DLL

Engine Function	**PXTblDecrypt()**
Description	Decrypts a table
Syntax	`decryptPXTable(<table file name>)`
Declaration	`INT decryptPXTable(STRING)`
Parameters	`<table file name>` Name of the table file to decrypt
Returns	0 Successful operation < 0 Error number

Example

```
--Remove password protection from file
get isPXTableProtected("c:\data\mydb")
if it = 1
  get addPXPassword("rosebud")        -- sent table password to engine
  get decryptPXPassword("c:\data\mydb")
end
```

deletePXPassword( )

TB30PDX.DLL

Engine Function	**PXPswDel()**
Description	Removes a password
Syntax	`deletePXPassword(<password>)`
Declaration	`INT deletePXPassword(STRING)`
Parameters	`<password>` Password to delete
Returns	0 Successful operation < 0 Error number

Example

```
--Remove password from system list
get deletePXPassword("rosebud")
if it < 0
  request "Invalid password"
end
```

deletePXRecord( )

TB30PDX.DLL

Engine Function	**PXRecDelete()**
Description	Deletes the current record from a table
Syntax	`deletePXRecord(<table alias>)`
Declaration	`INT deletePXRecord(STRING)`
Parameters	`<table alias>` Alias of table from which to delete current record
Returns	0 Successful operation < 0 Error number

Example

```
--Ask for confirmation to delete the current record
request "Do you really want to delete the current record?" with "OK" or
"Cancel"
if it = "OK"
  get deletePXRecord("myDatabase")
end
```

deletePXTable( )

TB30PDX.DLL

Engine Function	**PXTblDelete()**
Description	Deletes a table family
Syntax	`deletePXTable(<table file name>)`
Declaration	`INT deletePXTable(STRING)`
Parameters	`<table file name>` File name of table family to delete*
	*The engine does not check whether the file is a valid Paradox file before it deletes it.
Returns	0 Successful operation
	< 0 Error number

Example

```
--Delete database
request "do you really want to delete the whole database?" with "OK" or\
   "Cancel"
if it = OK
  get deletePXTable("c:\data\CDBase")
end
```

doesPXTableExist( )

TB30PDX.DLL

Engine Function	**PXTblExist()**
Description	Tests if a table exists
Syntax	`doesPXTableExist(<table name>)`
Declaration	`INT doesPXTableExist(STRING)`
Parameters	`<table name>` File name of table to look for

Returns

1	Table exists
0	Table does not exist
< 0	Error number

Example

```
--Check to see if a table exists before copying to the name
get doesPXTableExist("c:\data\myData")
if it = 1
  request "Table already exists"
else
  copyPXTable("n:\data\shared", "c:\data\myData")
end
```

dropPXBlob( )

TB30PDX.DLL

Engine Function	**PXBlobDrop()**	
Description	Drops (releases) a BLOB from the current record of a table	
Syntax	`dropPXBlob(<table alias>,<field name>)`	
Declaration	`INT dropPXBlob(STRING,STRING)`	
Parameters	`<table alias>`	Alias of table to use
	`<field name>`	Name of BLOB field to release
Returns	0	Successful operation
	< 0	Error number

Example

```
--Delete the memo BLOB "summaryText" from the current record
get dropPXBlob("myDatabase", "summaryText")
if it < 0
  request PXErrorString(it)
end
```

dropPXKey( )

TB30PDX.DLL

Engine Function	**PXKeyDrop()**
Description	Deletes a primary or secondary index
Syntax	`dropPXKey(<table alias>,<field name>,<index ID>)`
Declaration	`INT dropPXKey(STRING,STRING,WORD)`

Parameters

`<table alias>`	Alias of table to use
`<field name>`	Field name of key for case-sensitive, single-field, secondary index only otherwise leave null
`<index ID>`	Handle of index to delete (if field name is not provided)

If you delete the primary index, all secondary indexes are deleted also.

Returns

1	Deleted primary index, re-opened the table unindexed
2	Deleted index the table was open on; re-opened the table on the primary index
< 0	Error number

emptyPXField( )

TB30PDX.DLL

Engine Function	**None**
Description	Empties a field of the current record of a table. Changes do not take effect in the table until `updatePXRecord()` is used.
Syntax	`emptyPXField(<table alias>,<field name>)`
Declaration	`INT emptyPXField(STRING,STRING)`
Parameters	`<table alias>` Alias of table to use
	`<field name>` Name of field to set to blank
Returns	0 Successful operation
	< 0 Error number

Example

```
--Clear the data from field "date" for all records in the database
get firstPXRecord("myDatabase")
get emptyPXfield("myDatabase", "date")
while nextPXRecord("myDatabase") >= 0
  get emptyPXField("myDatabase", "date")
end
```

emptyPXRecord( )

TB30PDX.DLL

Engine Function	**PXRecBufEmpty()**
Description	Clears the buffer for current record of a table. The record in the table is changed when `updatePXRecord()`, `insertPXRecord()`, or `appendPXRecord()` is used.
Syntax	`emptyPXRecord(<table alias>)`
Declaration	`INT emptyPXRecord(STRING)`
Parameters	`<table alias>` Alias of table that contains the record
Returns	0 Successful operation < 0 Error number

Example

```
--Create an empty record in the database
get emptyPXRecord("myDatabase")
get appendPXRecord("myDatabase")
```

emptyPXTable( )

TB30PDX.DLL

Engine Function	**PXTblEmpty()**
Description	Removes all records from a table. Fails if table is open or preemptive locks exist that prevent a full lock.
Syntax	`emptyPXTable(<table file name>)`
Declaration	`INT emptyPXTable(STRING)`
Parameters	`<table file name>` File name of table from which to remove records
Returns	0 Successful operation < 0 Error number

Example

```
--Start over again on this table
get emptyPXTable("c:\data\myData")
if it = 0
   request "ready to start again"
else
   request getPXErrorString(it)
end
```

encryptPXTable( )

TB30PDX.DLL

Engine Function	**PXTblEncrypt()**
Description	Encrypts a table. Fails if table is open or preemptive locks exist that prevent a full lock.
Syntax	`encryptPXTable(<table file name>,<password>)`
Declaration	`INT encryptPXTable(STRING,STRING)`

Parameters

`<table file name>`	File name of table to encrypt
`<password>`	Password to use

Returns

0	Successful operation
< 0	Error number

Example

```
-- Set master password for file
get encryptPXTable("c:\data\mydb", "rosebud")
-- pass password to table, then open table
if it >= 0
  get addPXPassword("rosebud")
  get openPXTable("myDatabase", "c:\data\mydb", 0,0)
else
  request getPXErrorString(it)
end
```

exitPX( )

TB30PDX.DLL

Engine Function	**PXExit()**
Description	Closes the Paradox environment
Syntax	`exitPX()`
Declaration	`INT exitPX()`
Parameters	None

Returns

0	Successful operation
< 0	Error number

Example

```
--Close the table and exit the Paradox environment
get closePXTable("myDatabase")
get exitPX()
```

▶

firstPXRecord( )

TB30PDX.DLL

Engine Function	**PXRecFirst()**
Description	Moves to the first record of a table
Syntax	`firstPXRecord(<table alias>)`
Declaration	`INT firstPXRecord(STRING)`
Parameters	`<table alias>` Alias of table to use
Returns	0 Successful operation < 0 Error number

Example

```
--Navigate to the first record in the database and compare against \
--text in field "lastName"
get firstPXRecord("myDatabase")
if getPXFieldValue("myDatabase", "lastName") = text of field "lastName"
  request "matches the first record"
else
  request "values do not match"
end
```

freePXBlobMemory( )

TB30PDX.DLL

Engine Function	None
Description	Frees the global memory referenced by a BLOB handle
Syntax	`freePXBlobMemory(<handle>)`
Declaration	`INT freePXBlobMemory(WORD)`
Parameters	`<handle>` Handle to Windows global memory returned from `getPXBlob()`
Returns	0 Successful operation < 0 Error number

Example

```
--Get 4 bytes of BLOB at offset 1024 and free it if the value is not correct
hBinaryData = getPXBlob(hBlob, 4, 1024)
pBinaryData = globalLock(hBinaryData) -- globalLock is linked from KERNEL
if pointerDWORD(0, pBinaryData) <> 97825378  -- example literal value
  get globalUnlock(hBinaryData)  -- globalUnlock is linked from KERNEL
  get freePXBlobMemory(hBinaryData)
else ...
```

freePXGraphicBlob( )

TB30PDX.DLL

Engine Function	None
Description	Frees the GDI memory referenced by a bitmap handle
Syntax	`freePXGraphicBlob(<hBitmap>)`
Declaration	`INT freePXGraphicBlob(WORD)`
Parameters	`<hBitmap>` Bitmap handle returned by `getPXGraphicBlob()`
Returns	0 Successful operation < 0 Error number

Example

```
--Release GDI system resources used by bitmap handle after the \
--window closes
get closePXBitmapWindow(hWND)    --hWND is handle returned from
                                 --openPXBitmapWindow()
get freePXGraphicBlob(hBitmap)   --hBitmap is handle returned from
                                 --getPXGraphicBlob()
if it < 0
  request getPXErrorString(it)
end
```

freePXGraphicBlobPalette( )
TB30PDX.DLL

Engine Function	None
Description	Frees the GDI memory referenced by the Windows color palette
Syntax	`freePXGraphicBlobPalette(<hPalette>)`
Declaration	`INT freePXGraphicBlob(WORD)`
Parameters	`<hPalette>` Handle returned by `getPXGraphicBlobPalette()`
Returns	0 Successful operation < 0 Error number

Example

```
--Free GDI system resourses used by the palette
--hPal is the palette handle returned from the \
--getPXGraphicBlobPalette() function
get freePXGraphicBlobPalette(hPal)
if it < 0
  request "harmless internal error"
end
```

getPXBitmapSize( )

TB30PDX.DLL

Engine Function	**None**
Description	Gets the size of a bitmap for graphic BLOB operations
Syntax	`getPXBitmapSize(<bitmap handle>)`
Declaration	`LONG getPXBitmapSize(INT)`
Parameters	`<bitmap handle>` Handle to bitmap
Returns	>= 0 Size of graphic BLOB in bytes 0 Error

Example

```
BMPSize = getPXBitmapSize(hBitmap)
hPXBMP = openPXBlobWrite("myTable","Picture",BMPSize,0)
if hPXBMP < 0
   request getPXErrorString(it)
   -- clean up Windows stuff
   break
end if
get setPXGraphicBlob(hPXBMP,hBitmap)
```

getPXBlob( )

TB30PDX.DLL

Engine Function	**PXBlobGet()**
Description	Reads data from a BLOB
Syntax	`getPXBlob(<BLOB handle>,<size>,<offset>)`
Declaration	`LONG getPXBlob(INT,WORD,LONG)`

Parameters

`<BLOB handle>`	Handle to BLOB from which to read data
`<size>`	Size of BLOB in bytes (1 - 256mb)
`<offset>`	Offset from start of BLOB to begin reading

Returns

>= 0	Handle to Windows global memory
< 0	Error number

Example

```
--Read in WMF stored as BLOB and call user-defined \
--"put it on the Clipboard" function
hBlob = openPXBlobRead("myDatabase", "metafile")
if it >= 0
  hMeta = getPXBlob(hBlob, getPXBlobSize(hBlob), 0)
  if hMeta > 0
    get placeOnClipboard(hMeta)
  end
  get closePXBlob(hBlob, 0)
end
```

getPXBlobQuick( )

TB30PDX.DLL

Engine Function	**PXBlobQuickGet()**
Description	Reads BLOB leader directly from the current record of a table
Syntax	`getPXBlobQuick(<table alias>,<field name>,<size>)`
Declaration	`LONG getPXBlobQuick(STRING,STRING,INT)`

Parameters

`<table alias>`	Alias of table to use
`<field name>`	Name of BLOB field
`<size>`	Size of leader (0 - 240)

Returns

>= 0	Handle to private BLOB
< 0	Error number

Example

```
--Get the original file name for BLOB from beginning of BLOB and pass \
--it to function to write it to the same file again.

hFileName = getPXBlobQuick("myDatabase", "binaryData", 13)
pFileName = globalLock(hFileName) -- globalLock is linked from KERNEL
get saveBlob(pointerString(0, pFileName)) -- user defined function
get globalUnlock(hFileName) -- globalLock is linked from KERNEL
get freePXBlobMemory(hFileName)
```

getPXBlobSize( )

TB30PDX.DLL

Engine Function	**PXBlobGetSize()**	
Description	Gets the size of a BLOB	
Syntax	`getPXBlobSize(<BLOB handle>)`	
Declaration	`LONG getPXBlobSize(INT)`	
Parameters	`<BLOB handle>`	Handle to BLOB
Returns	>= 0	Size of BLOB
	< 0	Error number

Example

```
hBlob = openPXBlobRead("Bintest", "Binfield")
set bSize to getPXBlobSize(hBlob)
request bSize
hB = getPXBlob(hBlob,bSize, 0)
get closePXBlob(hBlob, 0)
```

getPXErrorString( )

TB30PDX.DLL

Engine Function	**None**
Description	Gets the error message for an error number
Syntax	`getPXErrorString(<error number>)`
Declaration	`STRING getPXErrorString(INT)`
Parameters	`<error number>` Error number returned from other DLL functions
Returns	Error message for given error number If an error occurs, NULL is returned and `sysError` is set to the error number

Example

```
--If an error occurs, request the error message for the user to see
get openPXTable("myDatabase", "C:\DATA\CUSTOMER.DB", 0, 0)
if it < 0
  request getPXErrorString(it)
end
```

getPXFieldCount( )

TB30PDX.DLL

Engine Function	**PXRecNFlds()**	
Description	Returns the number of fields in a table	
Syntax	`getPXFieldCount(<table alias>)`	
Declaration	`INT getPXFieldCount(STRING)`	
Parameters	`<table alias>`	Alias of table
Returns	>= 0	Number of fields in table
	< 0	Error number

Example

```
--Compare the number of fields in the database
--with the number of fields on the pages

dbFields = getPXFieldCount("myDatabase")
objList = objects of this page
step i from 1 to itemCount("objList")
  pop objList
  if object of it is "field"
    decrement dbFields
  end
end
if dbFields = 0
  request "correct number of fields"
end
```

getPXFieldNames( )

TB30PDX.DLL

Engine Function	**PXFldName()**
Description	Gets a list of field names in a table
Syntax	`getPXFieldNames(<table alias>)`
Declaration	`STRING getPXFieldNames(STRING)`
Parameters	`<table alias>` Alias of table
Returns	Comma-separated list of field names If an error occurs, NULL is returned and `sysError` is set to the error number.

Example

```
fieldNames = getPXFieldNames("myDatabase")
step i from 1 to textlineCount(fieldNames)
  put TAB&getPXFieldType("myDatabase", textline i of fieldNames)\
  after textline i of fieldNames
end
request "The record format is"&CRLF&fieldNames
```

getPXFieldType( )

TB30PDX.DLL

Engine Function	**PXFldType()**
Description	Gets the type for a field (as well as the length for an alphanumeric field and the width of a BLOB field)
Syntax	`getPXFieldType(<table alias>,<field name>)`
Declaration	`STRING getPXFieldType(STRING,STRING)`
Parameters	`<table alias>` Alias of table to use
	`<field name>` Name of field of which to get type
Returns	Field type using these codes:

Code	Meaning	Length	Notes
A*nnn*	Alphanumeric	*nnn* bytes	<= 255, no ASCII zero (embedded null)
D	Date	4 (Julian format, start date = Jan 1, 1 AD)	
N	Numeric	8	+/- 10^-307 to +/- 10^308 (15 significant digits)
S	Short numeric	2	+/- 32,767
$	Numeric, currency	8	+/- 10^-307 to +/- 10^308 (15 significant digits)
M	Unformatted memo BLOB	see BLOB topic	
B	Unformatted binary BLOB	see BLOB topic	
G	Graphics BLOB	see BLOB topic	

If an error occurs, null is returned and `sysError` is set to the error number.

Example

```
--Check whether the field is a BLOB.  If it is a BLOB \
--then call routine to get its value.
pxFieldType = getPXFieldType("myDatabase", fieldName)
if pxFieldType is in "M B F O G"
  --call user defined handler to get BLOB value
  get mygetBlobValue("myDatabase", fieldName, pxFieldType)
end
```

getPXFieldValue( )

TB30PDX.DLL

Engine Functions:	**PXGetAlpha(), PXGetDate(), PXGetDouble(), PXGetLong(), PXGetShort()**	
Description	Gets the contents of a field in the current record of a table	
Syntax	`getPXFieldValue(<table alias>,<field name>)`	
Declaration	`STRING getPXFieldValue(STRING,STRING)`	
Parameters	`<table alias>`	Alias of table to use
	`<field name>`	Name of field of which to get contents
Returns	Contents of given field from the current record of the given table.	
	If an error occurs, NULL is returned and `sysError` is set to the error number.	

Example

```
--Get the name, age, and birthDate values from the current record and put
them
--into fields on the page
set text of field "name" to getPXFieldValue("myDatabase", "name")
set text of field "age" to  getPXFieldValue("myDatabase", "age")
set text of field "birthdate" to getPXFieldValue("myDatabase", "birth date")
```

getPXFileSize( )

TB30PDX.DLL

Engine Function	**None**
Description	Gets the size of a file for BLOB operations.
Syntax	`getPXFileSize(<file name>)`
Declaration	`LONG getPXFileSize(STRING)`
Parameters	`<file name>` Name of file
Returns	>= 0 Size of file in bytes
	0 Error

Example

```
to handle buttonClick
-- find out how large the graphic BLOB will be
   request getPXFileSize("arrow1.bmp") + 8
end
```

getPXGraphicBlob( )
TB30PDX.DLL

Engine Function	None
Description	Returns a handle to a Windows bitmap
Syntax	`getPXGraphicBlob(<BLOB handle>)`
Declaration	`LONG getPXGraphicBlob(INT)`
Parameters	`<BLOB handle>` Handle returned by `openPXBlobRead()` or `openPXBlobWrite()`
Returns	HBITMAP - handle to a Windows device dependent bitmap suitable to be selected into a Device Context or put on the Clipboard.
Comments	It is the caller's responsibility to free the bitmap handle. This is done by calling `freePXGraphicBlob()`, which will free the GDI memory taken by the BLOB. Handles placed on the Clipboard are managed by the Clipboard, and freeing them will cause errors.

Example

```
--Get handle to bitmapBlob and show it in a window
hBlob = openPXBlobRead("myDatabase", "BlobField")
if hBlob >= 0
  hBitmap = getPXGraphicBlob(hBlob)
  hWnd = openPXBitmapWindow(hBitmap, null, clientHandle of mainWindow, \
  "0,0", 0, "")
end
```

getPXGraphicBlobPalette( )

TB30PDX.DLL

Engine Function	None
Description	Returns a handle to the standard Windows color palette
Syntax	`getPXGraphicBlobPalette(<BLOB handle>)`
Declaration	`LONG getPXGraphicBlobPalette(INT)`
Parameters	`<BLOB handle>` Handle returned by `openPXBlobRead()` or `openPXBlobWrite()`
Returns	Handle to the Windows palette for the graphic BLOB.
Comments	The user is responsible for freeing the handle using `freePXGraphicBlobPalette()`. Handles placed on the Clipboard are managed by the Clipboard and freeing them will cause errors.

Example

```
--Get handle to bitmap and handle to palette to put on Clipboard
hBlob = openPXBlobRead("myDatabase", "Graphic")
if hBlob >= 0
  hBitmap = getPXGraphicBlob(hBlob)
  hPalette = getPXGraphicBlobPalette(hBlob)
end
```

getPXKeyFieldCount( )

TB30PDX.DLL

Engine Function	**PXKeyNFlds()**
Description	Returns the number of key fields in a table
Syntax	`getPXKeyFieldCount(<table alias>)`
Declaration	`INT getPXKeyFieldCount(STRING)`
Parameters	`<table alias>` Aliasof table to use
Returns	>= 0 Number of fields in primary key
	< 0 Error number

Example

```
--Request the key fields in this database
get getPXKeyFieldCount("myDatabase")
```

getPXMaxFiles( )

TB30PDX.DLL

Engine Function	**PXGetDefaults()**
Description	Gets the value the Engine is currently using for the maximum number of files that can be used
Syntax	`getPXMaxFiles()`
Declaration	`INT getPXMaxFiles()`
Parameters	None
Returns	>= 0 Maximum file handles < 0 Error number

Example

```
--Make sure there are enough file handles for the tables and indexes
--The number needed is stored as a user defined property of the book
if getPxMaxFiles() < numHandlesNeeded of this book
   get setPXINIMaxFiles(numHandlesNeeded of this book)
   if it < 0
      request "Unable to allocate sufficient file handles for this
database"&CRLF\
      "Please increase FILES = in config.sys"
   end
end
```

getPXMaxLocks( )

TB30PDX.DLL

Engine Function	**PXGetDefaults()**
Description	Gets the value the Engine is currently using for the maximum number of locks that can be used
Syntax	`getPXMaxLocks()`
Declaration	`INT getPXMaxLocks()`
Parameters	None
Returns	>= 0 Maximum locks < 0 Error number

Example

```
--get the current INI setting
get getPXMaxLocksFromINI()

--If the current ini value is less than what I need then try to reset it.
if it < myExpectedValue
  get setPXINIMaxLocks(myExpectedValue)
  get initializePX(myTable)   --somebody else has initialized the engine.
  get getPXMaxLocks()
  if it < myExpectedValue
  --somebody else has linked the engine and I cannot set the maxlocks to
what I
  --need so I will give error and exit
    request "Unable to set number of locks.  Please Exit all other
Paradox"&CRLF&"Engine applications and try again."
    send exit
  end
end
```

getPXMaxTables( )

TB30PDX.DLL

Engine Function	**PXGetDefaults()**
Description	Gets the value the Engine is currently using for the maximum number of tables that can be used
Syntax	`getPXMaxTables()`
Declaration	`INT getPXMaxTables()`
Parameters	None

Returns

>= 0	Maximum tables usable by engine
< 0	Error number

Example

```
--Check whether all of the tables in this database can be opened at once
--The number of tables in this database is stored as a user defined property
of the book
if getPXMaxTables() < numTables of this book
   get setPXINIMaxTables(numTables of this book)
   if it < 0
      request "Unable to open all of the tables in this database"
   end
end
```

getPXMemoBlob( )

TB30PDX.DLL

Engine Function	None
Description	Returns a memo BLOB as a ToolBook string
Syntax	`getPXMemoBlob(<BLOB handle>)`
Declaration	`STRING getPXMemoBlob(INT)`
Parameters	`<BLOB handle>` Handle returned by `openPXBlobRead()` or `openPXBlobWrite()`
Returns	A string if successful If error, returns NULL and sets `sysError` to the error value

Example

```
--Get text of memo and put it into a field
hBlob = openPXBlobRead("myDatabase", "reviewText")
text of field "review" = getPXMemoBlob(hBlob)
get closePXBlob(hBlob, 0)
```

Comments Will return the whole BLOB if it is <= 64K bytes and return an error if the BLOB is > 64K bytes. Memo BLOBs > 64K bytes should be accessed by the `getPXBlob()` function and the data taken out in chunks from the global memory buffer.

getPXNetErrorUser( )

TB30PDX.DLL

Engine Function	**PXNetErrorUser()**
Description	Reports the name of the user causing a locking error
Syntax	`getPXNetErrorUser()`
Declaration	`STRING getPXNetErrorUser()`
Parameters	None
Returns	Name of network user causing locking error
	If an error occurs, null is returned and `sysError` is set to the error number

Example

```
--Request the name of the user who has the record locked if a locking error
--occurs
get lockPXRecord("myDatabase")
if it = -9 -- record locked
  request getPXNetErrorUser() && "has the record locked."
end
```

getPXNetUserName( )

TB30PDX.DLL

Engine Function	**PXNetUserName()**
Description	Obtains the name of the user
Syntax	`getPXNetUserName()`
Declaration	`STRING getPXNetUserName()`
Parameters	None
Returns	Name of network user
	If an error occurs, null is returned and `sysError` is set to the error number

Example

```
--Get the user name for this user to personalize the greeting
get getPXNetUserName()
if it <> null
  request "Hello" && it & ", welcome to my database."
end
```

getPXRaw( )

TB30PDX.DLL

Engine Function	**PXRawGet()**	
Description	Gets raw data from the current record of a table	
Syntax	`getPXRaw(<table alias>,<size>)`	
Declaration	`LONG getPXRaw(STRING,INT)`	
Parameters	`<table alias>`	Alias of table to use
	`<size>`	Size of the record structure in bytes. Use `getPXRawDataSize()` to get the size.
Returns	>= 0	handle to raw data
	< 0	Error number

Examples

```
--Save the contents of the current record for undo function
--sizeofData is calculated previously
hRawData = getPXRaw("myDatabase", sizeofData)
request "You may now change the contents of the record."
```

getPXRawDataSize( )

TB30PDX.DLL

Engine Function	**None**
Description	Gets the raw data size for the current record of a table
Syntax	`getPXRawDataSize(<table alias>)`
Declaration	`INT getPXRawDataSize(STRING)`
Parameters	`<table alias>` Alias of table to use
Returns	>= 0 Size of raw data < 0 Error number

Example

```
rSize = getPXRawDataSize("myTable")
```

getPXRecordCount( )

TB30PDX.DLL

Engine Function	**PXTblNRecs()**
Description	Gets the number of records in a table
Syntax	`getPXRecordCount(<table alias>)`
Declaration	`LONG getPXRecordCount(STRING)`
Parameters	`<table alias>` Alias of table to use
Returns	>= 0 Number of records in given table < 0 Error number

Example

```
--Get the number of records in the database and step through them \
--setting field "lastName" to 3
get firstPXRecord("myDatabase")
step i from 1 to getPXRecordCount("myDatabase")
  get setPXFieldValue("myDatabase", "lastName", "3")
  get nextPXRecord("myDatabase")
end
```

getPXRecordNumber( )

TB30PDX.DLL

Engine Function	**PXRecNum()**
Description	Gets number of the current record in a table
Syntax	`getPXRecordNumber(<table alias>)`
Declaration	`LONG getPXRecordNumber(STRING)`
Parameters	`<table alias>` Alias of table to use
Returns	>= 0 Current record number in given table < 0 Error number

Example

```
--Put the record number into the status box
get getPXRecordNumber("myDatabase")
if it >= 0
  text of field "status" = "RECORD" && it && "out of" &&
getPXRecordCount("myDatabase")
end
```

getPXSortOrder( )

TB30PDX.DLL

Engine Function	**PXGetDefaults()**
Description	Gets the default sort order character
Syntax	`getPXSortOrder()`
Declaration	`STRING getPXSortOrder()`
Parameters	None
Returns	Sort order character

a	ASCII sort order
i	International sort order
n	Norwegian/Danish sort order
s	Swedish/Finnish sort order
d	Norwegian/Danish sort order for Paradox 4.0

If an error occurs, NULL is returned and `sysError` is set to the error number

Example

```
--Make sure sort order is international before opening table.  The open\
--will fail if the sort orders don't match
if getPXSortOrder() is not "i"
   get setPXSortOrder("i")
end
get openPXTable("myDatabase" "C:\data\accts", 0, 0)
```

getPXSwapSize( )

TB30PDX.DLL

Engine Function	**PXGetDefaults()**
Description	Gets the internal swap buffer size
Syntax	`getPXSwapSize()`
Declaration	`INT getPXSwapSize()`
Parameters	None
Returns	>= 0 Size of swap buffer
	< 0 Error number

Example

```
--Make sure swap buffer is 4K  * number of tables (which is the \
--recommended minimum) The number of tables is stored as a user \
--defined property of the book
if getPXSwapSize() < 4*numTables of this book
  get setPXINISwapSize(4*numTables of this book)
end
```

gotoPXNetRecordLock( )

TB30PDX.DLL

Engine Function	**PXNetRecGotoLock()**
Description	Returns to a previously locked record
Syntax	`gotoPXNetRecordLock(<table alias>,<lock handle>)`
Declaration	`INT gotoPXNetRecordLock(STRING,INT)`

Parameters

`<table alias>`	Alias of table to use
`<lock handle>`	Handle to previous record lock

Returns

0	Successful operation
< 0	Error number

Example

```
--Go back to the locked record.  Cannot go back to it by number
--because somebody else could have deleted a record (which would
--change the numbers)
lockHandle = lockPXNetRecord("myDatabase")
send compareValues -- might navigate in the database
get gotoPXNetRecordLock("myDatabase", lockHandle)
get unlockPXNetRecord("myDatabase", lockHandle)
```

gotoPXRecord( )

TB30PDX.DLL

Engine Function	**PXRecGoto()**
Description	Moves to a specified record of a table (in index order)
Syntax	`gotoPXRecord(<table alias>,<record number>)`
Declaration	`INT gotoPXRecord(STRING,LONG)`
Parameters	`<table alias>` Alias of table to traverse
	`<record number>` Number of record to move to
Returns	0 Successful operation
	< 0 Error number

Example

```
--Ask the user which record to go to by number
ask "Enter the number of the record to go to."
if it <> null
  get gotoPXRecord("myDatabase", it)
end
```

initializePX( )

TB30PDX.DLL

Engine Function	**PXWinInit()**
Description	Initializes the Paradox engine for concurrent operations. This function checks the UserName, NetNamePath, NetType, ShareLocal, and PX35Locking parameters in the WIN.INI file. If values have not been set for them, it sets the default values. (See `setPXUserInfo()` for descriptions of these parameters.)
Syntax	`initializePX(<client name>)`
Declaration	`INT initializePX(STRING)`
Parameters	`<client name>` Arbitrary name for client application. For example, you can use the name of the current book.
Returns	0 Successful operation < 0 Error number

Example

```
--Initialize the Paradox environment
get initializePX(name of this book)
if it < 0
  request getPXErrorString(it)
end
```

insertPXRecord( )

TB30PDX.DLL

Engine Function	**PXRecInsert()**
Description	Inserts an empty record into a table
Syntax	`insertPXRecord(<table alias>)`
Declaration	`INT insertPXRecord(STRING)`
Parameters	`<table alias>` Alias of table into which to insert record
Returns	0 Successful operation < 0 Error number

Example

```
--Insert record before the current record in a non-indexed table
ask "Enter record number to insert before."
if it <> null
  get gotoPXRecord("myDatabase", it)
end
get insertPXRecord("myDatabase")
```

isPXNetRecordLocked( )

TB30PDX.DLL

Engine Function	**PXNetRecLocked()**
Description	Determines if the current record has been locked
Syntax	`isPXNetRecordLocked(<table alias>)`
Declaration	`INT isPXNetRecordLocked(STRING)`
Parameters	`<table alias>` Alias of table to use

Returns

1	Net record is locked
0	Net record is not locked
< 0	Error number

Example

```
--Check whether the record is locked before locking it
get isPXNetRecordLocked("myDatabase")
if it = 1
  request getPXNetErrorUser()&& "has the record locked."
else
  get lockPXRecord("myDatabase")
end
```

isPXNetTableChanged( )

TB30PDX.DLL

Engine Function	**PXNetTblChanged()**
Description	Tests if a table has been changed
Syntax	`isPXNetTableChanged(<table alias>)`
Declaration	`INT isPXNetTableChanged(STRING)`
Parameters	`<table alias>` Alias of table to test

Returns

1	Network table is changed
0	Network table is not changed
< 0	Error number

Example

```
--Check whether the table has been changed by another user \
--before updating
get isPXNetTableChanged("myDatabase")
if it = 0
  get updatePXRecord("myDatabase")
end
```

isPXTableProtected( )

TB30PDX.DLL

Engine Function	**PXTblProtected()**
Description	Tests if a table is encrypted
Syntax	`isPXTableProtected(<table file name>)`
Declaration	`INT isPXTableProtected(STRING)`
Parameters	`<table file name>` File name of table to test

Returns

1	Table is protected
0	Table is not protected
< 0	Error number

Example

```
--Check whether the table is encrypted before asking for the password
get isPXTableProtected(text of field "table name")
if it = 1
  ask "Enter table password"
  if it <> null
    get addPXPassword(it)
  end
end
```

lastPXRecord( )

TB30PDX.DLL

Engine Function	**PXRecLast()**
Description	Moves to the last record of a table
Syntax	`lastPXRecord(<table alias>)`
Declaration	`INT lastPXRecord(STRING)`
Parameters	`<table alias>` Alias of table to use
Returns	0 Successful operation < 0 Error number

Example

```
--Go to the last record in the table and get the value
--of the field "name"
get lastPXRecord("myDatabase")
if it >= 0
  get getPXFieldValue("myDatabase", "name")
  if it >= 0
    request it
  end
end
```

lockPXNetFile( )

TB30PDX.DLL

Engine Function	**PXNetFileLock()**
Description	Locks a file
Syntax	`lockPXNetFile(<file name>,<lock type>)`
Declaration	`INT lockPXNetFile(STRING,INT)`

Parameters

`<file name>`	Name of file to lock
`<lock type>`	Type of lock 1 Full lock, no concurrency 2 Write lock 3 Prevent write lock 4 Prevent full lock, full concurrency

Returns

0	Successful operation
< 0	Error number

Example

```
--Attempt to lock the config file so it can be edited
get lockPXNetFile("n:\db.cfg", 1)
if it = 0
   send updateConfig
   get unlockPXFile("n:\db.cfg", 1)
else
   request "config file in use, could not update"
end
```

lockPXNetRecord( )

TB30PDX.DLL

Engine Function	**PXNetRecLock()**
Description	Locks the current record of a table
Syntax	`lockPXNetRecord(<table alias>)`
Declaration	`LONG lockPXNetRecord(STRING)`
Parameters	`<table alias>` Alias of table of which to lock current record
Returns	>= 0 Handle to record lock < 0 Error number

Example

```
--Lock the record and then update it
lockHandle = lockPXNetRecord("myDatabase")
if lockHandle >= 0
  get updatePXRecord("myDatabase")
  get unlockPXNetRecord("myDatabase")
end
```

lockPXNetTable( )

TB30PDX.DLL

Engine Function	**PXNetTblLock()**
Description	Locks a table
Syntax	`lockPXNetTable(<table alias>,<lock type>)`
Declaration	`INT lockPXNetTable(STRING,INT)`

Parameters

`<table alias>`	Alias of table to lock
`<lock type>`	Type of lock
	1 Full lock, no concurrency
	2 Write lock
	3 Prevent write lock
	4 Prevent full lock, full concurrency

Returns

0	Successful operation
< 0	Error number

Example

```
--Put a full lock on the table to prevent access by other users
get lockPXNetTable("myDatabase", 1)
if it < 0
  request getPXErrorString(it)
else
  send workWithTable
  get unlockPXNetTable("myDatabase", 1)
end
```

mapPXKey( )

TB30PDX.DLL

Engine Function	**PXKeyMap()**
Description	Maps one or more fields to a single index key field handle.
Syntax	`mapPXKey(<table alias>,<field names>,<key field name>,<mode>)`
Declaration	`LONG mapPXKey(STRING,STRING,STRING,INT)`

Parameters

`<table alias>`	Alias of table to use
`<field names>`	Names of fields to be used for the key
`<key field name>`	Name of index key field
`<mode>`	Index mode 0 Case-sensitive index 1 Case-insensitive index

Returns

>= 0	handle to index
< 0	Error number

Example

See `addPXKey`

nextPXRecord( )

TB30PDX.DLL

Engine Function	**PXRecNext()**
Description	Moves to the next record of a table
Syntax	`nextPXRecord(<table alias>)`
Declaration	`INT nextPXRecord(STRING)`
Parameters	`<table alias>` Alias of table to traverse
Returns	0 Successful operation < 0 Error number

Example

```
--Step through the records starting at the first one and clear the \
--field "Balance due"
get firstPXRecord("myDatabase")
while nextPXRecord("myDatabase") >= 0
  get setPXFieldValue("myDatabase", "Balance due", 0)
end
```

openPXBitmapWindow( )

TB30PDX.DLL

Engine Function	None
Description	Opens a child window for displaying a bitmap
Syntax	`openPXBitmapWindow(<bitmap handle>,<palette handle>, <parent handle>,<position/bounds>,<mode>, <background color>)`
Declaration	`INT openPXBitmapWindow(WORD,WORD,WORD,STRING,INT,STRING)`

Parameters

`<bitmap handle>`	Windows bitmap handle returned by `getPXGraphicBlob()`
`<palette handle>`	Optional Windows palette handle returned by `getPXGraphicBlobPalette()` (Enter 0 to use the ToolBook palette.)
`<parent handle>`	Window in which to create child
`<position/bounds>`	Comma-separated list of two or four numbers representing the position or bounds of the child window in screen coordinates
`<mode>`	Display mode for bitmap: 0 Normal - Bitmap is displayed in top-left corner of window. If `<position/bounds>` is a point, window is sized to fit bitmap. If `<position/bounds>` is a rectangle, window is sized to rectangle 1 Centered - If `<position/bounds>` is a point, bitmap is centered around point and window is sized to fit bitmap. If `<position/bounds>` is a rectangle, bitmap is centered in window and window is sized to rectangle 2 Stretched - If `<position/bounds>` is a point, window is sized to fit bitmap. If `<position/bounds>` is a rectangle, bitmap is stretched to fit window and window is sized to rectangle
`<background color>`	Comma-separated list representing the RGB values for the window background color. Defaults to the system window background color

Returns

> 0	Handle to bitmap window
< 0	Error number

Example

```
-- table is the currently open table, fieldName is the name of the
-- field that has the BLOB in it.
to handle showBitmap table,fieldName
  system hBMP,hPal,hWndBMP
  -- We get a PX type handle to the BLOB
  hPXToBMP = openPXBlobRead(table,fieldName)
  -- We put the BLOB in memory and get a Windows style handle to it
  hBMP = getPxGraphicBlob(hPXToBMP)
  -- Same with its associated palette
  hPal = getPxGraphicBlobPalette(hPXToBMP)
  -- Done with the PX handle
  get closePXBlob(hPXToBMP, 0)
  -- Open a window and show the graphic BLOB.
  hWndBMP = openPXBitmapWindow (hBMP, hPal, clientHandle of mainWindow, \
  pageUnitsToClient(position of ellipse "Center"), 2, \
   rgbFill of ellipse "Center")
   -- Imagine the ellipse "center" to be a point, with the mode set to 2,
   -- the bitmap will show at its normal size centered on
   -- the 1 dimensional ellipse.
end showBitmap
```

openPXBlobRead( )

TB30PDX.DLL

Engine Function	**PXBlobOpenRead()**	
Description	Opens a BLOB for reading operations	
Syntax	`openPXBlobRead(<table alias>,<field name>)`	
Declaration	`LONG openPXBlobRead(STRING,STRING)`	
Parameters	`<table alias>`	Alias of table in which to open BLOB
	`<field name>`	Name of BLOB field
Returns	>= 0	Handle to BLOB
	< 0	Error number

Example

```
--Read a memo BLOB into a field
hBlob = openPXBlobRead("myDatabase", "ReviewText")
text of field "review" = getPXMemoBlob(hBlob)
get closePXBlob(hBlob, 0)
```

openPXBlobWrite( )

TB30PDX.DLL

Engine Function	**PXBlobOpenWrite()**
Description	Opens a BLOB for writing operations
Syntax	`openPXBlobWrite(<table alias>,<field name>,<size>,<save current>)`
Declaration	`LONG openPXBlobWrite(STRING,STRING,LONG,INT)`

Parameters

`<table alias>`	Alias of table in which to open BLOB
`<field name>`	Name of BLOB field
`<size>`	Size of BLOB field (0 - 256MB)
`<save current>`	Copy mode: 0 New BLOB, no copying takes place 1 Copy BLOB, new private BLOB is created of `<size>` bytes

Returns

>= 0	Handle to BLOB
< 0	Error number

Example

```
--place 10K of binary data into a BLOB
hBlob = openPXBlobWrite("myDatabase", "bits", 10240, 0)
get setPXBlob(hBlob, 10240, 0, hBits)
get closePXBlob(hBlob, 1)
```

openPXTable( )

TB30PDX.DLL

Engine Function	**PXTblOpen()**
Description	Opens a table
Syntax	`openPXTable(<table alias>,<table filename>,<index ID>,<save every change>)`
Declaration	`INT openPXTable(STRING,STRING,INT,INT)`

Parameters

`<table alias>`	A name you assign to the table to be used for subsequent access. Can be arbitrary, but must be unique to this view of the table during this session.
`<table filename>`	File name of table to open
`<index ID>`	Index type by which to order records: 0 Order of primary index (If you have not created an index for the table, enter 0) 1-255 Field number for case-sensitive, single-field, secondary index >256 Field handle for composite or case-insensitive, single-field index
`<save every change>`	Mode for saving changes:** 0 Changes are saved in swap buffer 1 Changes are saved to disk

**Changes to a shared table are always saved to disk, regardless of mode

Returns

0	Successful operation
< 0	Error number

Example

```
-- "myTable" is an arbitrary alias.
-- curTable is a user property set to the name of the db file (without
-- extension).
get openPXTable("myTable", curTable of this book, 0, 0)
if it < 0
  request getPXErrorString(it)
end if
```

packPXTable( )

TB30PDX.DLL

Engine Function	None
Description	Packs the database and removes any space occupied by deleted records.
Syntax	`packPXTable(<table alias>)`
Declaration	`INT packPXTable(STRING)`
Parameters	`<table alias>` Table alias specified when the table was opened
Returns	0 Successful operation < 0 Error number

Example

```
--Pack the database before copying it
--(This can take a while)
sysCursor = 4
get packPXTable("myDatabase")
if it = 0
  get copyPXTable("c:\data\myData", "n:\data\shareDB")
end
```

Comments This can be a long process and is not recommended as part of normal table operations.

previousPXRecord( )

TB30PDX.DLL

Engine Function	**PXRecPrev()**
Description	Moves to the previous record of a table
Syntax	`previousPXRecord(<table alias>)`
Declaration	`INT previousPXRecord(STRING)`
Parameters	`<table alias>` Alias of table to traverse
Returns	0 Successful operation < 0 Error number

Example

```
--Go back one record and compare the date field against today's date
get previousPXRecord("myDatabase")
if it >= 0
  if getPXFieldValue("myDatabase", "date") < sysDate
    request "book is overdue"
  else
    request "book is not overdue"
  end
end
```

queryPXKey( )

TB30PDX.DLL

Engine Function	**PXKeyQuery()**
Description	Gets information about an index. Use this function to get the index handle that you pass in `openPXTable()`.
Syntax	`queryPXKey(<index file name>)`
Declaration	`STRING queryPXKey(STRING)`
Parameters	`<index file name>` Name of index file to query
Returns	List containing field name, field count, mode, list of field handles, and index ID.

For case-sensitive, single-field index:
```
    <field name>, 1, 0, <field handle>, <index ID>
```
For composite or case-insensitive, single-field index:
```
    <field name>, <field count>, 1, <field handles>, <index
ID>
```
If an error occurs, null is returned and `sysError` is set to the error number

Example

```
--Search index files for table to find the right index ID
--fileList already contains a comma-seperated list of files

found = false
i = 1
while not found and i <= itemCount(fileList)
  get queryPXKey(item i of fileList)
  if item 1 of it = "Last name First"
    indexID = last item of it
    found = TRUE
  else
    increment i
  end
end
```

refreshPXNetTable( )

TB30PDX.DLL

Engine Function	**PXNetTblRefresh()**
Description	Resynchronizes a shared table
Syntax	`refreshPXNetTable(<table alias>)`
Declaration	`INT refreshPXNetTable(STRING)`
Parameters	`<table alias>` Alias of table to use
Returns	0 Successful operation < 0 Error number

Example

```
--Test whether the table has been changed and refresh it if it has
get isPXNetTableChanged("myDatabase")
if it = 1
  get refreshPXNetTable("myDatabase")
end
```

renamePXTable( )

TB30PDX.DLL

Engine Function	**PXTblRename()**
Description	Changes the base name of a table family
Syntax	`renamePXTable(<source table>,<destination table>)`
Declaration	`INT renamePXTable(STRING,STRING)`

Parameters	`<source table>`	Base name of source table family to be renamed
	`<destination table>`	New base name for the source table family

Returns	0	Successful operation
	< 0	Error number

Example

```
--Rename the old database before copying it to a local drive
get renamePXtable("c:\data\myData", "c:\data\oldData")
get copyPXTable("n:\data\commData", "c:\data\myData")
```

savePX( )

TB30PDX.DLL

Engine Function	**PXSave()**	
Description	Saves the swap buffer to disk	
Syntax	`savePX()`	
Declaration	`INT savePX()`	
Parameters	None	
Returns	0	Successful operation
	< 0	Error number

Example

```
--Force the swap buffer to write to disk every five appends to the table
if appendCount >= 5
  get savePX()
end
```

searchPXField( )

TB30PDX.DLL

Engine Function	**PXSrchFld()**
Description	Searches a table on a specified field. If the table is currently open on the primary index, you can search on any secondary index, whether composite or not. If the table is currently open on a secondary index (composite or not), you can only search on that index. The mode parameter controls the scope of the search. Mode 0 searches from the first record in the table looking for an exact match on the data in the key fields. Mode 1 starts from the current record searching for an exact match. Mode 2 starts from the first record and searches for a match. For modes 0 and 1, if -89 is returned the current record is not moved from where it was before the search. For mode 2, if there is not an exact match, two posibilities exist: 1. There is a record that is greater than the search value. The current record is moved to the first such record and -89 is returned. 2. The search value is greater than all records in the table. The current record is moved to the end of the table and -101 is returned
Syntax	`searchPXField(<table alias>,<field name>,<index ID>,<mode>)`
Declaration	`INT searchPXField(STRING,STRING,WORD,INT)`

Parameters

`<table alias>`	Alias of table to search
`<field name>`	Field name of key for case-sensitive, single-field, secondary index only; otherwise leave null
`<index ID>`	Handle of index (if field name is not provided)
`<mode>`	Search mode: 0 Search from beginning of table 1 Search from next record in table 2 Search for nearest record if no match

Returns

0	Successful operation
< 0	Error number

Example

```
--Open the table with a case insensitive index and search for a \
--last name and age
--When the index was created, the ID was saved in a user-defined \
--property of the book
get openPXTable("myDatabase", "c:\data\rolodex.db",lastName_Age_Index of this
book, 0)
get emptyPXRecord("myDatabase") -- set search buffer to known state
--set values to search for
get setPXFieldValue("myDatabase", "name", text of field "name")
get setPXFieldValue("myDatabase", "age", text of field "age")
get searchPXField("myDatabase", "", lastName_Age_Index of this book, 0)
if it < 0
   request getPXErrorString(it)
end
```

searchPXKey( )

TB30PDX.DLL

Engine Function	**PXSrchKey()**
Description	Searches a table for a key match. The mode parameter controls the scope of the search. Mode 0 searches from the first record in the table looking for an exact match on the data in the key fields. Mode 1 starts from the current record searching for an exact match. Mode 2 starts from the first record and searches for a match. For modes 0 and 1, if -89 is returned the current record is not moved from where it was before the search. For mode 2, if there is not an exact match two posibilities exist: 1. There is a record that is greater than the search value. The current record is moved to the first such record and -89 is returned. 2. The search value is greater than all records in the table. The current record is moved to the end of the table and -101 is returned
Syntax	`searchPXKey(<table alias>,<number of fields>,<field values>,<mode>)`
Declaration	`INT searchPXKey(STRING,INT,STRING,INT)`

Parameters

`<table alias>`	Alias of table to search
`<fields>`	Number of fields (<= fields in primary key)
`<field values>`	List of values to search for
`<mode>`	Search mode: 0 Search from beginning of table 1 Search from next record in table 2 Search for nearest record if no match

Returns

0	Successful operation -- exact match found
< 0	Error

Example

```
--Search for name in primary index
get searchPXKey("myDatabase", "2", "Smith, John", 0)
if it >= 0
  request "found"
else
  request "not found"
end
```

setPXBitmapWindowInfo( )

TB30PDX.DLL

Engine Function	**None**
Description	Sets the display attributes for a bitmap window. Null (or 0) data parameters are not changed.
Syntax	`setPXBitmapWindowInfo(<window handle>,<bitmap handle>,\` `<palette handle>,<mode>,<background color>)`
Declaration	`INT setPXBitmapWindowInfo(INT,INT,INT,INT,STRING)`

Parameters

`<window handle>`	Bitmap window handle
`<palette handle>`	Optional Windows palette handle
`<bitmap handle>`	Windows bitmap handle
`<mode>`	Display mode for bitmap: 1 Normal - Bitmap is displayed in top-left corner of window. If `<position/bounds>` is a point, window is sized to fit bitmap. If `<position/bounds>` is a rectangle, window is sized to rectangle 2 Centered - If `<position/bounds>` is a point, bitmap is centered around point and window is sized to fit bitmap. If `<position/bounds>` is a rectangle, bitmap is centered in window and window is sized to rectangle 3 Stretched - If `<position/bounds>` is a point, window is sized to fit bitmap. If `<position/bounds>` is a rectangle, bitmap is stretched to fit window and window is sized to rectangle
`<background color>`	Comma-separated list representing the RGB values for the window background color

Returns

0	Successful operation
< 0	Error number

Example

```
-- this will show the graphic blob in a window
-- hWndBitmap is a handle to a window you've already created,
-- possibly with openPXBitmapWindow()
hPrivateBlob = openPXBlobRead("myTable", "Picture")
if hPrivateBlob < 0
  send PXError hPrivateBlob   -- cleanup
  break
else
  hBitmap = getPXGraphicBlob(hPrivateBlob)
  hPalette = getPXGraphicBlobPalette(hPrivateBlob)
  get closePXBlob(hPrivateBlob, 0)
end if
get setPXBitmapWindowInfo(hWndBitmap, hBitmap, hPalette, 2, \
rgbfill of this page)
if it < 0
  request getPXErrorString(it)
end if
```

setPXBlob( )

TB30PDX.DLL

Engine Function	**PXBlobPut()**
Description	Writes data to a BLOB
Syntax	`setPXBlob(<BLOB handle>,<size>,<offset>,<buffer>)`
Declaration	`INT setPXBlob(INT,DWORD,LONG,WORD)`

Parameters

`<BLOB handle>`	Handle to BLOB
`<size>`	Size of buffer (1 - 256MB)
`<offset>`	Offset from start of BLOB to start writing
`<buffer>`	Handle to BLOB data

Returns

0	Successful operation
< 0	Error number

Example

See `openPXBlobWrite()`

setPXBlobFromFile( )

TB30PDX.DLL

Engine Function	None
Description	Imports a file directly into a BLOB field
Syntax	`setPXBlobFromFile(<BLOB handle>,<size>,<file offset>,<offset>,<file name>)`
Declaration	`INT setPXBlobFromFile(INT,DWORD,DWORD,DWORD,STRING)`

Parameters

`<BLOB handle>`	Handle returned by `openPXBlobWrite()`
`<size>`	Number of bytes of file to read
`<file offset>`	Offset from beginning of file to start reading
`<offset>`	Offset from beginning of BLOB to write
`<file name>`	File to read from.

Returns

0	Successful operation
< 0	Error number

Example

```
--Store wave file as BLOB
hBlob = openPXBlobWrite("myDatabase", "wave", fileSize, 0)
if hBlob >= 0
  get setPXBlobFromFile(hBlob, fileSize, 0, 0, fileName)
  get closePXBlob(hBlob, 1)
end
```

Comments If size is 0 the whole file is read up to 256MB.

setPXFieldValue( )

TB30PDX.DLL

Engine Functions:	**PXPutAlpha(), PXPutDate(), PXPutDouble(), PXPutLong(), PXPutShort()**
Description	Sets the contents of a field in the current record of a table
Syntax	`setPXFieldValue(<table alias>,<field name>,<field value>)`
Declaration	`INT setPXFieldValue(STRING,STRING,STRING)`

Parameters

`<table alias>`	Alias of table to use
`<field name>`	Name of field to set
`<field value>`	New contents for field

Returns

0	Successful operation
< 0	Error number

Example

```
--Set the fields in the current record from the fields on the page
get setPXFieldValue("myDatabase", "name", text of field "name")
get setPXFieldValue("myDatabase", "age", text of field "age")
get setPXFieldValue("myDatabase", "birthDate", text of field "birthDate")
```

setPXGraphicBlob( )

TB30PDX.DLL

Engine Function	None
Description	Creates a graphic BLOB from a Windows GDI handle to a bitmap
Syntax	`setPXGraphicBlob(<BLOB handle>,<hBitmap>,<hPalette>)`
Declaration	`INT setPXGraphicBlob(INT,WORD,WORD)`

Parameters

`<BLOB handle>`	Handle returned by `openPXBlobWrite()`
`<hBitmap>`	Handle to Windows device dependent bitmap
<hPalette>	Handle to Windows palette for bitmap (0 for Windows default palette)

Returns

0	Successful operation
< 0	Error number

Example

```
--Save bitmap from ToolBook resource as a BLOB in the database
hBitmap = GDIHandle(bitmap "logo")
hBlob = openPXBlobWrite("myDatabase", "graphic", item 5 of \
resourceInfo of bitmap "logo", 0)
if hBlob >= 0
  get setPXGraphicBlob(hBlob, hBitmap, hPalette)
  get closePXBlob(hBlob, 1)
end
```

setPXGraphicBlobFromFile( )

TB30PDX.DLL

Engine Function	None	
Description	Imports a bitmap file into a BLOB field	
Syntax	`setPXGraphicBlobFromFile(<BLOB handle>, <file name>)`	
Declaration	`INT setPXGraphicBlobFromFile(INT,STRING)`	
Parameters	`<BLOB handle>`	Handle returned by `openPXBlobWrite()`
	`<file name>`	File to read from.
Returns	0	Successful operation
	< 0	Error number

Example

```
--Put 256color.bmp bitmap into BLOB field of current record
hBlob = openPXBlobWrite("myDatabase", "graphic", fileSize, 0)
if hBlob >= 0
   get setPXGraphicBlobFromFile(hBlob, "c:\windows\256color.bmp")
   get closePXBlob(hBlob, 1)
end
```

Comments File must be a Windows bitmap (.BMP or .DIB). All other graphic file formats are considered unstructured.

setPXINIMaxFiles( )

TB30PDX.DLL

Engine Function	**PXSetDefaults()**
Description	Sets the value of the MaxFiles entry of the [Paradox Engine] section of the WIN.INI file. The Paradox Engine reads this value the first time it is initialized to set the maximum number of files that can be used by the Engine. Calling this function does not cause the Engine to read the defaults, it only writes to the INI file.
Syntax	`setPXINIMaxFiles(<maximum files>)`
Declaration	`INT setPXINIMaxFiles(INT)`
Parameters	`<maximum files>` Number of file handles (3 - 255)
Returns	0 Successful operation < 0 Error number

Example

See `getPXMaxFiles()`

setPXINIMaxLocks( )

TB30PDX.DLL

Engine Function	**PXSetDefaults()**
Description	Sets the value of the MaxLocks entry of the [Paradox Engine] section of the WIN.INI file. The Paradox Engine reads this value the first time it is initialized to set the maximum number of locks that can be used by the Engine. Calling this function does not cause the Engine to read the defaults, it only writes to the INI file.
Syntax	`setPXINIMaxLocks(<maximum locks>)`
Declaration	`INT setPXINIMaxLocks(INT)`
Parameters	`<maximum locks>` Number of record locks (1 - 128)
Returns	0 Successful operation < 0 Error number

Example

See `getPXMaxLocks()`

setPXINIMaxTables( )

TB30PDX.DLL

Engine Function	**PXSetDefaults()**
Description	Sets the value of the MaxTables entry of the [Paradox Engine] section of the WIN.INI file. The Paradox Engine reads this value the first time it is initialized to set the maximum number of tables that can be used by the Engine. Calling this function does not cause the Engine to read the defaults, it only writes to the INI file.e
Syntax	`setPXINIMaxTables(<maximum tables>)`
Declaration	`INT setPXINIMaxTables(INT)`
Parameters	`<maximum tables>` Number of tables (1 - 64)
Returns	0 Successful operation < 0 Error number

Example

See `getPXMaxTables()`

setPXMemoBlob( )

TB30PDX.DLL

Engine Function	None	
Description	Copies a string into a memo BLOB	
Syntax	`setPXMemoBlob(<BLOB handle>,<text>)`	
Declaration	`INT setPXMemoBlob(INT,STRING)`	
Parameters	`<BLOB handle>`	Handle returned by `openPXBlobWrite()`
	`<text>`	String of text, < 64K bytes in length.
Returns	0	Successful operation
	< 0	Error number

Example

```
--Save text of notes field into a BLOB in the database
hBlob = openPXBlobWrite("myDatabase", "notes", charCount(text of \
field "notes"), 0)
if hBlob >= 0
   get setPXMemoBlob(hBlob, text of field "notes")
   get closePXMemoBlob(hBlob, 1)
end
```

▸

setPXRaw( )

TB30PDX.DLL

Engine Function	**PXRawPut()**
Description	Writes raw data to the current record of a table
Syntax	`setPXRaw(<table alias>,<data handle>,<size>)`
Declaration	`INT setPXRaw(STRING,INT,INT)`

Parameters

`<table alias>`	Alias of table to use
`<data handle>`	Handle to raw data
`<size>`	Size of raw data

Returns

0	Successful operation
< 0	Error number

Example

```
--Restore the original contents of the current record from saved buffer
get setPXRaw("myDatabase", hRawData, sizeofData)
-- Free the memory allocated for the raw data buffer in the call to
getRawData()
get freePXBlobMemory(hRawData)
clear hRawData
```

setPXSortOrder( )

TB30PDX.DLL

Engine Function	**PXSetDefaults()**
Description	Sets the sort order
Syntax	`setPXSortOrder(<sort order>)`
Declaration	`INT setPXSortOrder(STRING)`
Parameters	`<sort order>` Sort order character:

a	ASCII sort order
i	International sort order
n	Norwegian/Danish sort order
s	Swedish/Finnish sort order
d	Norwegian/Danish sort order for Paradox 4.0

Returns

0	Successful operation
< 0	Error number

Example

See `getPXSortOrder()`

setPXINISwapSize( )

TB30PDX.DLL

Engine Function	**PXSetDefaults()**
Description	Sets the value of the SwapSize entry of the [Paradox Engine] section of the WIN.INI file. The Paradox Engine reads this value the first time it is initialized to set the size of the Swap Buffer that will be used by the Engine. Calling this function does not cause the Engine to read the defaults, it only writes to the INI file.e
Syntax	`setPXINISwapSize(<swap size>)`
Declaration	`INT setPXINISwapSize(INT)`
Parameters	`<swap size>` Swap size in K (8 - 256). Minimum 4K per table
Returns	0 Successful operation < 0 Error number

Example

See `getPXSwapSize()`

setPXTableCreateMode( )

TB30PDX.DLL

Engine Function	**PXTblCreateMode()**
Description	Sets the mode for creating tables
Syntax	`setPXTableCreateMode(<mode>)`
Declaration	`INT setPXTableCreateMode(INT)`

Parameters

`<mode>`	Mode for creating tables:	
	0	Create Paradox 3.5 compatible tables
	1	Create Paradox 4.0 compatible tables

Returns

0	Successful operation
< 0	Error number

Example

```
--Ask the user which Paradox version table to create
request "Which version Paradox table do you want to create?" with "4.0" or
"3.5"
if it = "4.0"
  get setPXTableCreateMode(1)
else
  get setPXTableCreateMode(0)
end
```

setPXTableMaxSize( )

TB30PDX.DLL

Engine Function **PXTblMaxSize()**

Description Sets the maximum table size and block size for new tables. Block size is determined by the maximum table size:

Maximum table size	Block size
64MB	1KB
128MB	2KB
256MB	4KB

Syntax `setPXTableMaxSize(<size>)`

Declaration `INT setPXTableMaxSize(INT)`

Parameters `<size>` Maximum table size in megabytes (64,128, or 256)

Returns

0	Successful operation
< 0	Error number

Example

```
--Set the maximum size for the table
get setPXMaxTableSize(64) -- 64 megabytes
```

unlockPXNetFile( )

TB30PDX.DLL

Engine Function	**PXNetFileUnlock()**
Description	Unlocks a file
Syntax	`unlockPXNetFile(<file name>,<lock type>)`
Declaration	`INT unlockPXNetFile(STRING,INT)`

Parameters

`<file name>`	Name of file to unlock	
`<lock type>`	Type of lock:	
	1	Full lock, no concurrency
	2	Write lock
	3	Prevent write lock
	4	Prevent full lock, full concurrency

Returns

0	Successful operation
< 0	Error number

Example

See `lockPXNetFile()`

unlockPXNetRecord( )

TB30PDX.DLL

Engine Function	**PXNetRecUnlock()**	
Description	Unlocks the current record of a table	
Syntax	`unlockPXNetRecord(<table alias>,<lock handle>)`	
Declaration	`INT unlockPXNetRec(STRING,INT)`	
Parameters	`<table alias>`	Alias of table to use
	`<lock handle>`	Handle to record lock
Returns	0	Successful operation
	< 0	Error number

Example

See `lockPXNetRecord()`

unlockPXNetTable( )

TB30PDX.DLL

Engine Function	**PXNetTblUnlock()**	
Description	Unlocks a table	
Syntax	`unlockPXNetTable(<table alias>,<lock type>)`	
Declaration	`INT unlockPXNetTable(STRING,INT)`	
Parameters	`<table alias>`	Alias of table to unlock
	`<lock type>`	Type of lock: 1 Full lock, no concurrency 2 Write lock 3 Prevent write lock 4 Prevent full lock, full concurrency
Returns	0	Successful operation
	< 0	Error number

Example

See `lockPXNetTable()`

updatePXRecord( )

TB30PDX.DLL

Engine Function	**PXRecUpdate()**
Description	Updates the current record in a table
Syntax	`updatePXRecord(<table alias>)`
Declaration	`INT updatePXRecord(STRING)`
Parameters	`<table alias>` Alias of table to use
Returns	0 Successful operation < 0 Error number

Example

```
--Write the contents of the current record buffer to the current record
request "Save changes to record?" with "OK" or "Cancel"
if it = "OK"
  get updatePXRecord("myDatabase")
end
```

upgradePXTable( )

TB30PDX.DLL

Engine Function	**PXTblUpgrade()**
Description	Upgrades an older Paradox table (Paradox 3.5 or later)
Syntax	`upgradePXTable(<table alias>)`
Declaration	`INT upgradePXTable(STRING)`
Parameters	`<table alias>` Alias of table to upgrade
Returns	0 Successful operation < 0 Error number

Example

```
--Upgrade the requested table to 4.0 format
ask "Enter the name of the table to upgrade."
if it <> null
  get upgradePXTable(it)
end
```

writePXBlobToFile( )

TB30PDX.DLL

Engine Function	None
Description	Exports a BLOB to a DOS file
Syntax	`writePXBlobToFile(<BLOB handle>,<file name>,<mode>)`
Declaration	`LONG writePXBlobToFile(INT,STRING,INT)`

Parameters

`<BLOB handle>`	Handle returned by `openPXBlobRead()` or `openPXBlobWrite()`
`<file name>`	Path and file name of file to write to
`<mode>`	Mode in which to write file: 0 normal; can be read or written without restriction. 1 read-only; cannot be opened for writing by other applications. 2 overwrite; if the file exists overwrite it. 3 read-only & Overwrite.

Returns

> 0	number of bytes written to the file if successful
< 0	Error number

Comments If the specified file exists and modes 0 or 1 are specified, no file is written and an error is returned. A Windows .BMP file is written.

Example

```
--Save wave file to disk
hBlob = openPXBlobRead("myDatabase", "sound")
if hBlob >= 0
   get writePXblobToFile(hBlob, "sound.wav", 2)
   playSound("sound.wav")
   get closePXBlob(hBlob, 0)
end
```

writePXGraphicBlobToFile( )

TB30PDX.DLL

Engine Function	None
Description	Exports a graphic BLOB as a bitmap file
Syntax	`writePXGraphicBlobToFile(<BLOB handle>, <file name>, <mode>)`
Declaration	`LONG writePXGraphicBlobToFile(INT,STRING,INT)`

Parameters

`<BLOB handle>`	handle returned by `openPXBlobRead()` or `openPXBlobWrite()`
`<file name>`	path and file name of file to write to
`<mode>`	Mode in which to write file: 0 normal; can be read or written without restriction. 1 read-only; cannot be opened for writing by other applications. 2 overwrite; if the file exists overwrite it. 3 read-only & Overwrite.

Returns

> 0	number of bytes written to the file if successful
< 0	Error number

Comments If the specified file exists and modes 0 or 1 are specified, no file is written and an error is returned. A windows BMP file is written.

Example

```
--Save bitmap out to file
hBlob = openPXBlobRead("myDatabase", "graphic")
if hBlob >= 0
  get writePXGraphicBlobToFile(hBlob, "graphic.bmp", 0)
  get closePXBlob(hBlob)
end
```

getPXUserInfo( )

TB30PDX.DLL

Engine Function	**None**
Description	Returns a comma-separated list of values from the [Paradox engine] section of the WIN.INI file
Syntax	`getPXUserInfo(<list of values>)`
Declaration	`STRING getPXUserInfo()`
Parameters	`none`
Returns	UserName, NetNamePath, NetType, ShareLocal, PX35Locking (For descriptions of the returned values, see setPXUserInfo().) If an error occurs, null is returned and `sysError` is set to the error number.

Example

```
get getPXUserInfo()
if it is NULL
  request sysError
end if
if item 1 of it is "Charlie"
  set item 1 of it to "Charles"
  get setPXUserInfo(it)
end if
```

setPXUserInfo( )

TB30PDX.DLL

Engine Function	None
Description	Takes a comma-separated list of values and writes them to the [Paradox engine] section of the WIN.INI file
Syntax	`setPXUserInfo(<list of values>)`
Declaration	`INT getPXUserInfo(STRING)`

Parameters `<list of values>` Comma-separated list of values:

UserName	the name of the user of the application. If no name is entered, the default "PxEngine" is used in the WIN.INI file.
NetNamePath	the location of PARADOX.NET or PDOXUSRS.NET. (The default is C:\) If you are using only local tables on your own PC, enter C:\
NetType	Always use a value of 2, even if you're not using a network. (Default is 2.)
ShareLocal	"yes" if your tables are to be shared with Paradox or with an engine application running in a DOS window. (Default is "no.")
PX35Locking	"yes" if the Paradox 3.5 locking protocol is used. (Default is "no.")

Returns

0	Successful operation
< 0	Error number

Example

```
get setPXUserInfo("John User,c:\,2,No,No")
```

getPXSwapSizeFromINI()

TB30PDX.DLL

Engine Function	none	
Description	Gets the value of the SwapSize entry from the [Paradox Engine] section of the WIN.INI file. The Paradox Engine reads this value the first time it is initialized to set the maximum size of the swap buffer that can be used by the Engine. Calling this function does not cause the Engine to read the defaults, it only queries the .INI file. To get the value the Engine is currently using call getPXSwapSize().	
Syntax	`getPXSwapSizeFromINI()`	
Declaration	`INT`	`getPXSwapSizeFromINI()`
Parameters	None	
Returns	>= 0	ini setting
	< 0	Error number

Example

```
if getPXSwapSizeFromINI() < 64
  request "Swap buffer is less than 64K, this application runs"&CRLF\
    "faster with a larget swap buffer."
```

getPXMaxTablesFromINI()
TB30PDX.DLL

Engine Function	none	
Description	Gets the value of the MaxTables entry from the [Paradox Engine] section of the WIN.INI file. The Paradox Engine reads this value the first time it is initialized to set the maximum number of tables that can be used by the Engine. Calling this function does not cause the Engine to read the defaults, it only queries the .INI file. To get the value the Engine is currently using call getPXMaxTables().	
Syntax	`getPXMaxTablesFromINI()`	
Declaration	`INT`	`getPXMaxTablesFromINI()`
Parameters	None	
Returns	>= 0	Maximum tables usable by engine
	< 0	Error number

Example
```
--get the current INI setting
get getPXMaxTablesFromINI()

--If the current ini value is less than what I need then try to reset it.
if it < myExpectedValue
   get setPXINIMaxTables(myExpectedValue)
   get initializePX(myTable)   --somebody else has initialized the engine.
   get getPXMaxTables()
   if it < myExpectedValue
--somebody else has linked the engine and I cannot set the maxtables to what
--I need so I will give error and exit
      request "Unable to set number of tables.  Please Exit all other 
Paradox"&CRLF&\
      "Engine applications and try again."
      send exit
   end
end
```

getPXMaxFilesFromINI()

TB30PDX.DLL

Engine Function	none
Description	Gets the value of the MaxFiles entry from the [Paradox Engine] section of the WIN.INI file. The Paradox Engine reads this value the first time it is initialized to set the maximum number of files that can be used by the Engine. Calling this function does not cause the Engine to read the defaults, it only queries the .INI file. To get the value the Engine is currently using call getPXMaxFiles().
Syntax	`getPXMaxFilesFromINI()`
Declaration	`INT     getPXMaxFilesFromINI()`
Parameters	None
Returns	>= 0 Maximum tables usable by engine < 0 Error number

Example

```
--get the current INI setting
get getPXMaxFilesFromINI()

--If the current ini value is less than what I need then try to reset it.
if it < myExpectedValue
   get setPXINIMaxFiles(myExpectedValue)
   get initializePX(myTable)   --somebody else has initialized the engine.
   get getPXMaxFiles()
   if it < myExpectedValue
   --somebody else has linked the engine and I cannot set the maxfiles to
what
   --I need so I will give error and exit
      request "Unable to set number of files.  Please Exit all other
Paradox"&CRLF&\
      "Engine applications and try again."
      send exit
   end
end
```

getPXMaxLocksFromINI()

TB30PDX.DLL

Engine Function	none
Description	Gets the value of the MaxLocks entry from the [Paradox Engine] section of the WIN.INI file. The Paradox Engine reads this value the first time it is initialized to set the maximum number of locks that can be used by the Engine. Calling this function does not cause the Engine to read the defaults, it only queries the .INI file. To get the value the Engine is currently using call getPXMaxLocks().
Syntax	`getPXMaxLocksFromINI()`
Declaration	`INT getPXMaxLocksFromINI()`
Parameters	None
Returns	>= 0 Maximum tables usable by engine < 0 Error number

Example

```
--get the current INI setting
get getPXMaxLocksFromINI()

--If the current ini value is less than what I need then try to reset it.
if it < myExpectedValue
  get setPXINIMaxLocks(myExpectedValue)
  get initializePX(myTable)   --somebody else has initialized the engine.
  get getPXMaxLocks()
  if it < myExpectedValue
  --somebody else has linked the engine and I cannot set the maxlocks to
what
  --I need so I will give error and exit
    request "Unable to set number of locks.  Please Exit all other
Paradox"&CRLF&\
    "Engine applications and try again."
    send exit
  end
end
```

checkDBIndex( )

TB30DB3.DLL

Syntax	`checkDBIndex(<file name>)`
Declaration	`INT checkDBIndex(STRING)`
Description	Checks the specified index file against the current dBASE file.
Parameter	`<file name>` Valid file name of the index file to check.

Returns

1	The index file is accurate.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 18	Bad key reference.
- 19	Multiple keys refer to same record.
- 25	A record newer than the corresponding key was found.
- 26	A key was not sorted.
- 22	No key for a record.
- 53	No index file with the specified name.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
to handle buttonClick
  get checkDBIndex("c:\egypt\hiero.ndx")
  if It <= - 3
    local newFileName
    ask "Index file not accurate. Try new index file?"
    if sysError <> cancel and It <> null
      put It into newFileName
      get checkDBIndex(newFileName)
    end if
  else
    request "Accurate index file"
  end if
end buttonClick
```

closeAllDBFiles( )

TB30DB3.DLL

Syntax	`closeAllDBFiles()`
Declaration	`INT closeAllDBFiles()`
Description	Closes all open dBASE and index files.
Parameters	None

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
to handle exit
  get closeAllDBFiles()
end exit
```

closeDBFile( )

TB30DB3.DLL

Syntax	`closeDBFile(<file name>)`
Declaration	`INT closeDBFile(STRING)`
Description	Closes the specified dBASE file and all related index files.
Parameter	`<file name>` The name of the dBASE file to close.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 28	No dBASE file with the specified name.
- 29	The index file could not be closed.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
to handle error value, string
  request "Error (" & val & ") " & string
  get closeDBFile(fileName)
end error
```

closeDBIndexFile( )

TB30DB3.DLL

Syntax `closeDBIndexFile(<file name>)`

Declaration `INT closeDBIndexFile(STRING)`

Description Closes the specified dBASE index file.

Parameter `<file name>` The name of the dBASE index file to close.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 53	No index file with the specified name.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
to handle error value, string
  request "Error (" & val & ") " & string
  get closeDBIndexFile(fileName)
end error
```

createDBFieldTag( )

TB30DB3.DLL

Syntax	`createDBFieldTag(<number of fields>)`	
Declaration	`INT createDBFieldTag(WORD)`	
Description	Creates a field tag that specifies the number of fields in a dBASE file to create with `createDBFile()`. For details about creating a dBASE file, see `createDBFile()`.	
Parameter	`<number of fields>`	The number of fields the tag can contain. It can be a valid number of fields for a dBASE file.
Returns	If no error occurs, `createDBFile()` returns a number that is the `<field tag number>` parameter to reference the tag in `setDBFieldTag()`, `createDBFile()`, and `freeDBFieldTag()`. If an error occurs, such as not enough memory, `createDBFieldTag()` returns 0.	

Example

```
--Gets return value for tag number
put createDBFieldTag(noFields) into tagNumber
if tagNumber = 0
  request "Error creating field tag"
  break
end if
```

createDBFile( )

TB30DB3.DLL

Syntax `createDBFile(<file name>,<field tag number>,<preserve existing>)`

Declaration `INT createDBFile(STRING,WORD,WORD)`

Description Creates and opens a dBASE file with the specified name and number of fields using the contents of the field tag data structure. The number of fields and their potential values depend on parameters passed with `createDBFieldTag()` and `setDBFieldTag()`.

To create a dBASE file with the dBASE DLL:

1. Create a field tag of the appropriate size with `createDBFieldTag()`, with a parameter that indicates the number of fields the tag can contain. The return value from this function is a number used to reference the field tag when calling `createDBFile()` and `freeDBFieldTag()`.

   A field tag is an array of data structures used to create a dBASE file.
2. Set the contents of the field tag with `setDBFieldTag()`, with parameters for each field that indicate the tag number, name, type, width, and decimal precision (numeric fields only).
3. Call `createDBFile()` with the appropriate parameters to create the dBASE file based on the contents of the field tag.
4. Free the field tag with `freeDBFieldTag()`.

Parameters

`<file name>`	The name of the dBASE file to create.
`<field tag number>`	The return value from the `createDBFieldTag()`, which must be called before calling `createDBFile()`.
`<preserve existing>`	If this parameter is 1, ToolBook will not create the new file nor delete the existing one if the dBASE file `<file name>` already exists. If this parameter is 0, ToolBook will delete the existing file and create a new one.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 27	The dBASE file could not be created.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Examples

```
get createDBFile(dbFileName,tagNumber,1)
if It <= 0
  request "Error creating file"
break
end if
step i from 1 to itemCount(fieldNames)
  put item i of fieldNames into It
  ask "What is field type for field " & It & "?"
  put It && ", " into item i of fieldType
  put item i of fieldNames into It
  ask "What is field width for field " & It & "?"
  put It && ", " into item i of fieldWidth
end step
```

createDBIndexFile( )

TB30DB3.DLL

Syntax `createDBIndexFile(<file name>,<sort expression>, \`
`   <unique key>,<preserve existing>)`

Declaration `INT createDBIndexFile(STRING,STRING,WORD,WORD)`

Description Creates and opens a dBASE index file with the specified name and sort expression, and defines whether each key in the index file must be unique and whether to delete an existing index file with the same name.

Parameters

`<file name>` The name of the index file to create.

`<sort expression>` The expression that defines the index file's sort criteria. The expression can include literal field names, constants, and operators. The resulting type of the sort expression must be numeric, logical, date, or character.

Sort expression syntax elements:

Type	Values	
Numeric operators	+ - * / ** ^	
Character operators	+	
Relational operators	= <> # < > <= >= $	
Logical operators	.NOT. .OR. .AND.	
Functions	CTODDATE DELETED DTOC IIF RECCOUNT	RECNO STR TIME UPPER VAL
Constants	.T.	.F.

`<unique key>` An integer that specifies whether each key in the index file must be unique. If `<unique key>` is 1, then the uniqueness of each key in the index file will be maintained. If `<unique key>` is 0, the keys will not be checked for uniqueness.

`<preserve existing>` An integer that specifies whether to preserve an existing index file. If `<preserve existing>` is 1, ToolBook will not create the new file nor delete the existing one. If `<preserve existing>` is 0, ToolBook will delete the existing file and create a new one.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The sort expression contains an error or the file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
--Makes index available for browsing records in order defined by
--sort expression
if createDBIndexFile("temp.ndx","LASTNAME + FIRSTN",1,0) <
   1 request getDBErrorString(It)
end
```

deleteDBFile( )

TB30DB3.DLL

Syntax	`deleteDBFile(<file name>)`
Declaration	`INT deleteDBFile(STRING)`
Description	Deletes the specified file and any associated index files currently open.
Parameter	`<file name>` The name of the dBASE file to delete.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 20	The function failed because of an out-of-memory condition.
- 28	No dBASE file with the specified name.
- 29	The index file could not be closed.

deselectDBIndexFile( )

TB30DB3.DLL

Syntax	`deselectDBIndexFile()`
Declaration	`INT deselectDBIndexFile()`
Description	Deselects the currently selected index file so it is no longer the current index file. The current index file determines the order in which to navigate in the current dBASE file. The index file must have been opened previously with `openDBIndexFile()`.
Parameters	None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

findDBKey( )

TB30DB3.DLL

Syntax	`findDBKey(<search string>)`
Declaration	`INT findDBKey(STRING)`
Description	Searches the current index file to find the closest key that matches the specified string. The resulting key becomes the current key and the record referenced by the key becomes the current record. When the file is already indexed, this is the fastest way to search for a record.
Parameter	`<search string>` The search string that ToolBook uses to search for a match in the index file.

Returns The return value indicates the result of the search:

1	Exact match found.
2	String found at beginning of longer string.
3	String not found, next key becomes current key.
4	String greater than last key, last key becomes current key.

If an error occurs, this function returns:

0	An internal error occurred.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 10	The record was marked as deleted.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

firstDBKey( )

TB30DB3.DLL

Syntax	`firstDBKey()`
Declaration	`INT firstDBKey()`
Description	Makes the first key in the current index file the current key and makes the record referenced by the first key the current record. If the navigate-to-deleted switch is off, this function skips records marked as deleted.
Parameters	None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	TThe file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.
- 15	The database is empty.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

firstDBRecord( )

TB30DB3.DLL

Syntax	`firstDBRecord()`
Declaration	`INT firstDBRecord()`
Description	Makes the first record in the current dBASE file the current record. If the navigate-to-deleted switch is off, this function skips records marked as deleted until an undeleted record is found.
Parameters	None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 10	No such record; the database is empty or all records are deleted and navigate-to-deleted is not in effect.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
--Does lead read before looping through records
set recordcount to
set It to null
get firstDBRecord()
if It <= 0
  send error It, "getting first record in dBASE file"
  break buttonClick
end if
```

freeDBFieldTag( )

TB30DB3.DLL

Syntax	`freeDBFieldTag(<field tag number>)`
Declaration	`INT freeDBFieldTag(WORD)`
Description	Frees memory used by the field tag created with `createDBFieldTag()`.
Parameter	`<field tag number>` The number returned when creating the tag with `createDBFieldTag()`.

Returns

1	The function was successful.
- 7	The field tag is invalid.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
--Gets return value for tag number
put createDBFieldTag(noFields) into tagNumber
if tagNumber = 0
  request "Error creating field tag"
  break
end if
step i from 1 to itemCount(fieldNames)
  get setDBFieldTag(tagNumber,i,item i of fieldNames, \
    item i of fieldType,item i of fieldWidth,0)
  if It <= 0
    request "Error setting field tag"
    break
  end if
end step
get createDBFile(dbFileName,itemCount(fieldNames), \
  tagNumber,1)
if It <= 0
  request "Error creating file"
  break
end if
get freeDBFieldTag(tagNumber)
if It <= 0
  request "Error freeing field tag"
  break
end if
```

getDBDateFormat( )

TB30DB3.DLL

Syntax	`getDBDateFormat()`
Declaration	`STRING getDBDateFormat()`
Description	Gets the date format currently in use by the DLL.
Parameters	None.
Returns	A string containing the current date format.

If no error occurs, `getDBDateFormat()` returns the name of the current index file. Otherwise, it returns null and `sysError` is set to one of these values:

- 3	Too many clients for this DLL or not enough memory.
- 20	Not enough memory to execute the function.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

getDBErrorString( )

TB30DB3.DLL

Syntax	`getDBErrorString(<error code>)`
Declaration	`STRING getDBErrorString(INT)`
Description	Returns a string describing the error.
Parameter	`<error code>` A numeric code returned by any function in the DLL.
Returns	If no error occurs, `getDBErrorString()` returns a string that describes the error. Otherwise, it returns null and `sysError` is set to one of these values:

- 8	The file, index, or system is corrupted.
- 20	The function failed due to an out-of-memory condition.
- 72	No string corresponds to that code.

getDBFieldCount( )

TB30DB3.DLL

Syntax `getDBFieldCount()`

Declaration `INT getDBFieldCount()`

Description Returns the number of fields in the current dBASE file.

Parameters None.

Returns If no error occurs, `getDBFieldCount()` returns the number of fields in the current dBASE file. Otherwise, it returns:

- 3	Too many clients for this DLL or not enough memory.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
if getDBFieldCount() < 1
  send error It, "getting count of fields in dBASE file"
end if
```

getDBFieldName( )

TB30DB3.DLL

Syntax	`getDBFieldName(<field position>)`
Declaration	`STRING getDBFieldName(WORD)`
Description	Returns the name of the field of the specified column number.
Parameter	`<field position>` The column number of the field in the current dBASE file.

Returns If no error occurs, `getDBFieldName()` returns the name of the field in the current dBASE file. Otherwise, it returns null and sysError is set to one of these values:

- 3	Too many clients for this DLL or not enough memory.
- 11	No such field.
- 12	No current dBASE file; use `openDBFile()` first.
- 20	The function failed due to an out-of-memory condition.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
--Loops through the fields in the dBASE file, building background
--of book
step i from 1 to numFields
  clear sysError
  get getDBFieldName(i)
  if sysError is not null
    send error sysError, "getting field name in dBASE file"
    break buttonClick
  end if
  put It & "," after fldnames              --Builds list of
                                           --field names for
                                           --this file
  draw recordField from x,y to 9180,y+360 --Draws and names
  set name of selection to It             --record field
  increment y by adv
end step
```

getDBFieldPrecision( )

TB30DB3.DLL

Syntax	`getDBFieldPrecision(<field name>)`
Declaration	`INT getDBFieldPrecision(STRING)`
Description	Returns the number of decimal places for the specified numeric field.
Parameter	`<field name>` The name of the numeric field.

Returns If no error occurs, `getDBFieldPrecision()` returns the number of decimal places for the specified numeric field. Otherwise, it returns:

- 3	Too many clients for this DLL or not enough memory.
- 4	The field name is invalid.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

getDBFieldType( )

TB30DB3.DLL

Syntax	`getDBFieldType(<field name>)`
Declaration	`INT getDBFieldType(STRING)`
Description	Returns the field type for a field name in the current dBASE file.
Parameter	`<field name>` The name of the field for which you want the field type.

Returns

If no error occurs, `getDBFieldType()` returns:

1	The field type is character.
2	The field type is logical.
3	The field type is date.
4	The field type is numeric.
5	The field type is memo.

If an error occurs, this function returns:

- 3	Too many clients for this DLL or not enough memory.
- 4	The field name is invalid.
- 6	The field type is invalid.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

getDBFieldValue( )

TB30DB3.DLL

Syntax `getDBFieldValue(<field name>)`

Declaration `STRING getDBFieldValue(STRING)`

Description Returns the value of the specified field for the current dBASE record and file. This is the value as it exists in the record buffer; in other words, if you made changes to the field with `setDBFieldValue()`, but did not use `writeDBRecord()`, this function returns the value stored in the buffer, not the value in the file.

Parameter `<field name>` The name of the field whose value you want.

Returns If no error occurs, `getDBFieldValue()` returns the value of the specfied field. Otherwise, it returns null and `sysError` is set to one of these values:

- 3	Too many clients for this DLL or not enough memory.
- 4	The field name is invalid.
- 6	The field type is invalid.
- 12	No current dBASE file; use `openDBFile()` first.
- 20	The function failed because of an out-of-memory condition.
- 23	An error occurred while reading a memo file.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
while true
   increment recordCount
   step i from 1 to totalField
      set sysError to null
      get getDBFieldValue(item i of fieldNames)
      if sysError is not null
         send error sysError, "getting value of field."
         break buttonClick
      end if
      put It into text of recordField (item i of fieldNames)
   end step
   get nextDBRecord()
   if It <= 0
      break while
   end if
   send newPage
end while
```

getDBFieldWidth( )

TB30DB3.DLL

Syntax	`getDBFieldWidth(<field name>)`
Declaration	`INT getDBFieldWidth(STRING)`
Description	Returns the width of the specified field in the current dBASE record and file.
Parameter	`<field name>` The name of the field whose width you want.
Returns	If no error occurs, `getDBFieldWidth()` returns the width in characters for the current dBASE record and file. Otherwise, it returns:

- 3	Too many clients for this DLL or not enough memory.
- 4	The field name is invalid.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

getDBFileName( )
TB30DB3.DLL

Syntax	`getDBFileName()`
Declaration	`STRING getDBFileName()`
Description	Returns the name of the current dBASE file.
Parameters	None.
Returns	If no error occurs, `getDBFileName()` returns the path and file name of the current dBASE file. Otherwise, it returns null and `sysError` is set to one of these values:

- 3	Too many clients for this DLL or not enough memory.
- 12	No current dBASE file; use `openDBFile()` first.
- 20	The function failed because of an out-of-memory condition.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

getDBIndexExpression( )

TB30DB3.DLL

Syntax	`getDBIndexExpression()`
Declaration	`STRING getDBIndexExpression()`
Description	Returns the expression used to form the keys of the current index file for the current dBASE file.
Parameters	None.
Returns	If no error occurs, `getDBIndexExpression()` returns the expression used to form the keys of the current index file for the current dBASE file. Otherwise, it returns null and `sysError` is set to one of these values:

- 3	Too many clients for this DLL or not enough memory.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.
- 20	The function failed because of an out-of-memory condition.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

getDBIndexFileName( )

TB30DB3.DLL

Syntax	`getDBIndexFileName()`
Declaration	`STRING getDBIndexFileName()`
Description	Returns the name of the currently selected index file.
Parameters	None.
Returns	If no error occurs, `getDBIndexFileName()` returns the name of the current index file. Otherwise, it returns null and `sysError` is set to one of these values:

- 3	Too many clients for this DLL or not enough memory.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.
- 20	The function failed because of an out-of-memory condition.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

getDBKeyType( )

TB30DB3.DLL

Syntax	`getDBKeyType()`
Declaration	`INT getDBKeyType()`
Description	Returns the type of the current key in the current index file.
Parameters	None.

Returns If no error occurs, `getDBKeyType()` returns one of three possible values for the type of the current key:

1	The current key type is character.
3	The current key type is date.
4	The current key type is numeric.

Otherwise, it returns null and `sysError` is set to one of these values:

- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.

For other negative return values, use `getDBErrorString()`.

getDBKeyValue( )

TB30DB3.DLL

Syntax	`getDBKeyValue()`
Declaration	`STRING getDBKeyValue()`
Description	Returns the value of the current key for the current index file.
Parameters	None.
Returns	If no error occurs, `getDBKeyValue()` returns the value of the current key for the current index file. Otherwise, it returns null and `sysError` is set to one of these values:

- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.
- 14	No current key; use a function such as `firstDBKey()` first.
- 20	The function failed because of an out-of-memory condition.

For other negative return values, use `getDBErrorString()`.

getDBNavigateToDeleted( )

TB30DB3.DLL

Syntax	`getDBNavigateToDeleted()`
Declaration	`INT getDBNavigateToDeleted()`
Description	Returns the state of the switch that controls navigation to records marked as deleted. This switch is off by default, which means that normal navigation from record to record will skip over the deleted records. The records are not actually removed from the file until it is compacted with `packDBFile()`. To set the switch, use `setDBNavigateToDeleted()`.
Parameters	None.

Returns

0	Navigate-to-deleted switch is off.
1	Navigate-to-deleted switch is on.
- 3	Too many clients for this DLL or not enough memory.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
request "Allow navigation to deleted records?" with "No" or "Yes"
if It is "Yes"
  get setDBNavigateToDeleted(1)
else
  get setDBNavigateToDeleted(0)
end
```

getDBRecordCount( )

TB30DB3.DLL

Syntax `getDBRecordCount()`

Declaration `LONG getDBRecordCount()`

Description Returns the number of records in the current dBASE file.

Parameters None.

Returns If no error occurs, `getDBRecordCount()` returns the number of records in the current dBASE file. Otherwise, it returns:

- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
to handle recordCount
   get getDBRecordCount()
   if It <= 0
      send error It, "getting record count in dBASE file"
      break recordCount
   end if
   put It into totalRecords
   while true
      increment recordCount
      step i from 1 to totalFields
         set sysError to null
         get getDBFieldValue(item i of fieldNames)
         if sysError is not null
            send error sysError, "getting value of field"
            break recordCount
         end if
         put It into text of recordField(item i of fieldNames)
      end step
      set It to null              --Reads the next record
      get nextDBRecord()
      if It <= 0
         if recordCount < totalRecords
            request "Only processed " & recordCount & \
               " out of " & totalRecords
         end if
         break while
      end if
   end while
end recordCount
```

getDBRecordDeleted( )

TB30DB3.DLL

Syntax	`getDBRecordDeleted()`
Declaration	`INT getDBRecordDeleted()`
Description	Determines if the current record is marked for deletion.
Parameters	None.

Returns

1	The current record is marked for deletion.
0	The current record is not marked for deletion.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 32	No current record.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

getDBRecordNumber( )

TB30DB3.DLL

Syntax	`getDBRecordNumber()`
Declaration	`LONG getDBRecordNumber()`
Description	Returns the number of the current record.
Parameters	None.
Returns	If no error occurs, `getDBRecordNumber()` returns the number of the current record. Otherwise, it returns:

0	No current record.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.

gotoDBRecord( )

TB30DB3.DLL

Syntax	`gotoDBRecord(<record number>)`
Declaration	`INT gotoDBRecord(DWORD)`
Description	Navigates to the specified record number in the data file and makes it the current record.
Parameter	`<record number>` The number of the record you want to make current.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 10	No such record.
- 12	No current dBASE file; use `openDBFile()` first.
- 87	The record was marked as deleted and navigate-to-deleted is off; see `getDBNavigateToDeleted()`.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

lastDBKey( )

TB30DB3.DLL

Syntax	`lastDBKey()`
Declaration	`INT lastDBKey()`
Description	Makes the last key in the current index file the current key and makes the record referenced by the last key the current record. Unless the navigate-to-deleted switch is on, this function skips records marked as deleted.
Parameters	None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.
- 15	The database is empty.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

lastDBRecord( )

TB30DB3.DLL

Syntax	`lastDBRecord()`
Declaration	`INT lastDBRecord()`
Description	Makes the last record in the current dBASE file the current record. If the last record is marked as deleted and the navigate-to-deleted switch is off, the last undeleted record becomes the current record.
Parameters	None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 10	No such record; the database is empty or all records are deleted and navigate-to-deleted is off.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

nextDBRecord( )

TB30DB3.DLL

Syntax `nextDBRecord()`

Declaration `INT nextDBRecord()`

Description Makes the next record in the current dBASE file the current record. Unless the navigate-to-deleted switch is on, this function skips records marked as deleted.

Parameters None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 10	No such record (current record is last record).
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
get nextDBRecord()
if It <= 0
   if recordCount < totalRecords
      request "Only processed" && recordCount &&\
         "out of" && totalRecords
   end
end
```

openDBFile( )

TB30DB3.DLL

Syntax	`openDBFile(<file name>)`
Declaration	`INT openDBFile(STRING)`
Description	Opens and initializes the specified file and makes it the current dBASE file.
Parameter	`<file name>` The name of the dBASE file to open.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 36	The file could not be opened.
- 88	A memo field exists but the memo file could not be opened.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
ask "Name of dBASE file to open?"
put It into text of field "dBASEFileName"
get openDBFile(text of field "dBASEFileName")
if It <> 1
  send error It, "Opening dBASE file"
  break
end if
```

openDBIndexFile( )

TB30DB3.DLL

Syntax `openDBIndexFile(<file name>)`

Declaration `INT openDBIndexFile(STRING)`

Description Opens the specified index file and makes it the current index file for the current dBASE file. The first logical record in the order defined by the index becomes the current record.

Parameter `<file name>` The name of the index file to open for the current dBASE file.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

previousDBKey( )

TB30DB3.DLL

Syntax	`previousDBKey()`
Declaration	`INT previousDBKey()`
Description	Makes the key before the current key in the current index file the current key. Unless the navigate-to-deleted switch is on, this function skips records marked as deleted.
Parameters	None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.
- 71	The key was already the first key.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

previousDBRecord( )

TB30DB3.DLL

Syntax `previousDBRecord()`

Declaration `INT previousDBRecord()`

Description Makes the record before the current record in the current dBASE file the current record. If the current record is the first record, then it will remain the current record. Unless the navigate-to-deleted switch is on, this function skips records marked as deleted.

Parameters None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 10	No such record; the current record is the first record.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
--Backs up the record pointer
step i from 1 to 100
   get previousDBRecord()
end step
send displayRecord
```

reindexDBFile( )

TB30DB3.DLL

Syntax	`reindexDBFile(<file name>)`
Declaration	`INT reindexDBFile(STRING)`
Description	Reindexes the specified index file. The data file associated with the index file must be open before calling the function. On completion of reindexing, the specified index file becomes the current index.
Parameter	`<file name>` The name of the index file to reindex.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 53	No index file with the specified name.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

removeDBRecords( )

TB30DB3.DLL

Syntax	`removeDBRecords(<start number>,<end number>)`	
Declaration	`INT removeDBRecords(DWORD,DWORD)`	
Description	Removes from the current dBASE file the records in the specified range, inclusive.	
Parameters	`<start number>`	The beginning of the range of records to delete.
	`<end number>`	The end of the range of the records to delete. To delete only one record, specify that record number for both `<start number>` and `<end number>`.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

nextDBKey( )

TB30DB3.DLL

Syntax	`nextDBKey()`
Declaration	`INT nextDBKey()`
Description	Makes the next key in the current index file the current key and makes the record referenced by that key the current record. If the current key is the last key, then it will remain the current key. Unless the navigate-to-deleted switch is on, this function skips records marked as deleted.
Parameters	None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 13	No current index file; use `openDBIndexFile()` first.
- 70	The key is the last key.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

selectDBFile( )

TB30DB3.DLL

Syntax	`selectDBFile(<file name>)`
Declaration	`INT selectDBFile(STRING)`
Description	Makes the specified file the current dBASE file. This function opens the file if it is not already open. If the dBASE file is already open and has an associated index file, this file becomes the current index file.
Parameter	`<file name>` The name of the file to be the current dBASE file.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 28	No dBASE file with the specified name.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

selectDBIndexFile( )

TB30DB3.DLL

Syntax	`selectDBIndexFile(<file name>)`
Declaration	`INT selectDBIndexFile(STRING)`
Description	Makes the specified index file the current index file and uses it to navigate in the current dBASE file.
Parameter	`<file name>` The name of the index file to select.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 53	No index file with the specified name.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

packDBFile( )

TB30DB3.DLL

Syntax	`packDBFile()`
Declaration	`INT packDBFile()`
Description	Compacts the current dBASE file by reclaiming space occupied by records marked for deletion. After the file is compacted, the current record is the last record in the data file.
Parameters	None.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 12	No current dBASE file; `use openDBFile()` first.
- 31	The dBASE file may be corrupted.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

writeDBRecord( )
TB30DB3.DLL

Syntax	`writeDBRecord(<record number>)`
Declaration	`INT writeDBRecord(DWORD)`
Description	Writes the contents of the record buffer into the specified record and updates all open index files. The contents of the record buffer is set with `setDBFieldValue()`.

Parameter

`<record number>`	The number of the record where you want to write the contents of the record buffer. If `<record number>` is 0, then the current record is updated with the contents of the record buffer. If `<record number>` indicates a higher record number than is in the dBASE file, ToolBook appends a new record that contains the contents of the record buffer. If `<record number>` specifies a record with contents, then `writeDBRecord()` overwrites the contents of that record.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 9	The function failed because of a duplicate key.
- 12	No current dBASE file.
- 79	The record was marked as deleted: write operation was denied.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
get writeDBRecord(totRecs + 1)
if It <> 1
   send error It, "writing dBASE record"
   break
end if
increment totRecs
```

setDBDateFormat( )

TB30DB3.DLL

Syntax	`setDBDateFormat(<format string>)`	
Declaration	`INT setDBDateFormat(STRING)`	
Description	Sets the date format to use in transactions through the DLL.	
Parameter	`<format string>`	Any string that specifies a valid dBASE date format. The date format specified is a visual representation of the date. For Example: YY.MM.DD CCYY.MM.DD MM/DD/YY DD-MM/YY DD-MM/CCYY MMM DD/YY Y, M, and D specify the year, month, and day, respectively. For more than two M characters, a letter representation of the month is used. The default date format is MM/DD/YY (month/day/year).

Returns

1	The function was successful (even if the date format string itself was invalid).
- 3	Too many clients for this DLL or not enough memory.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

setDBFieldTag( )

TB30DB3.DLL

Syntax
```
setDBFieldTag(<field tag number>,<field item>,<field name>, \
   <field type>,<field width>,<field decimals>)
```

Declaration `INT setDBFieldTag(WORD,WORD,STRING,STRING,WORD,WORD)`

Description Sets the specified field in the specified field tag to the specified field name, type, width, and decimal precision. You can use this function to specify information for more than one field by calling it again for different fields with the same tag number.

Parameters

`<field tag number>` The number returned by `createDBFieldTag()`.

`<field item>` The number of the field to set. For details about setting the number of fields in a field tag, see `createDBFieldTag()`.

`<field name>` The name of the field. The name cannot be more than ten characters.

`<field type>` The field type, which can be one of the following:

Value	dBASE field type
"1", "c", or "C"	Character
"2", "l", or "L"	Logical
"3", "d", or "D"	Date
"4","n", or "N"	Numeric
"5", "m", or "N"	Memo

If `<field type>` evaluates to character, you must specify a field width with `<field width>`. If `<field type>` evaluates to date or memo, you do not need to specify a field width. If `<field type>` evaluates to numeric, you must specify a field width and the decimal precision with `<field decimals>`.

`<field width>` An integer indicating the width of the field. If this parameter is <= 0 for a character field, then `<field width>` defaults to 254. If this parameter is <= 0 for a numeric field, then `<field width>` defaults to 10. The maximum width for a numeric field is 19. If the field type is memo, date or logical, this parameter is ignored.

`<field decimals>` The number of decimal places in a numeric field. It is ignored for other field types. If `<field decimals>` is 0 for a numeric field, then the contents of the field is treated as an integer. If it is more than 0, it must be smaller than or equal to `<field width>` - 2. If `<field decimals>` is negative, larger than 15, or larger than `<field width>` - 2, the function returns an error.

Returns

1	The function was successful.
- 6	The field type is invalid.
- 7	The field tag is invalid.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example
```
step i from 1 to itemCount(fieldNames)
   get setDBFieldTag(tagNumber,i,item i of fieldNames, \
      item i of fieldType,item i of fieldWidth,0)
   if It <= 0
   request "Error setting field tag"
      break
   end if
end step
```

setDBFieldValue( )

TB30DB3.DLL

Syntax `setDBFieldValue(<field name>,<new value>)`

Declaration `INT setDBFieldValue(STRING,STRING)`

Description Sets the contents of the specified field in the current record to the specified new value. Changes to field values made by this function are stored in the record buffer until you call `writeDBRecord()`.

Parameters

`<field name>`	The name of the field whose value you want to change.
`<new value>`	The new value for the specified field.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 4	The field name is invalid.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 26	A memo field could not be written.
- 32	No current record.
- 60	The date is invalid for a date field.
- 62	The data is invalid for the field type.
- 73	The text is too long to fit in the field.

For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Example

```
if setDBFieldValue(textline j of text of field "sname",1) <> 1
  increment totInval
end if
```

setDBNavigateToDeleted( )

TB30DB3.DLL

Syntax	`setDBNavigateToDeleted(<option>)`
Declaration	`INT setDBNavigateToDeleted(INT)`
Description	Sets a switch to allow navigation to records marked for deletion.
Parameter	`<option>` 0 to turn navigate-to-deleted off; 1 to turn it on. The default is off.
Returns	1 The function was successful. - 3 Too many clients for this DLL or not enough memory. For other negative return values, use `getDBErrorString()` to get an explanation of the error.

setDBRecordDeleted( )

TB30DB3.DLL

Syntax	`setDBRecordDeleted(<delete value>)`	
Declaration	`INT setDBRecordDeleted(WORD)`	
Description	Marks the current record for deletion.	
Parameter	`<delete value>`	Determines whether the current record is marked for deletion. If this value is >= 1, then the record is marked for deletion. If the value is <= 0, then this function removes the mark that indicates the record is to be deleted.

Returns

1	The function was successful.
- 3	Too many clients for this DLL or not enough memory.
- 8	The file, index, or system is corrupted.
- 12	No current dBASE file; use `openDBFile()` first.
- 32	No current record; use a function such as `firstDBRecord()` first.

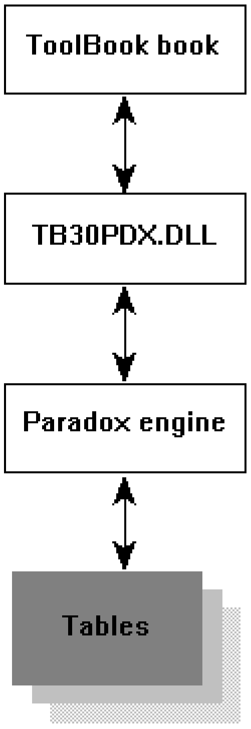
For other negative return values, use `getDBErrorString()` to get an explanation of the error.

Using the Paradox Engine

Paradox overview

See also...

The Paradox Engine is a set of functions that perform the data-handling operations -- for example, creating, opening, and closing tables and reading and writing data -- of the Paradox relational database manager. ToolBook's TB30PDX.DLL contains functions that enable you to control the Paradox Engine to perform database operations. You can work with tables created by the Paradox database manager, or you can create your own tables.



Data Storage

Data is stored in tables, with one table to a file. A table contains one or more records. Each record contains one or more fields. A field can contain alphabetic or numeric data.

Binary Large Objects (BLOBs)

A BLOB is any block of binary data (such as a graphic, video clip, or sound) or a block of text data that is larger than 255 bytes. BLOBs are stored in a separate file; the BLOB's handle is stored in a field in the table.

Indexes

Tables can be indexed by any field or combination of fields, other than BLOBs. You can not index on a BLOB field. You can have any number of indexes for a table (but you can only use one at a time).

Moving data

Logically, data moves to and from the table by way of a transfer buffer. TB30PDX.DLL functions that read and write data take care of moving data into and out of the buffer, so you never have to deal with it directly. To write a value to a field, go to the record where you want to put the record, set the value for the field in the record, and then update the record in the database table.

Physically, data moves in blocks by way of a swap buffer (cache). The engine takes care of all physical data movement automatically, although you can control the operations yourself.

Specifications
Paradox overview

Maximum table size	256MB
Maximum records per table	As many as will fit in 256MB
Maximum bytes per record	4000 for non-keyed (non-indexed) tables
	1350 for keyed (indexed) tables
Maximum fields per record	255
Maximum field size in table	255 characters
Maximum size of a BLOB	256MB
Maximum size of the BLOB file	256MB
Maximum fields in a composite index	16
Maximum passwords	50
Maximum password size	15 characters
Maximum field name size	25 characters
Maximum table name size	128 characters

Naming rules
Paradox overview

Tables
The base database table is a DOS file with a .DB extension; its name must follow standard DOS naming conventions.

If index and BLOB files are associated with the base table, they are given the same name with different extensions. BLOB files have the .MB extension.

Primary indexes have a .PX extension. Secondary indexes have a .Xnn or .Ynn extension, where nn is a number supplied by the engine.

Fields
- Maximum 25 characters
- May contain spaces but may not begin with a space
- May contain any printable character except double quotes ("), brackets ([ ]), braces ({ }), number signs (#), left or right parentheses ( ), or the character combination ->
- May not duplicate another field name in the same table

Data specifications

Paradox overview

Five field types

Alphanumeric (A)	Full range of ASCII characters except embedded null (ASCII 0). Maximum length 255 characters Each character requires one byte
Numeric (N)	15 significant digits including the decimal point. Numbers with more than 15 significant digits are rounded and stored in scientific notation. Real numbers from ±10 ^ -307 to ±10 ^ 308 Requires 8 bytes per field
Currency ($)	15 significant digits including the decimal point Real numbers from ±10 ^ -307 to ±10 ^ 308 Requires 8 bytes Formatted according to the currency format selected in the Windows Control Panel
Short number (S)	Signed integers between -32,767 and 32,767 Requires 2 bytes
Date (D)	Any valid date between January 1, 1 AD to December 31, 9999 Paradox stores the data as a 4-byte integer equal to the number of days since January 1, 1 AD.

Three BLOB types

Memo (M)	Unformatted text
Binary (B)	Unstructured binary (arbitrary data such as bit patterns and sound)
Graphic (G)	Windows .DIB bitmaps

Initialization

Paradox overview

See also...

Before you can use the Paradox Engine, you have to initialize it by calling `initializePX()`. Calling `initializePX()` sets the following parameters to their default values. If you want to set any of these parameters to a different value, call the function listed below, then call `initializePX()`.

Parameter	Description	Default value (range)	To change, call
Swap buffer size	Amount of memory available for swapping data in and out of memory. Each table requires 4K. Allocate more to reduce disk accesses, but too large an allocation could cause program to run out of memory.	32K (8K - 256K)	`setPXINISwapSize()`
Max file handles	Maximum number of file handles available to the engine.	10 (3 - 255)	`getPXMaxFiles()`
Max tables	Maximum number of tables that can be open at one time	5 (1 - 64)	`setPXINIMaxTables()`
Max locks	Maximum number of locks per table	32 (1 - 128)	`setPXINIMaxLocks()`
Sort order	The type of sort to use	ASCII (see function description)	`setPXSortOrder()`

Maximum table and block size

Paradox overview

The maximum table size is the maximum amount of disk space a table can occupy. The block size is the amount of data that is moved into and out of the swap buffer at one time. (The buffer size can be from 8 to 256KB; the default size is 32KB.)

The block size is determined by the maximum table size. The defaults are 2KB blocks and 128MB maximum table size. You can call `setPXTableMaxSize()` to select one of these combinations:

Maximum table size	Block size
64MB	1KB
128MB	2KB
256MB	4KB

Working with tables

Paradox overview

See also...

Creating tables

To create a table, call the function `createPXTable()` and pass it the table name, a list of 1 to 255 field names, and a list of corresponding field types and sizes.

Warning

If a table with the same name already exists in the same directory and is not open, it (and all associated BLOB and index files) are deleted without warning. Before creating a table, use `doesPXTableExist()` to check for the existence of a table with the same name.

By default, the engine creates a Paradox 4.0-format table. If you want to create a table in Paradox 3.5 format, call `setPXTableCreateMode()` first. The argument specifies the type of format to use. All subsequent tables created by `createPXTable()` will be in the format specified until you call `setPXTableCreateMode()` again to switch modes. (When you initialize the engine, the create mode is reset to the default.)

You can set a maximum size for a table by calling `setPXTableMaxSize()` before calling `createPXTable()`. All subsequent tables created by `createPXTable()` are limited to that size until you call `setPXTableCreateMode()` again to change the limit.

After you create a table, you must open it before you can use it.

Opening a table

Paradox overview

Call `openPXTable()` to open a table. The arguments for the function include:

- An alias to assign to the table to be used for subsequent access
- The file name of the table (without the extension)
- The handle (number) of the index to use to order records. Use 0 if you haven't created an index for the table
- The mode for saving changes

If the table has an index, the records are presented in the order of the index. If not, they are presented in the order in which they were entered into the table. You can open the same table multiple times with different indexes to get different views of the records.

Closing a table

Paradox overview

Call `closePXTable()` to close a table. Specify the alias of the table you want to close.

If you don't close all tables before quitting the application, the DLL closes the tables for you, but you lose all data that has not been posted to the tables.

Deleting a table

Paradox overview

To delete a table, call `deletePXTable()` with the file name of the table you want to delete. The table must be closed. If there are index files or a BLOB file associated with the table, they are automatically deleted as well.

The contents of the table and related BLOB and index files are physically erased from the disk.

Other table operations

Paradox overview

TB30PDX.DLL functions can be used to perform a variety of operations on tables. For each of these functions, the tables must be referred to by name, not by alias.

`copyPXTable()`	Makes a copy of a table family (the base table plus associated BLOB and index files) with another name. Composite indexes are not copied. If a table by that name already exists and is closed, it is overwritten.
`renamePXTable()`	Changes the name of the base table and any associated BLOB and index files
`addPXTable()`	Adds records from one table to another. Data types for corresponding fields in the two tables must match; BLOB fields must match in both type and length of leader.
`emptyPXTable()`	Removes all records from a table, physically deleting them from the disk, but leaves the table structure intact. Private BLOBs in associated record buffers are released, and pending `dropPXBlob()` operations are ignored.

Table information operations

Paradox overview

TB30PDX.DLL functions can be used to obtain information about tables.

`doesPXTableExist()`	Does the specified table exist? (Refer to the table by name, not alias.)
`isPXTableProtected()`	Is the specified table encrypted? (Refer to the table by alias.)
`isPXNetTableChanged()`	Has the table been changed since the last time you called `refreshPXNetTable()`? (Refer to the table by alias.)
`getPXRecordCount()`	How many records are in the table? (Refer to the table by alias.)
`getPXFieldCount()`	How many fields are in the table? (Refer to the table by alias.)
`getPXFieldType()`	What is the type of a field? Also returns the length of an alphanumeric field and the width of a BLOB field. (Refer to the table by alias and the field by name.)
`getPXFieldNames()`	What are the names of the fields in the table? (Refer to the table by alias.)
`getPXKeyFieldCount()`	How many key fields are in the table? (Refer to the table by alias.)

Working with records

Paradox overview

Moving to a record

Functions for moving to specific records include:

gotoPXRecord()	Go to the record specified by index number. (This method is unreliable when the table is shared on a network, because records can be added or deleted without your knowledge, thereby changing record numbers.)
firstPXRecord()	Go to the first record in the table or index.
lastPXRecord()	Go to the last record in the table or index.
nextPXRecord()	Go to the record after the current one in the table or index.
previousPXRecord()	Go to the record before the current one in the table or index.

The result you get from using one of these functions to move to a particular record depends on whether the table has an index and, if it does, on the index you assigned when you opened the table.

If the table has no index, records are accessed in the order in which they were entered into the table. The first physical record is record number 1 (not 0), the next is record 2, and so on. "First record" is the first physical record in the table; "next record" is the next physical record after the current one.

If you assign an index to the table when you open it, records are referred to according to their order in the index, which may not be the same as their physical order in the table. Thus, record number 1 is the first record in the index, number 2 is the next record in the index, and so on.   "First record" is the first record in the index; "next record" is the next record in the index after the current one.

If you have locked a record, you can use gotoPXNetRecordLock() to go to the locked record.

Adding a record
Deleting records
Getting the number of the current record
Clearing a record
Record functions

Adding a record

Paradox overview

To add a record to a table:

1. Use `setPXFieldValue()` to set values in the record fields.
2. Call `appendPXRecord()` or `insertPXRecord()` to add the record to the table.
   If the table is indexed, the record is added in its proper position in the index.
   If the table is not indexed, `appendPXRecord()` adds the record to the end of the table, while `insertPXRecord()` inserts the record before the current record.

Deleting records

Paradox overview

To delete a record:

1. If the table is not already open, open it and assign it an alias.
2. Find the record you want to delete.

   Go to a particular record or search for the contents of a key field in an indexed table
3. Call `deletePXRecord()` to delete the record.

   This function deletes one record at a time. To delete additional records, go to the record and call `deletePXRecord()` again for each record.

To delete all records:

With the table closed, call `emptyPXTable()`.

Getting the number of the current record

Paradox overview

To get the number of the current record:

1. Call `getPXRecordNumber()`.

   This function returns the number of the record in the index that is currently open for the table. If you change indexes for a table, the records could have different numbers. There is no way to get the physical record number.

   The record number is not a reliable way to find a record in shared tables, because other users can add or delete records without your knowledge, thereby changing the numbers of some or all of the records.

Clearing a record

Paradox overview

Clearing a record means deleting the data in the fields without deleting the record from the table.

To clear a record:

1. Move to the record you want to clear.
2. Call `emptyPXRecord()`.

Working with fields

Paradox overview

Getting information about a field

There are two functions that return information about fields:

- getPXFieldNames() returns a list of field names in the specified table. The table must be open.
- getPXFieldType() returns the type of the named field in the current record, as well as the length of an alphanumeric field and the width of a BLOB field. To get the types of all fields in a record, use the getPXFieldType() function in a loop, as shown in the example.

Example

```
to handle buttonClick
-- We eliminate blobs to let the user choose a field to index the table on
   local temp
   fieldNames = getPXfieldNames (DBTable of this book)
   step i from 1 to itemCount(fieldNames)
      testName = item i of fieldNames
      get PXFieldType(testName)
      if first char of it is in "MGBFO"
         continue step
      end if
      put testName & "," after temp
   end step
   clear last char of temp
-- lists all the fields that can have  an index
   request temp
end buttonClick
```

Reading field values
Writing field values
Field functions

Reading field values

Paradox overview

To read a field value:

1. Move to the record you want to read.
2. Call `getPXFieldValue()` once for each field you want to read in the current record.

   This function can be used with any type of data except BLOBs.

Writing field values

Paradox overview

To write a field value:

1. Move to the record you want to write.
2. Call `setPXFieldValue()` to change the value of a specific field in the current record.

   This function can be used with any type of data except BLOBs.

When you're finished writing field values, call `updatePXRecord()` to write the changes to the table.

Indexing and searching

Paradox overview

Using indexes

An index to a table is like the thumb index to a dictionary: It helps you find information quickly. When you want to look up a word in a dictionary, the thumb index narrows your search to words starting with the same letter. In the same way, when the engine needs to find data in an indexed table, it can go directly to the part of the table that contains the information before it starts to search each record in detail.

If the table had no index, the engine would have to search each record, starting with the first one, every time it wanted to find something.

When the engine sorts records according to an index, the sorting is done in memory. The engine does not physically rearrange the records on the disk.

Kinds of indexes

Paradox overview

The engine provides two main kinds of indexes: primary and secondary.

A primary index is always the first one or more contiguous fields in a record. These fields, taken together, are the key for the record. Duplicate key values are not allowed.

Each table must have a primary index. If you don't create one, the engine will create one for you.

Secondary indexes are optional. You can use a secondary index to present the records in a different order from the primary index. Secondary indexes can use any single field or any group of fields.

BLOBs cannot be used in either kind of index.

Primary indexes are always maintained automatically. Whenever you change the data in a table, the engine automatically checks for key violations, sorts the table in key-field order, and updates the index.
A single-field secondary index can be case-sensitive or case-insensitive.
Secondary indexes can be automatically maintained or not, as you choose.
You can have only one primary index for a table, but you can have as many secondary indexes as you want. You can have only one index open at a time for each table alias. (However, you can have the same table open more than once under different aliases, and each could have a different index.)

Creating indexes

Paradox overview

To create a primary index:

1. Open the table.
2. Call `addPXKey()`.

   Pass the the numbers of the fields you want to have the index comprise and the code for the type of index you want.

To create a secondary index:

1. Open the table.
2. Call `mapPXKey()` with the numbers and names of the fields you want to use for the index.
3. Call `addPXKey()` to create the index.

Maintaining indexes

Paradox overview

Indexes are maintained automatically. There's nothing you need to do to maintain them.

Deleting indexes

Paradox overview

Call `dropPXKey()` to delete an index. The table must be open when you delete the index. If you delete a primary index, all secondary indexes for the same table are also deleted.

Sort order

Paradox overview

To set the sort order for a table, call `setPXSortOrder()` before you call `initializePX()`. (If you don't call `setPXSortOrder()`, ASCII sort order is used by default.)

You can choose from these sort orders:

- ASCII sort order
- International sort order
- Norwegian/Danish sort order
- Swedish/Finnish sort order
- Norwegian/Danish sort order for Paradox 4.0

How searching works

Paradox overview

You can search for a match on a key field in the primary index or in a secondary index. When you're using a primary index that has more than one field in its key, you can specify how many of those fields (starting with the first) you want to include in the search.

You can search only for exact matches (although you can have the engine return the nearest non-matching record). Boolean operators -- such as greater than, less than, or not equal to -- are not available. You cannot search for BLOBs.

You can specify one of three types of search modes:

Mode 0	-- Search for an exact match, starting at the beginning of the table.
Mode 1	-- Search for an exact match, starting at the next record after the current one.
Mode 2	-- Search for the nearest match, starting at the beginning of the table.

In all cases, the search proceeds downward in the table.

In mode 0 or mode 1, if the engine finds an exact match, it makes that record the current record. If it does not find an exact match, it does not change the current record.

In mode 2, the engine finds the first record that contains a value equal to or greater than the value you're searching for. For example, suppose a series of records contain the following values in the first field:

Harriott

Harry

Horrace

Now suppose you search for "Herman" in mode 2. "Horrace" is the first value equal to or greater than the search value, so the engine makes that record the current record. If the engine doesn't find a value equal to or greater than the value you're searching for, it makes the last record in the table the current record.

Remember that the order of records is determined by the index in use at the moment. If you perform the same search with a different index, you might get different results.

Performing a search

Paradox overview

Performing a search is a two-step operation:

1. Put the value you want to match into the appropriate field of `setPXFieldValue()`.
2. Call `searchPXKey()` or `searchPXField()` to search for a match to the value.

Example

```
get setPXFieldValue("myDatabase", "name", text of field "name")
get searchPXField("myDatabase", "name", "nameIndex", 2)
if it < 0
   request getPXErrorString(it)
end
```

Working with BLOBs

Paradox overview

BLOBs (binary large objects) are binary objects of any size (such as a video clip, sound file, or bitmap) or blocks of text larger than 255 bytes. They can be any size up to 256MB.

A table can have up to 255 BLOB fields.

BLOBs are not stored in the table but in a separate file. The BLOB field in the table contains a pointer to the BLOB data in the separate file.

The BLOB field can also have a leader of up to 240 bytes, which stores a copy of that many bytes from the beginning of the BLOB. If the BLOB is smaller than the size of the leader, the entire BLOB can be stored in the table.

TB30PDX.DLL supports three kinds of BLOBs:

M -- memo (unformatted text)

B -- unstructured binary (arbitrary data such as bit patterns and sound)

G -- structured graphics (arbitrary data with a header defining structure)

When you call `createPXTable()` to create a table, you specify the type of BLOB and the size of the BLOB field leader (if any) in bytes in the list of field types.

Public and private BLOBs

Paradox overview

Public BLOBs are those that can be read by anyone who has access to the database table with which they are associated. A private BLOB is a local copy of a public BLOB or a new BLOB that has not been written to the table for the first time. A private BLOB can be seen only by the user who opened it. When a private BLOB is posted to a table, it becomes a public BLOB.

TB30PDX.DLL and the engine take care of managing public and private BLOBs for you.

Cloning BLOBs

Reading a memo BLOB

Paradox overview

To read a memo BLOB:

1. Go to the record that contains the reference to the BLOB.
2. Use `openPXBlobRead()` to open the BLOB to read.

   The engine returns the handle to the BLOB in the BLOB file.
3. Use the `getPXMemoBlob()` function to set a variable equal to the BLOB.
4. Use `closePXBlob()` to close the BLOB.
5. Display the variable with any suitable ToolBook technique.

Example

```
to handle buttonClick
   hPrivateBlob = openPXBlobRead(DBTable, fieldName)
   if hPrivateBlob < 0
      send PXError hPrivateBlob
   else
      request getPXMemoBlob(hPrivateBlob)
      get closePXBlob(hPrivateBlob, 0)
   end
end
```

Reading a binary BLOB

Paradox overview

To read a binary BLOB:

1. Go to the record that contains the reference to the BLOB.
2. Use `openPXBlobRead()` to open the BLOB to read.

   The engine returns the handle to the BLOB in the BLOB file.
3. Use the `getPXBlob()` function to read the BLOB into memory.

   The function returns the Windows handle to the BLOB.
4. Use `closePXBlob()` to close the BLOB.
5. Use the kernel `globalLock()` function to get a pointer to the BLOB in memory.
6. Set a variable equal to the contents of memory at the pointer location.
7. Use the kernel `globalUnlock()` function to unlock memory.
8. Display the variable with any suitable ToolBook technique.
9. Use the kernel `globalFree()` function to remove the BLOB from memory.

Example

```
to handle buttonClick
   request "Close Table:" && getPXErrorString(closePXTable("Bintest",1))
   request "Open Table:" &&
getPXErrorString(openPXTable("Bintest","Bintest",0,0))
   linkDLL "kernel"
      pointer globalLock(word)
      word globalunlock(word)
      word globalFree(word)
   end linkDLL
   -- we know the fieldName, in this case we'll try "Binfield"
   hBlob = openPXBlobRead("Bintest", "Binfield")
   set bSize to getPXBlobSize(hBlob)
   hB = getPXBlob(hBlob,bSize, 0)
   get closePXBlob(hBlob, 0)
   set pB to globalLock(hB)
   request pointerString(0,pB)
   get globalUnlock(hB)
   get globalFree(hB)
   request "Close Table:" && getPXErrorString(closePXTable("Bintest",1))
end
```

Reading a graphic BLOB

Paradox overview

To read a graphic BLOB:

1. Go to the record that contains the reference to the BLOB.
2. Use `openPXBlobRead()` to open the BLOB to read.

   The engine returns the handle to the BLOB in the BLOB file.
3. Use the getPXGraphicBlob function to read the BLOB into memory.

   The function returns the Windows handle to the BLOB.
4. Use `closePXBlob()` to close the BLOB.
5. Display the BLOB.
6. Use `freePXGraphicBlob` and `freePXGraphicBlobPalette` to free the Windows handle to the bitmap and the palette.

Example

```
-- table is the currently open table, fieldName is the name of the
-- field that has the BLOB in it.
to handle showBitmap table,fieldName
   system hBMP,hPal,hWndBMP
   --get a PX-type handle to the BLOB
   hPXToBMP = openPXBlobRead(table,fieldName)
   --put the BLOB into memory and get a Windows-style handle to it
   hBMP = getPxGraphicBlob(hPXToBMP)
   --Same with its associated palette
   hPal = getPxGraphicBlobPalette(hPXToBMP)
   get closePXBlob(hPXToBMP, 0)
   -- Open a window and show the graphic BLOB.
   myParent = clientHandle of mainWindow
   myPosition = pageUnitsToClient(position of ellipse "Center")
   myColor = rgbFill of ellipse "Center"
   hWndBMP = openPXBitmapWindow(hBMP, hPal, myParent, myPosition, 2, myColor)
   --Imagine the ellipse "center" to be a point. With the mode
   --set to 2, the bitmap will show at its normal size centered on
   --the 1-dimensional ellipse.
end showBitmap

to handle closeBitmap
   system hBMP,hPal,hWndBMP
   --When you're finished with the bitmap and the window, close the window
   get closePXBitmapWindow(hWndBMP)
   --free the Windows handle to the bitmap
   get freePXGraphicBlob(hBMP)
   get freePXGraphicBlobPalette(hPal) -- same for the palette
end closeBitmap
```

Writing a memo BLOB

Paradox overview

To write a memo BLOB:

1. Go to the record you want to write.
2. Use the `charCount()` function to get the size of the BLOB you want to write.
3. Use `openPXBlobWrite()` to open the BLOB to write.
4. Use the `setPXMemoBlob()` function to write the BLOB.
5. Use `closePXBlob()` to close the BLOB.
6. Use `appendPXRecord()` or `insertPXRecord()` to add a new record or `updatePXRecord()` to update an existing one.

Example

```
blobSize = charCount(value)
blobHandle= openPXBlobWrite(DBTable of self, fieldName, BLOBSize, 0)
if blobHandle < 0
   sysError = getPXErrorString(it)
else
   get setPXMemoBlob(blobHandle, value)
   sysError = getPXErrorString(it)
   get closePXBlob(blobHandle, 1)
end
```

Writing a graphic BLOB

Paradox overview

To write a graphic BLOB:

1. Save the BLOB you want to write as a file.
2. Go to the record you want to write.
3. Use the `getPXFileSize()` function to get the size of the BLOB you want to write.
4. Add 8 bytes to that size.

   The extra 8 bytes are for a header that is added automatically to tell the engine what kind of graphic the BLOB is.
5. Use `openPXBlobWrite()` to open the BLOB to write.
6. Use the `setPXGraphicBlobFromFile()` function to write the BLOB.
7. Use `closePXBlob()` to close the BLOB.
8. Use `appendPXRecord()` or `insertPXRecord()` to add a new record or `updatePXRecord()` to update an existing one.

Example

```
blobSize = getPXFileSize(value) + 8
blobHandle= openPXBlobWrite(DBTable of self, fieldName, blobSize, 0)
if blobHandle< 0
   send PXError blobHandle
else
   get setPXGraphicBlobFromFile(blobHandle, value)
   sysError = getPXErrorString(it)
   get closePXBlob(blobHandle, 1)
end
```

Writing a binary BLOB

Paradox overview

Example

```
to handle buttonClick
   set foo to text of field "foo"
   linkDLL "kernel"
      WORD globalAlloc(word,dword)
      POINTER globalLock(word)
      WORD globalFree(word)
      WORD globalUnlock(word)
   end linkDLL

   set hBuf to globalAlloc(68,512)
   set pMemBuf to globalLock(hBuf)
   request "Close Table:" && getPXErrorString(closePXTable("Bintest",1))
   request "Open Table:" &&
getPXErrorString(openPXTable("Bintest","Bintest",0,0))
   get pointerString(0,pMemBuf,foo)
   hBlob = openPXBlobWrite("Bintest", "BinField", 512, 0)
   if hBlob < 0
      request getPXErrorString(hBlob)
      get globalUnlock(hBuf)
      get globalFree(hBuf)
      break
   end
   request "Set blob:" && getPXErrorString(setPXBlob(hBlob,512,0,hBuf))
   get closePXBlob(hBlob, 1)
   get globalUnlock(hBuf)
   get globalFree(hBuf)
   request "Update blob:" && getPXErrorString(updatePXRecord("Bintest"))
   request "Close Table:" && getPXErrorString(closePXTable("Bintest",1))
end
```

Displaying a graphic BLOB

Paradox overview

Before you can display a graphic BLOB, you must create a window for it. You can create the window and display the BLOB at the same time, or you can create the window without displaying a BLOB. You can leave the window open and use it to display a series of BLOBs.

To create the window and display a BLOB:

1. Call `openPXBitmapWindow()` to create the window.

   The parameters for the function are:

`<bitmap handle>`	The handle to the BLOB returned by `getPXGraphicBlob()`
`<palette handle>`	The palette handle returned by `getPXGraphicBlobPalette()`. (Enter 0 to use the ToolBook palette.)
`<parent handle>`	Window in which to create the child window
`<position/bounds>`	Comma-separated list of two screen-coordinate numbers representing the position, or four screen-coordinate numbers representing the bounds, of the child window
`<mode>`	Display mode for bitmap 0 Normal 1 Centered 2 Stretched
`<background color>`	Comma-separated list representing the RGB values for the window background color. Pass null or "" to use the system window background color.

   The function returns the handle to the window.

To create the window without displaying a BLOB:

When you call `openPXBitmapWindow()`, pass the handle to a bitmap that is a ToolBook resource instead of the handle to a BLOB. Pass 0 instead of the handle to a BLOB palette.

The ToolBook resource bitmap is displayed in the window.

To display a BLOB in an existing window:

1. Call `setPXBitmapWindowInfo()` to display the bitmap.

   The parameters for the function are:

`<window handle>`	The handle to the bitmap window returned by `openPXBitmapWindow()`
`<bitmap handle>`	The handle to the BLOB returned by `getPXGraphicBlob()`
`<palette handle>`	The palette handle returned by `getPXGraphicBlobPalette()`. (Enter 0 to use the ToolBook palette.)
`<mode>`	Display mode for bitmap 0 Normal 1 Centered 2 Stretched
`<background color>`	Comma-separated list representing the RGB values for the window background color

When you finish looking at the BLOB, you should call `freePXgraphicBlob()` and `freePXGraphicBlobPalette()` to free the memory used by the BLOB handle and the palette handle. You should also redraw the screen to remove the image of the BLOB.

Example

```
--table is the currently open table, fieldName is the name of the
--field that has the BLOB in it.
to handle showBitmap table,fieldName
   system hBMP,hPal,hWndBMP
   --We get a PX type handle to the BLOB
   hPXToBMP = openPXBlobRead(table,fieldName)
   --We put the BLOB in memory and get a Windows style handle to it
   hBMP = getPxGraphicBlob(hPXToBMP)
   --Same with its associated palette
   hPal = getPxGraphicBlobPalette(hPXToBMP)
   --Done with the PX handle
   get closePXBlob(hPXToBMP, 0)
   --Open a window and show the graphic BLOB.
   hWndBMP = openPXBitmapWindow(hBMP, hPal, clientHandle of \
   mainWindow, pageUnitsToClient(position of ellipse "Center"), \
   2,rgbFill of ellipse "Center")
   --Imagine the ellipse "center" to be a point. With the mode set to
   --2, the bitmap will show at its normal size centered on
   --the 1 dimensional ellipse.
end showBitmap
```

To close the window:

Example

```
to handle closeBitmap
   system hBMP,hPal,hWndBMP
   --When we're finished with the bitmap and the window:
   --close the window
   get closePXBitmapWindow(hWndBMP)
   --free the Windows handle to the bitmap
   get freePXGraphicBlob(hBMP)
   --ditto for the palette
   get freePXGraphicBlobPalette(hPal)
end closeBitmap
```

Writing and reading the BLOB leader

Paradox overview

The BLOB leader is a part of the BLOB that is stored in the database table itself as well as in the BLOB file. The leader is optional and can be up to 240 bytes long. You set the length of the leader when you use `createPXTable()` to create the table.

Example

```
get createPXTable("c:\data\CDbase", "Title, artist, songs, cover", \
   "A25, A25, M240, G0")
```

creates two BLOB fields: one memo field with a 240-byte leader and a graphic field with a 0-byte leader.

The leader is useful for storing a description or other data about the BLOB. If the BLOB itself is less than 240 bytes long, it can be stored entirely in the leader, but it is also stored in the BLOB file.

To write the BLOB leader:

1. If the leader for a BLOB field is longer than 0 bytes, it is filled automatically with the first part of the BLOB when you write the BLOB to the BLOB file. The entire BLOB is stored in the BLOB file; the part that is stored in the leader is a duplicate.

You can read the BLOB leader without reading the entire BLOB.

To read the BLOB leader:

1. Go to the record from which you want to read the BLOB leader.
2. Use the `getPXBlobQuick()` function to read the leader.

Example

```
to handle buttonClick
   linkDLL "kernel"
      pointer globalLock(word)
      word globalunlock(word)
      word globalFree(word)
   end linkDLL
   request "Close Table:" && getPXErrorString(closePXTable("Bintest",1))
   request "Open Table:" &&
getPXErrorString(openPXTable("Bintest","Bintest",0,0))
      -- we know the fieldName, in this case we'll try "home_address"
   fieldSize = getPXFieldType("Bintest","home_address")
   clear first char of fieldSize
   hQuickBlob = getPXBlobQuick("Bintest","home_address",fieldSize)
   pQB = globalLock(hQuickBlob)
   request pointerString(0,pQB)
   get globalUnlock(hQuickBlob)
   get globalFree(hQuickBlob)
end
```

Sharing tables with multiple users

Paradox overview

Paradox database tables can be shared over a network, or they can be shared between two or more ToolBook applications running on the same computer. The engine has several functions you can use to allow several users access to tables while protecting the integrity of the data.

Locking
Managing concurrent operations
Managing security
Sharing and concurrency functions
Password and security functions

Locking

Paradox overview

The main requirement for sharing tables safely and successfully is to work out a scheme for locking and unlocking records and tables. There are four types of locks. Some locks are placed and removed automatically by the engine, but most must be handled explicitly by the ToolBook application.

Locks can be placed on tables, on other kinds of files, and on records.

A single user can lock an object only once, but more than one user can place locks on the same object as long as the locks don't conflict. For example, if you place a full lock on a table, another user can place a write lock on the same table, but no one can place a prevent full lock on the table.

Each user can place only one lock on a single object at a time. However, each user can lock up to 128 records at a time in each open table.

Your locking scheme needs to balance protection of data with access to the data. More stringent locks provide greater assurance of data integrity but make it more difficult for multiple users to have access. For example, if you place a write lock on an entire table while you are working on just one record, no one else will be able to write to any record in the table.

In general, you use a full lock when you want to prevent another user from seeing data that may be invalid or when you're performing operations such as sorting, restructuring, emptying, or deleting a table.

Prevent locks preclude others from placing locks on objects. A prevent full lock on a table means that no one can place a full lock on the table; however, they can place a write lock on it. Placing a prevent write lock or a prevent full lock on an object ensures that you will have access to the object any time you want. However, prevent locks can interfer with other users' operations. For example, if another user wants to perform an operation that requires write-locking a table, and you have placed a prevent write lock on the table, the other user will not be able to proceed until you remove it.

Note that locking a table or file is different from protecting it with a password.

Types of locks

Paradox overview

There are two types of locks and two types of "prevent" locks:

Lock type	Effect	Applies to
Full lock	Prevents all other users from accessing the object in any way	Tables, files
Write lock	Allows other users to read from the object but does not allow them to change its structure or contents	Tables, files, records
Prevent write lock	Prevents other users from placing either a full lock or write lock on the object	Tables, files
Prevent full lock	Prevents other users from placing a full lock on an object. They can still place write locks on it.	Tables, files

Automatic locks

Paradox overview

The engine automatically locks objects when the following functions are called:

Function	Lock
addPXTable()	Write lock on source table, full lock on destination table
copyPXTable()	Write lock on source table, full lock on destination table
createPXTable()	Full lock on table being created
deletePXTable()	Full lock on table being deleted
emptyPXTable()	Full lock on table being emptied
openPXTable()	Prevent full lock on table being opened
renamePXTable()	Full lock on source table, full lock on destination table
addPXKey()	Full lock on table
dropPXKey()	Full lock on table

Manual locks

Paradox overview

Use the following functions to place and remove locks from files, tables, and records:

Function	Use
`lockPXNetFile()`	Place a full, write, prevent full, or prevent write lock on the specified file
`unlockPXNetFile()`	Remove a lock from the specified file
`lockPXNetTable()`	Place a full, write, prevent full, or prevent write lock on the specified table
`unlockPXNetTable()`	Remove a lock from the specified table
`lockPXNetRecord()`	Place a write lock on the current record in the specified table. Returns the lock handle for the locked record.
`unlockPXNetRecord()`	Remove a lock from the record specified by the lock handle returned by `lockPXNetRecord()`

You can lock a record only once. You can lock up to 128 records at a time in each open table.

Checking lock status

Paradox overview

To find out if the current record is locked, call `isPXNetRecordLocked()`. The function returns 1 if the record is locked and 0 if it is not.

Example

```
to handle buttonClick
   set DB to DBTable of this book
   get isPXNetRecordLocked(DB)
   if it < 0
      request getPXErrorString(it)
   else
      request it
   end if

   get isPXNetTableChanged(DB)
   if it < 0
      request getPXErrorString(it)
   else
      request it
   end if

   get isPXTableProtected(DB)
   if it < 0
      request getPXErrorString(it)
   else
      request it
   end if
end
```

Going to a locked record

Paradox overview

To go to a locked record, call `gotoPXNetRecordLock()`, specifying the lock handle returned by `lockPXNetRecord()`.

Managing concurrent operations

Paradox overview

There are four functions that are useful for managing concurrent operations:

`isPXNetTableChanged()`	Tells you if the table has been changed by another user since the last time you read from it; notifies you if the current record has been deleted or if the record number has changed
`refreshPXNetTable()`	Gets current data for the current record; notifies you if the current record has been deleted
`getPXNetUserName()`	Gets the name of the user from the WIN.INI file
`getPXNetErrorUser()`	Reports the name of the user causing a locking error

Managing security

Paradox overview

The engine provides password protection at the table level. If you add a password to a table, the table is encrypted, and all users must supply the correct password before they can open it   Once the correct password is supplied, data is decrypted when it is read from the table and encrypted when it is written to the table.

A password can be up to 15 characters long.

The following functions are used for managing security:

`encryptPXTable()`	Adds password protection to a table. The table is encrypted with the password you choose. The table must be closed when you add password protection.
`decryptPTTable()`	Removes password protection from a table. The table is decrypted and can be opened by anyone. The table must be closed when you remove password protection.
`isPXTableProtected()`	Determines whether a table is protected by a password
`addPXPassword()`	Passes a password to the engine to enable you to open a password-protected table. You can pass several passwords (but only one per function call) so you can open multiple password-protected tables.
`deletePXPassword()`	Deactivates a password so it can no longer be used to open password-protected tables. You can continue to use any tables that were opened with the password.

Alias

A name you give to a table when you open it. The alias is used to refer to the table in functions as long as it is open.

Normal bitmap display mode

The bitmap is displayed in top-left corner of the window. If `<position/bounds>` is a point, the window is sized to fit   the bitmap. If `<position/bounds>` is a rectangle, the window is drawn as specified. If the bitmap is smaller than the window, the background color shows around it. If the bitmap is larger than the window, it is cropped.

Centered bitmap display mode

If `<position/bounds>` is a point, the bitmap is centered on the point and the window is sized to fit the bitmap. If `<position/bounds>` is a rectangle, the window is drawn as specified and the bitmap is centered in the window. If the bitmap is smaller than the window, the background color shows around it. If the bitmap is larger than the window, it is cropped.

Stretched bitmap display mode

If `<position/bounds>` is a point, the window is sized to fit the bitmap. If `<position/bounds>` is a rectangle, the window is drawn as specified and the bitmap is stretched to fill it.

Handle

A number that can be used to refer to a file, index, or BLOB.

BLOB

A BLOB is any block of binary data (such as a graphic, video clip, or sound) or a block of text data that is larger than 255 bytes. BLOBs are stored in a separate file, and the BLOB's handle is stored in a field in the table.

Mode

0	Changes are saved in swap buffer
1	Changes are saved to disk

Changes to a shared table are always saved to disk, regardless of the mode.

Index handle

0	for the primary index for a table
1 - 255	(the number of the field) for case-sensitive, single-field secondary indexes
>256	Field handle for composite or case-insensitive, single-field indexes

Cloning BLOBs

Paradox overview

If you read a BLOB from a record, the BLOB is cleared from memory when you go to another record. If you want to keep that BLOB in memory while going to other records, you can use the clonePXBlob() function to clone it.

When you clone a BLOB, you create a private copy of a public BLOB.

Writing a binary BLOB from a file

Paradox overview

Example

```
to handle buttonClick
   request "Close Table:" && getPXErrorString(closePXTable("Bintest",1))
   request "Open Table:" &&
getPXErrorString(openPXTable("Bintest","Bintest",0,0))
   set blobSize to getPXFileSize("blob.txt")
   hBlob = openPXBlobWrite("Bintest", "BinField", blobSize, 0)
   if hBlob < 0
      request getPXErrorString(hBlob)
      get globalUnlock(hBuf)
      get globalFree(hBuf)
      break
   end
   request "Set blob from file:" &&
getPXErrorString(setPXBlobFromFile(hBlob,blobSize,0,0,"blob.txt"))
   get closePXBlob(hBlob, 1)
   request "Update blob:" && getPXErrorString(updatePXRecord("Bintest"))
   request "Close Table:" && getPXErrorString(closePXTable("Bintest",1))
   get globalUnlock(hBuf)
   get globalFree(hBuf)
end
```

Writing a binary BLOB to a file

Paradox overview

Example

```
to handle buttonClick
   request "Close Table:" && getPXErrorString(closePXTable("Bintest",1))
   request "Open Table:" &&
getPXErrorString(openPXTable("Bintest","Bintest",0,0))
   --we know the fieldName, in this case we'll try "Binfield"
   hBlob = openPXBlobRead("Bintest", "Binfield")
   set wE to writePXBlobToFile(hBlob,"c:\database\blob2.txt",2)
   if wE < 0
      request "Write to file:" && \
      getPXErrorString(wE)
   else
      request "File size written:" && wE
   end if
   request "Close Table:" && \
   getPXErrorString(closePXTable("Bintest",1))
end
```

Engine defaults

When the first Engine function is called, the MaxFiles, MaxLocks, MaxTables, and SwapSize entries are read from the [Paradox Engine] section of the WIN.INI file and stored in the PDOXWIN.DLL as long as it is in memory. All applications that use the Paradox Engine share this instance of the PDOXWIN.DLL. Therefore, once any application has loaded the DLL and called an Engine function, the only way to change the value of the any of these variables is to exit all applications that use the Engine and change the WIN.INI settings. The following functions can be called to query and change the WIN.INI file without causing the Engine to read the defaults.

getPXSwapSizeFromINI()
getPXMaxTablesFromINI()
getPXMaxFilesFromINI()
getPXMaxLocksFromINI()
setPXINIMaxFiles()
setPXINIMaxLocks()
setPXINIMaxTables()
setPXINISwapSize()