vxBase™

Visual Basic
xBase Functions for Windows 3.1

vxBase is Shareware, not freeware. After a thirty day evaluation period, if you continue to use vxBase, you are required to register the product and include a license fee of $59.95 plus $5.00 Shipping and Handling (North America) by check, money order, Visa, or Mastercard. Registration Information may be found at the end of this document. The License fee will entitle you to a registration code (to be rid of the opening nagware) and the latest version of the software.

If you distribute vxBase with your Visual Basic application, you must distribute an unregistered copy of the software unless you purchase a developer distribution license.

Developer Distribution Licenses

You may distribute an unlimited number of copies of vxBase with your application by purchasing a developer distribution license for $195.00 (less the shareware registration fee if already registered). This license entitles you to a printed copy of the manual, the latest version of the software, and a run time only version of vxbase.dll which is distributed to the end user. See the last page in this document for ordering details.

Please read the License Agreement and Limited Warranty found at the end of this manual before you begin to use vxBase.

Manuals

A printed copy of this manual is available for $20.00 to users who do not purchase the developer distribution license.

Requirements

Windows 3.1 and Visual Basic 2.0 or greater are required to run the vxBase sample applications.

Release Notes

Release 3.07 August 11, 1993

vxBase was written in C by Terry Orletsky. Address inquiries and bug reports (preferably Dr. Watson along with a listing of the offending code) to

Terry Orletsky

vxBase Systems
#488, 9707 - 110 Street
Edmonton, Alberta, Canada
T5K 2L9

Phone (403) 488-8100
Fax (403) 488-8150
BBS (403) 488-8365
Compuserve I.D. 70524, 3723

Trademarks

Acknowledgements

Thanks to Ray Donahue of Hamden, CT for his three dimensional controls, to Jonathan Zuck of UFI for his help and advice through the Microsoft Basic forum on Compuserve, to Willy Koch of Geneva for his French translation, to George Santamarina of Miami for his Spanish translation, to Manfred Waldmeyer of Tornesch-Esingen, Germany for his German translation, to Massimiliano Bellucci of Cecina, Italy for the Italian translation, and to Kurt Rombaut of Lokeren, Belgium for the Dutch.

Testing

vxBase was written and tested extensively on a Pegasus 386 33mhz microcomputer with 8 megabytes of RAM, SVGA, and a 200 megabyte hard disk running Dos 5.0, QEMM 6.0, Stacker 2.0, and Windows 3.1 in Enhanced mode. The sample application has been installed and successfully run on a variety of 286, 386, and 486 PCs.

Record and file locking routines were tested and verified on an 18-station Novell 386 Netware LAN with 3 workstations running the sample application concurrently.

If your hardware or LAN software differs significantly and vxBase does not run properly, I would appreciate a Dr. Watson report sent by fax, to the vxBase BBS, or to my Compuserve address.

vxBase Table of Contents

Functions

Contents (continued)

Contents (continued)

Installation from Diskette

If you receive a copy of vxBase from the manufacturer on diskette, insert the diskette in drive A: or B: and run A:INSTALL or B:INSTALL from the Program Manager Run Command.

The Visual Basic sample programs and files will be set up in directory \VB\VXBTEST. It is strongly recommended that you do not change this directory. A sample application written in C will be set up in directory \VB\VXC.

The following files will be set up in your \WINDOWS directory:

vxbase.inf	vxbase language and registration info
vxbase.dll	the vxbase dynamic link library
vxbase.wri	vxbase documentation in a Windows Write file
vxload.exe	vxbase dll loader for use with Visual Basic in Design Mode
unpack.exe	unpacking utility in case you have to copy any vxbase installation files directly from the diskette

Installation from Compuserve, Other Bulletin Boards, or Shareware Houses

vxBase is distributed on bulletin boards or from Shareware Houses as two compressed .ZIP files. The first ZIP file is vxbdoc.zip, which contains the Windows Write file that you are reading now. It is separated from the rest of vxBase to allow potential users to preview the documentation before installing and actually using vxBase. This is especially helpful to potential users who extract vxBase from a bulletin board. They can evaluate the system from a documentation standpoint before committing to down-loading the larger system.

The second ZIP (vxbase.zip) file contains the sample source code and Visual Basic project files, vxbase.txt which includes all of the Visual Basic declarations for the routines in the vxBase DLL and the vxBase DLL itself.

If you are going to upload vxBase to a bulletin board, it must be sent as it was received - in two ZIP files.

When the system ZIP file is decompressed, it contains a readme.doc file which contains these installation instructions, and 3 more ZIP files. These ZIP files are:

vxbdll.zip	the vxBase DLL and vxload.exe
vxbtest.zip	sample source code, sample database, and vxbase.txt
vxbezy.zip	simple one database sample app

To install vxBase, first make a subdirectory under your \VB directory named \VB\VXBTEST and copy the vxbtest.zip file there. Unzip it and delete the vxbtest.zip file from your hard disk. To run the sample application it is essential that these files be in directory \VB\VXBTEST because this path is hard-coded into the sample code. If you MUST put it somewhere else, you'll have to modify the file names in the source code to reflect your new location. Also unzip vxbezy.zip into the \VB\VXBTEST directory.

Unzip vxbdll.zip and place the resulting files (VXBASE.DLL and VXLOAD.EXE) in your \WINDOWS directory. The DLL must be in a directory that Windows can find (i.e., in your path). The handiest place is in your \WINDOWS directory.

To run the sample application see Creating a vxBase Application and the Sample Application sections below.

Release History

vxBase 1.00
November 10, 1991 original release

vxBase 2.00
October 9, 1992
28 Functions added:

vxBrowseSetup	allows the user to fine tune the appearance of a Browse table (both the old window browse and the new vxCtlBrowse).
vxCtlBrowse	allows the placement of a browse table in a form multiline text box. Communication with the browse table is enabled with the new vxCtlBrowseMsg function. The Browse table no longer has to be terminated when a selection is made, etc. It also allows dynamic memo linking. All standard events and procedures attached to the text box may be used in normal fashion while the browse is running.
vxCtlBrowseMsg	communicates with a vxCtlBrowse. Messages the programmer can pass are both interrogatory and procedural (e.g., VXB_GETCURRENTREC extracts the record number of the currently highlighted record and VXB_REFRESH redraws the browse starting at a different record number).
vxCtlFormat	adds TEXT FORMATTING to vxBase.
vxCtlHwnd	gets the window handle associated with a Visual Basic control.
vxCtlPenWidth	added to control the depth of Recessing and Raising a control when using vxCtlStyle.
vxDbfCurrent	reports the current database select area.
vxErrorTest	added to test the result of a vxBase function that uses the alternate error method set by vxSetErrorMethod. Add VxErrorStruc type as defined in the function reference.
vxGetVersion	returns a string containing the current vxBase version number.
vxLocate	searches for a record from and including the current record position that satisfies a logical xBase expression. The search direction may be specified.

vxLocateAgain	searches for a record from and NOT including the current record position that satisfies a logical xBase expression as defined by the last vxLocate for the selected database. The search direction may be specified.
vxFieldTrim	returns a string representing the defined field with trailing spaces removed.
vxNtxCurrent	reports the current index select area.
vxNtxRecNo	returns the ordinal position of the key in the current index.
vxPrinterDefault	returns a string describing the Windows default printer in a format suitable for use by vxPrinterSelect
vxPrinterEnum	enumerates all printers on the system and returns a string suitable for setting the default printer with vxPrinterSelect.
vxPrinterSelect	changes the default Windows printer. The printer setup string must be in the same format as that returned by vxPrinterEnum.
vxReplDateString	replaces a field with a date string formatted as per vxSetDate (default "mm/dd/yy"). This goes hand in glove with dates input into text boxes via vxCtlFormat or displayed with vxDateString.
vxReplRecord	replaces the entire record buffer with the data pointed to by a record typedef or string (see vxRecord). BE CAREFUL with this function. No data checks are implemented!
vxSeekFast	speeds up seek times on Read Only files by 35%.
vxSetAlias	allows field qualification in all vxBase field functions.
vxSetErrorMethod	allows an alternate method of trapping errors found by vxBase. The normal method is to report the error through a message box at run time. If you use the alternate method, nothing is reported (for most functions); instead, an error structure is filled with information about the error which may be extracted with the vxErrorTest function.
vxSetMeters	allows you to turn the analog meter bars displayed by vxPack, vxReindex, and vxTestNtx on or off (default is ON).

vxSetRelation	adds true relational capability to vxBase.
vxTableFieldExt	added to provide column definitions to vxBrowse when using vxSetRelation to add multi-file fields on the same browse row.

International Functions Added

The following functions all deal with the problem of a database that contains characters from the high end of the ANSI or OEM character sets, which is commonplace if the database stores data in a language other than English.

vxSetAnsi(FALSE) properly handles databases that were created with a DOS based application (such as Clipper). These databases are OEM databases. Characters with diacritical marks in the high end of the OEM character collating sequence are NOT the same as the ANSI characters. It is necessary for vxBase to translate the characters to ANSI (both Windows and vxBase native mode) before they can be used in a vxBase application. They also must be translated back again when they are written.

The default value of vxSetAnsi is TRUE (no translation takes place). If the database was created and is maintained by vxBase (or DataWorks), and the database is going to be used exclusively by Windows applications, vxSetAnsi should be TRUE.

vxCollate allows the programmer to create his own collating sequence table (for EITHER an ANSI database or an OEM database). The OEM character set in particular does not use any kind of logical collating sequence for characters with diacritical marks. The ANSI table handles these characters more intelligently - but its sequence is also incorrect for some languages.

It is necessary to define a collating sequence table to properly build an index that uses these characters.

vxSetCollate can toggle a defined collating table on or off.

vxBase 3.01

June 7, 1993

Changes made to Version 2.00

File open error when using vxSetupPrinter corrected.

vxFilterReset memory deallocation corrected.

vxBrowse creeping window corrected when using a parent window that has no menu bar.

vxUseNtx consecutive calls now returns the correct, previous select area and does not open the index file again.

vxBrowseSetup corrected to display user menus if system menus disabled.

vxCtlStyle corrected to handle new Visual Basic 2.0 "graphical objects" (e.g., labels).

All SETTINGS cited as System Wide in the current documentation are now local to the vxBase task. These settings include the following:

vxCtlPenWidth
vxSetAnsi
vxSetCollate/vxCollate collating table
vxSetDate
vxSetErrorCaption
vxSetErrorMethod
vxSetLanguage
vxSetLocks
vxSetMeters
vxSetString

If more than one vxBase task is running at the same time, the items set by the functions above are now local to each concurrent task (e.g., vxBase in French and in English can be run on the same machine at the same time).

Inconsistent index file lockup in multiuser mode when vxSeek fails corrected.

DESCEND and vxDescend index algorithms changed to use 2s complement instead of 1s complement arithmetic to convert defined descending keys into the correct sequence. This makes the vxBase DESCEND index compatible with CLIPPER.

NOTE: ALL DESCENDING INDEXES CREATED WITH VXBASE PRIOR TO THIS RELEASE SHOULD BE REBUILT.

Memo soft returns from Clipper apps are now properly stripped on European memos when vxSetAnsi is FALSE.

Dutch language support added.

The vxBase alternate error method now hooks in to Visual Basic ON ERROR GOTO.

vxBrowse and vxCtlBrowse quick key displays and vertical scrolling speeds have been increased by a factor of 2 to 3.

Memory leaks created by VB 2.0 string creation corrected by using new function to create Visual Basic strings if version is 2.0 or greater.

Max memo size now increased to 64k (was 32k).

Memo reading procedures changed to read 4k blocks instead of 64k file chunks. This should speed memo reads and packs with memo files considerably.

Memo reads no longer lock the memo file.

vxAppendFrom now appends memos and bitmaps. Restriction removed.

vxGo() passed with a zero record number now returns FALSE instead of crashing.

Dbf files opened as Read Only with vxUseDbfRO no longer have their index files locked when positioning to a specific record.

More elegant on screen edit within a vxBrowse or vxCtlBrowse table.

Initial memory allocation requirements reduced.

vxTop() on empty file now properly returns FALSE.

Dutch language support added.

vxBase 3.00 18 New Functions

vxBlobRead	read a binary large object that has been attached to a memo field.
vxBlobWrite	write a binary large object to a memo field by passing a global memory handle.
vxCreateSubNtx	create a permanent sub index that represents a subset of records in the main database that conform to some condition.
vxIsPicture	determine whether a memo field holds a picture (i.e., a blob or bitmap) or not.
vxIsSubNtx	determine whether or not the defined index is a sub index.
vxLockRetry	set error message protocol for file/record in use and also set lock try timeout value.
vxMemCompact	perform memory compacting routines on Windows global memory.
vxMemoClear	remove a reference to a memo or a bitmap from the dbf memo field.
vxMemoPos	set initial size and position of a vxMemoEdit window.
vxMemRealloc	shrink the initial size of the vxBase memory pool.
vxNtxSubExpr	extract the conditional expression that controls a sub index.
vxNumRecsFilter	determine the number of records in a database that pass a filter condition.
vxNumRecsSub	determine the number of records in a sub index.
vxPictureImport	import a bmp or rle picture file into a memo.
vxPictureRead	read a bitmap from a memo file.
vxSetSelect	turn auto database selection on or off.
vxUseDbfAgain	open another instance of an already open dbf.
vxUseDbfEX	open a database for exclusive use.

Release 3.03-3.07 Changes

Release 3.03 increased max filter string length to 1024 from 512. Filters set in child files are now respected in vxSetRelation chains.

Release 3.04 corrected vxPack problem (640/1602 error) when every record in the 64k read buffer was deleted.

Release 3.05 added function vxPicturePrint and the following elements to the xBase expression parsing engine:

function ALLTRIM()	trims right and left
functiom EMPTY()	true if string is all spaces, blank date, or 0 numeric value or .F. logical
operator !	equivalent to .NOT.
operator ==	exactly equal
#	not equal (equivalent to <>)
!=	not equal (equivalent to <>)

Release 3.06 adds:

(1) sub function vxSetGauge allows the programmer to use his own gauge controls in vxReindex, vxPack, etc.

(2) new vxCtlBrowseMsg messages VXB_GO, VXB_SKIP (see below).

(3) quasi-xBase function TIMEODAY(). TIMEODAY(val) converts a number (either literal or contained in a numeric field) representing minutes into a string representing the time of day (e.g., TIMEODAY(510) = " 8:30 am").

(4) "Exit" item aded to vxMemoEdit window.

(5) minor positioning and scroll problems corrected in vxCtlBrowse.

(6) xBase expressions that do not have their elements separated by spaces now parse properly (e.g., (Y_N1.and.Y_N2) formerly returned an error unless ".and." was separated from the field names by spaces).

(7) memo overwrite caused by 3.02 changes corrected in 3.07. 3.07 also corrects an index key append problem with sub indexes and also makes some minor changes to vxNumRecsSub and vxNtxRecNo.

Compatibility Issues

With release 3.0, vxBase breaks away from strict xBase standards. Bitmaps of course may only be retrieved and displayed using vxBase functions. The header block of a subindex also differs from a standard Clipper NTX index header block. New elements have been added that define whether or not the index is a subindex and an area is used to hold the conditional expression as well.

Release 5.2 of CA-Clipper changes the index header as well and also changes the index locking scheme. vxBase NTX indexes ARE NOT COMPATIBLE with Clipper 5.2 indexes. Indexes created and/or maintained with any version of Clipper will NOT properly maintain a vxBase subindex.

Bitmaps and subindexes are specialty items requested by a great many vxBase users. If compatibility is an issue with your application, it would be best not to use these functions (although bitmaps may be safely ignored; if the memo text returns "BIT*MAP" then it may be ignored in a Clipper application).

Creating a vxBase Application

SHARE.EXE

The program SHARE.EXE must be loaded at the workstation to run vxBase. This is a WINDOWS 3.1 requirement!

Visual Basic

Your application requires the vxbase.txt file (which should be in directory \vb\vxbtest if you followed the installation instructions) placed in the Global module. You may simply wish to copy the Global module from the sample application, which contains some useful declarations from the WIN API, as well.

Visual Basic and VXLOAD.EXE

A utility program named vxload.exe is included with the vxbase DLL and is normally installed in your \WINDOWS directory. This program is for use with Visual Basic in Design Mode. vxBase maintains a single memory pool for use by all concurrent vxBase applications. This memory pool is attached to the FIRST program that calls a vxBase function. Programmers testing their Visual Basic/vxBase programs by running them in Design Mode have frequent program failures (syntax errors, etc.). In Windows 3.0 we relied on a call to the vxDeallocate() function to detach the vxBase memory pool from VB.EXE (i.e., Visual Basic running in Design Mode). Whenever the test run ended, we could always rely on vxbase.dll being unloaded. Under Windows 3.1, however, an ungraceful exit from a test run does not always unload the DLL. Subsequent attempts at running the program (or even another Visual Basic program) could end in failure with a General Protection Fault in the memory allocation routines. VXLOAD was written to overcome this problem.

Set up VXLOAD as a program item with its icon adjacent to the Visual Basic icon and ALWAYS RUN IT prior to starting up Visual Basic. It runs in an iconized state, consumes little extra memory, and controls the vxBase memory pool. With VXLOAD running, any unexpected failures of your test programs in Visual Basic design mode will never result in compromised memory because VXLOAD controls it.

It is highly recommended that you also include two statements after your call to vxInit() in your program initialization sequence:

```
Call vxSetLocks(FALSE)
j% = vxCloseAll()
```

The first statement will ensure that no file is locked if your program terminates abnormally. Subsequent runs will not balk because of a lock left in place due to the program terminating before its time.

The second statement will close all files left open by an abnormal termination so you can start with a clean slate when you try again.

If you wish to use the default locking scheme in your running application, remove the vxSetLocks command before creating your .EXE file.

Note that vxInit and vxDeallocate are still required elements in a vxBase program.

Also note that if you terminate Visual Basic with vxDeallocate never having been called, an attempt to close VXLOAD from the VXLOAD system menu will fail with a "Task Closure Sequence Error" because Visual Basic as a task has never been deregistered from the vxBase task list (vxDeallocate does this). If this happens to you, you may force an unload of VXLOAD by restoring the VXLOAD window and selecting the EXIT item from the VXLOAD menu.

Realizer

vxBase may be used with Realizer as well as Visual Basic. VB Strings are not compatible with Realizer, however. See vxSetString and vxRecord for an example of extracting data from a database using Realizer. VB specific functions that use handles to Visual Basic controls (e.g., vxCtlLength, vxCtlStyle) will not work either.

C

vxBase functions may also be called from C languages (Borland C++, Turbo C for Windows, Microsoft C/C++ 7.0, Microsoft Quick C, Microsoft Visual C/C++). Contact the author for a sample application written in Microsoft C/C++ 7.0. The sample includes the vxbase header file and import library.

xBase Expressions, Functions, and Operators

Compatibilities and Incompatibilities

vxBase dbf files (database files) and dbt files (memo files) are compatible with those of Clipper, dBase III and III+, and any other "xBase product". They are not compatible with dBase IV.

vxBase index files use Clipper standard .ntx files (Clipper version 5.1 and below). These indexes are more efficient both in speed and size than traditional ndx files. vxBase again imposes one important restriction. In the interests of speed and simplicity, all indexing expressions must evaluate as strings.

vxBase sub indexes may be read by Clipper applications but they will NOT be properly maintained by a Clipper application. vxBase indexes built with the DESCEND() function are only compatible with Clipper descending indexes with vxBase release 2.09 and above.

NOTE: current indexes you wish to use in a new vxBase application must be converted if they contain numeric fields or date fields. Use the STR() function to convert numeric fields to strings, and the DTOS() function to convert date fields to strings within your index expression.

vxBase bitmaps and blobs (binary large objects) stored in memo files will not interfere with Clipper memo functions. They will appear in a Clipper memo as the string "BIT*MAP". You can test for this value in a Clipper application and disallow editing. If the bitmap is overwritten with Clipper memo text, it will of course be lost.

Alias Names

An alias name is an alternative method for indicating the database that a field belongs to. vxBase uses integer select areas returned when the file is opened to refer to a database and to select it as the database of record. An alaias name (as a character string) may be assigned to this integer with the vxSetAlias function. The alias is used to access a field in an open database in a select area other than the one currently selected (or to ensure that the correct database is referenced no matter what the current database selection). vxBase will not permit access to fields in a database open in a work area other than the current selection unless an alias name is used.

xBase style alias names and alias names set with the vxSetAlias function are supported within a vxBase xBase expression. The alias names used must be set with vxSetAlias. File alias names are separated from the field reference by "->" (classical xBase syntax) within an xBase expression string. When alias names are used within vxBase functions that refer to field names, a period delimiter is used instead (to conform to Visual Basic syntax).

For example,

```
MasterDbf = vxUseDbf("cmaster.dbf")
If vxSetAlias("master", MasterDbf) Then
    If Not vxEval("master->country = 'Canada'") Then
      MsgBox "Country does not exist"
   Else
      Country$ = vxField("master.country")
   End If
End If
```

Conventions

This section and those following on Expressions, Constants, Operators, and Functions refer to xBase conventions. xBase expressions are used within vxBase to communicate with the xBase file via standard xBase index expressions, filter strings, etc. These expressions are not available directly from Visual Basic; rather, they are passed as parameters to vxBase functions that do the low level work of translating and validating the expressions.

For example, Newdate$ = DTOS(datefield) is illegal. Instead, the xBase expression must be evaluated by the vxBase parser by passing the expression as a character string parameter (either as a literal string or as a string variable) in a vxBase function that takes xBase expression parameters.

The following vxBase functions take xBase expressions as parameters:

vxCreateNtx	vxJoin
vxEval	vxLocate
vxEvalDouble	vxSetRelation
vxEvalLogical	vxTableField
vxEvalString	vxTableFieldExt
vxFilter	vxCtlBrowseMsg
vxCreateSubNtx	

Program variables may not be used directly within an xBase expression. For example, suppose you wish to build a filter expression to display only records in a certain country. You would solicit the name of the country from the user and store it in variable Country$. To build a filter string suitable for passing to vxFilter, you would use the following code (assuming the xBase field name is "country").

```
Filter$ = "country = upper('" + Country$ + "')"
Call vxFilter(Filter$)
```

If the user entered "Canada" and you stored it in Country$, the content of Filter$ would be country = upper('Canada'), which is a valid xBase expression.

Expressions

Expressions are character strings that consist of field names, functions, constants, and operators that are formatted in conventional xBase syntax. They are used for index expressions, filter expressions, and expressions that control vxBrowse displays. The only difference between vxBase expressions and conventional xBase expressions is in the characters that delimit strings. vxBase only supports single or double quotes; the traditional square bracket [ ] is not supported. Visual Basic functions must not be included in a parameter passed to a vxBase

function that requires an xBase expression.

Conventional xBase functions and operators that are supported by vxBase are listed below. These and only these may be included in the construction of an xBase expression.

An expression may be as simple as a single field name (e.g., cust_name) or as complicated as an IIF function which returns the result of complex expressions (e.g., IIF(left(phone_num,1)=" ","No phone on File",area_code+phone_num)).

The IIF example expressed in normal language would read as "If the first character of the phone_num field is blank, output the phrase 'No phone on file'; otherwise, output the area code plus the phone number". This expression contains two functions (IIF() and LEFT()), two constants (a space between the two quotation marks and the phrase "No phone on file"), two field names (phone_num and area_code), and two operators (the relational operator equal sign = and the string concatenation operator plus sign +).

Expressions are used in index keys, filter definitions, definition of beginning of file and end of file logic to a user table, in statements used to join (or relate) one file to another, in statements used to define the contents of a display column when creating a browse table, and to return the results of logical, numeric, or character expressions to the program when using the vxEvalXXX functions.

All expressions return a value of a specific type - either character, numeric, date, or logical. In many cases, vxBase requires that an expression return a value of a specific type. For example, when defining a filter expression to limit the viewable records, the expression must evaluate as logical (i.e., either TRUE or FALSE). A conditional filter may be defined that limits a view to all customer records that begin with the letter "A". This condition could be expressed as LEFT(cust_name,1)="A". vxBase would interpret this as "If the leftmost character of the field CUST_NAME is an "A", then display the record". The presence of a relational operator (in this case, the equal sign) generally denotes an expression that will evaluate as logical.

Expressions may be entered in upper or lower case.

Constants

An expression may contain one or more numeric, character, or logical constants. An expression which consists of a single constant is not very useful. Constants are usually used within more complex expressions.

A numeric constant represents a number. For example, 4, 9.21, and -26 are all numeric constants.

Character or string constants are always delimited with quotation marks, either single or double. "This is a string", 'so is this', and "John has 3 apples" are all character constants. A string that contains either a double or single quotation mark must be delimited with the

other mark. For example, "John's apple" is a valid string. 'John's apple' is not a valid string. You will normally be passing constants from the Visual Basic environment to vxBase. In this case, the normal procedure would be to delimit the entire expression in double quotes and any string constants that form part of the expression in single quotes.

Logical constants are represented by .TRUE. or .FALSE.. Note the leading and trailing periods. .T. and .F. are valid abbreviations for the logical constants and the letters must be bounded by periods on both sides.

Operators

Operators are signs used to manipulate fields, constants, and the results of functions. A plus sign (+) is used as an Add Operator in the expression 4+5. Two numeric constants are added together to return the numeric value 9.

Operators are type specific. For example, arithmetic operators must act on numeric types. The Divide Operator (/) only acts on numeric types. Some operators perform double duty. The Plus and Minus signs are both arithmetic and string operators. vxBase determines the appropriate operation according to the type of data being acted upon. The data types on either side of a relational operator must be the same (i.e., strings must be compared to strings and numbers must be compared to numbers). Functions which change the data type may be used to convert operands for use in relational expressions.

The only mixed operands allowed are involved in Date Arithmetic. A numeric constant, field, or expression may be added to or subtracted from a date type. Dates subtracted from dates yield a numeric type (i.e., the number of days between two dates).

Numeric Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division (divide by zero returns zero instead of crashing)
^ or **	Exponentiation
()	Groups sets of numbers (evaluation order)

Relational Operators

=	Equal to
#	Not equal to
!=	Not equal to
<>	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
$	Is contained in the set or is a subset of

All relational operators return a Logical result. All operators except the Contains($) operator work on numeric, character, or date values. The $ operator works on two values of type character and returns

true if the first value is contained in the second (e.g., "DC"$"ABDC" returns .TRUE.).

Logical Operators

.AND. both expressions are true
.OR. either expression is true
.NOT. either expression is false
! equivalent to .NOT.

Note the leading and trailing periods that delimit a logical operator (except with !).

Character (String) Operators

+ Concatenates (joins) two or more character expressions. Trailing blank spaces in the expressions will be placed at the end of each expression.

- Concatenates two or more expressions. Trailing blank spaces will be removed from the expression preceding the operator and placed at the end of the expression following the minus sign operator.

Operator Precedence

When more than one type of operator appears in an expression, the order of evaluation is as follows:

string
numeric
relational
logical

Expressions containing more than one operator are evaluated from left to right. Parentheses can be used to change the precedence level of operators (see example below). If parentheses are nested, the innermost set is evaluated first.

Numeric operators are evaluated as follows:

operators contained in parentheses
exponentiation
multiplication and division
addition and subtraction

Evaluation order may be altered with parentheses:

1+2*3+4 = 11
(1+2)*3+4 = 13
(1+2)*(3+4) = 21

Functions

Functions may be used as expressions or parts of expressions. Functions always return a value.

One of the most common uses of functions is to convert one data type into another. Functions can also extract system and database-specific information.

Functions are formatted as FunctionName(Parameters). The number and type of parameters contained within the function parentheses depend on the specific function being called.

The following functions are available. For more information, see the specific commands following the table.

Function	Returns
ALLTRIM(Char Value)	removes leading and trailing spaces
CTOD(Char_Value)	Character to date
DATE()	System date
DAY(Date_Value)	Numeric day
DELETED()	.TRUE. if deleted
DESCEND()	Create descending index
DTOC(Date_Value)	Date to character
DTOS(Date_Value)	Date to string
EMPTY(Value)	True if string is blank or number is 0
IIF(Logical, True Result, False Result)	Logical if
LEFT(Char_Value, Length)	Leftmost n characters
LEN(Char_Value)	Return the length of the char expr
MONTH(Date_Value)	Numeric month
RECNO()	Record number
RIGHT(Char_Value, Length)	Rightmost n characters
SOUNDEX(Char_Value)	String to phonetic complement
SPACE(n)	Generate a string consisting of n spaces
STR(Number, Len, Dec)	Numeric value to string
SUBSTR(Char_Value, Start, Length)	Substring
TIME()	System time as string
TIMEODAY(Num_Value)	Converts minutes to time display format
TRIM(Char_Value)	Trim trailing spaces from string
UPPER(Char_Value)	Convert to uppercase
VAL(Char_Value)	Character to numeric value
YEAR(Date_Value)	Numeric Year

ALLTRIM(Char_Value)

Trim leading and trailing spaces from a character field

Example: ALLTRIM(" White space string ") returns "White space string"

CTOD(Char_Value)

Character to date function.

Converts a character value in the form "MM/DD/YY" into a date value. If vxSetDate has been used to set a different date format, that format MUST be used instead of the default "MM/DD/YY".

Example: CTOD("07/22/91") returns a date in the form CCYYMMDD 19910722. A blank date defaults to January 1, 1980.

DATE()

System date function.

Returns the system date as a date value.

Example: DTOC(DATE()) returns "07/22/91" If the date is July 22, 1991 and the default date format is used (VX_AMERICAN). If another international format is selected with vxSetDate then that format will be used.

DAY(Date_Value)

Numeric day function.

Returns the day in a date_value as a number.

```
    Example:  DAY(DATE()) returns 22 if the date is July 22, 1991

DELETED()
    Logical delete function.
    Returns .TRUE. if the current record has been flagged for deletion.
    Example:  IIF(DELETED(), "Deleted", "Not Deleted")

DESCEND()
    Create descending index.
    An entire index expression or an element of a complex index
          expression may be encapsulated within a DESCEND function to
          reverse the normal ascending sequence.
    Example: UPPER(cust_name) + DESCEND(DTOS(order_day)) could be used
          with vxCreateNtx to create an index that displayed records in
          ascending customer  name order and descending order date
          (i.e., the latest dates first instead of    the oldest).

DTOC(Date_Value)
    Date to character function.
    Converts a date value into a character string in the format
          "MM/DD/YY" or into whatever format has been selected with
          vxSetDate.
    Example:  DTOC(DATE()) returns "07/22/91" if the date is July 22,
          1991

DTOS(Date_Value)
    Date to string function.
    Converts a date value into a character string in the format
          "CCYYMMDD".  Should always be used in index expressions if a
          date field is part of the index key expression.
    Example:  DTOS(DATE()) returns "19910722" if the date is July 22,
          1991

EMPTY(Value)
    Returns TRUE if a character string is all blank or a numeric field
          is zero or a logical field is FALSE.

IIF(Logical_Value, True_Result, False_Result)
    Logical if function.
    If Logical_Value is evaluated as .TRUE., then the expression
          represented by True_Result is returned; otherwise, the
          expression represented by False_Result is returned.
          True_Result and False_Result must be of the same type.
    Example 1:  IIF(YEAR(DATE()) < 1992, "Last Year", "This Year")
    Example 2:  IIF(amt_owing>0, amt_owing, 0)

    Note: IF() may be used instead of IIF() to conform to Clipper
         conventions.

LEFT(Char_Value, Length)
    Leftmost characters function.
    Returns the characters on the left side of the string for the
          specified length.
    Example:  IIF(LEFT(NAME,1)<>"A", "Does not begin with A", "begins
```

with A")

LEN(Char_Value)
Get the length of a character expression.
The length of a blank character expression that has been TRIMmed will be zero.
Example: IIF(LEN(TRIM(company))>0, company, name)

MONTH(Date_Value)
Numeric month function.
Returns the month in a date_value as a number.
Example: MONTH(DATE()) returns 7 if the date is July 22, 1991

RECNO()
Record number function.
Returns the physical record number of the current record. The record's logical position according to the current index is probably not the same as this number. The record number normally reflects the sequence in which the record was entered.
Example: STR(RECNO(),6,0)

RIGHT(Char_Value, Length)
Rightmost characters function.
Returns the characters on the right side of the string for the specified length.
Example: RIGHT("ABCDEF", 3) returns "DEF"

SOUNDEX(Char_Value)
Character string to phonetic complement function. Useful for indexing and searching.
Returns a character string in the form AA111.
Used primarily for indexes on names and descriptions to conserve index file space and simplify lookups where the precise spelling of an item (other than the first two characters) is unknown. Always results in a table display that approximates alphabetical order.
Note: The vxBase Soundex function is NOT the same as the Clipper function of the same name. The vxBase function preserves the first TWO characters before translating the remainder of the field into a numeric phonetic complement.

This algorithm results in table displays that more closely approximate alphabetical order than the traditional soundex algorithm which only preserves the first character of the field.
Example: SOUNDEX(cust_name) returns a 5 character string

SPACE(n)
Generate a string consisting of n spaces.
Would be generally used iin conjunction with the TRIM() function to create fixed length strings for use in vxBrowse and index expressions. See vxTableField for a practical example.
Example:

```
SUBSTR((TRIM(lastname)+", "+TRIM(firstname)+ SPACE(40)),1,40)
```

```
STR(Number, Len, Dec)
    Numeric to string function.
    Converts a number to a string representation of that number. Len is
          the number of characters in the new string, and Dec is the
          number of decimals.
    Note: If you wish to use a numeric field as an element in an index
          expression, always use the STR() function to convert the
          number into a string.
    Example:  STR(CURRENT+PAST_DUE,9,2) would result in "123456.78" if
          the sum of the fields CURRENT and PAST_DUE was equal to the
          number 123,456.78.
    Note: If the resulting number is too large for the allotted space,
          the string is filled with asterisks.

SUBSTR(Char_Value, Start, Length)
    Substring function.
    Returns a substring of the string represented by Char_Value.
    Example:  SUBSTR("abcdef,4,3") returns "def" (i.e., extract a
          substring from "abcdef" beginning with the fourth character
          for a length of 3)

TIME()
    Time of day function.
    Returns the system time as a character string in the form HH:MM:SS.
    Example 1:  TIME() returns 12:00:00 at noon
    Example 2:  TIME() returns 13:45:00 at one forty-five p.m.

TIMEODAY(Num_Value)
    Convert a number representing minutes to a display format time of
          day (i.e., am/pm).
    Example: TIMEODAY(510) returns " 8:30 am"

TRIM(Char_Value)
    Trim trailing (rightmost) spaces from a character field.
    May be used to logically concatenate fields in a browse column or in
          an index expression AS LONG AS the length of the resultant
          expression is FIXED.
    Example: See vxTableField for correct usage.

UPPER(Char_Value)
    Convert string to uppercase.
    Only alphabetic characters are affected.
    Should ALWAYS be used in index expressions to ensure correct
          collating sequence for character strings without regard to
          data entry formats.
    Example:  UPPER("abCD123g") returns "ABCD123G"
```

```
VAL(Char_Value)
    String to numeric conversion.
    Evaluation is terminated when a second decimal point, the first non-
          numeric character, or the end of the string is reached.
    Example 1:  VAL("23") returns 23
    Example 2:  VAL("12A12") returns 12
    Example 3:  VAL("-76.5") returns -76.5
    Example 4:  VAL("    12.12") returns 12.12
    Example 5:  VAL("12.   12") returns 12.00
    Example 6:  VAL("A12") returns 0

YEAR(Date_Value)
    Numeric year function.
    Returns the year in a date_value as a number.
    Example:  YEAR(DATE()) returns 1991 if the date is July 22, 1991
```

Sample Applications

There are two sample applications included with vxBase. The VXBTUT project is located in directory \VB\VXBTEST and shows you how to manage a single database using vxCtlBrowse as the primary database engine. It is fully commented.

The second sample application is more complex and attempts to give you examples of usage for almost every vxBase function call. It is described fully below.

Run vxload and then Visual Basic. Load project VXBTEST from the \VB\VXBTEST directory.

The sample application forms are designed for VGA/SVGA monitors using vxBase control drawing functions to give them a metallic, three-dimensional appearance. If you are running vxBase on a machine that does not have VGA capabilities, the appearance of the forms will not impress. Text on a gray background on an EGA monitor uses a different fill gray than the standard light gray that appears on a VGA screen, and the controls will all have a standard black border around them instead of a recessed or raised appearance.

The sample applications included with vxBase are intended to be used as templates for the developer in designing his own applications.

The source code is liberally sprinkled with comments. In some cases, more error checking would be required in a real application to provide a more stable product for the end user. Source code comments point out a number of these areas.

Almost all vxBase functions return a TRUE or FALSE value depending on the outcome of the operation. It is up to the individual programmer to decide just how much error trapping he would like to include. Some functions would fail only rarely (and only in the case of severely corrupted data). Such is the case with vxSkip(), for example, in a single user environment. In a multiuser environment, vxSkip() could be counted on to fail regularly when an attempt is made to access a record that another user has locked. In this case, vxBase will tell the end user that the record is locked and give him the opportunity to retry the operation or abort. What if he aborts? Now it is up to the programmer to decide on a strategy to take care of this eventuality.

The sample application is intended to illustrate the use of the vxBrowse function in controlling the logical flow of an application. It is used everywhere as a primary entry point for file editing and also as a help mechanism when the user is required to select a value from another file as input to a relational field.

Study the examples that set up visual relationships in a browse table that are accessed through the LINK menu in the sample application. This is a very powerful and unique function in the xBase world.

To institute a file editing application, use the VXFORM2 module as your first guide. This is a simple file consisting of two character fields that illustrates most of the techniques you will use to build

your own applications. More advanced techniques can be found in other modules.

The Problem

Our client is an aircraft brokerage firm who deals in used single engine aircraft. He does not maintain an inventory of airplanes. Rather, he solicits business from potential sellers, who usually are interested in selling their existing airplane and buying something else more upscale (or downscale depending on their current financial status). If he can find a buyer for the airplane, he receives a commission on the sale. The whole business is rather like real estate.

His problem is keeping track of what he has available for sale and remembering who was interested in it last month. In this sample application we are going to solve his problem.

First of all we build a sign-on screen. This is VXFORM0. The main controlling form will be VXFORM1. On it we will place all of the menu items we need to complete the application.

Note that this sample application doesn't do any printing. I'll leave that to you.

The Airtypes.Dbf File

The first thing we need is some way of categorizing the airplanes. We build a database of aircraft makes and models and assign simple three character codes to each type that we deal in. This file is critical to the whole operation. A buyer is interested in this or that category. Seller "A" is selling that category, and seller "B" is selling this category, so we can easily match them up.

Module VXFORM2 is used to maintain the airtypes file. Its file layout is as follows:

Field Name	Type	Length	Decimals	
category	C	3	0	user defined code
catname	C	35	0	make and model

This file is indexed on the CATEGORY field to file AIRTYPES.NTX.

Module VXFORM2 (Menu item File Types) does all the work of maintaining this file. This is an excellent place to start your investigation of vxBase because its as simple as it gets.

The Aircust.Dbf File

The next thing we need is some way to keep track of the names of our buyers and sellers. Instead of having two files (one for buyers and one for sellers), we can get away with just one. On the customer record we have logical fields telling us if the customer is a buyer and/or a seller.

Field Name	Type	Length	Decimals	
a_code	C	6	0	user defined code
a_name	C	40	0	his name
a_company	C	40	0	and company
a_address	C	40	0	street address
a_city	C	25	0	city
a_state	C	2	0	state/prov abbreviation
a_zip	C	10	0	postal code
a_phoneres	C	13	0	residence phone
a_phonebus	C	13	0	business phone
a_fax	C	13	0	fax
a_buyer	L	1	0	buyer?
a_seller	L	1	0	seller?
a_cdate	D	8	0	record creation date
a_rdate	D	8	0	record revision date
a_memo	M	10	0	memo reference

The file is indexed three ways:
(1) on a_code to aircust1.ntx
(2) on upper(a_name) to aircust2.ntx
(3) on a_state + a_code to aircust3.ntx

There is a supporting file for the state/provincial abbreviation (I'm Canadian so you'll have to put up with the province bit and probably some strange spelling). It simply contains the valid postal abbreviation for the state or province and the state/provincial name. We use it to validate data entry and also to provide a vxBrowse help example when the user is entering data in the a_state field. The file is airstate.dbf

Field Name	Type	Length	Decimals	
statecode	C	2	0	postal abbreviation
statename	C	20	0	name

This file is indexed on statecode to airstat1.ntx and on upper(statename) to airstat2.ntx. It was built with DataWorks, my xBase File Manager for Windows (which you've just got to have if you plan to do any serious development with vxBase: they go hand in glove).

Form VXFORM3 maintains the customer file. The customer file is accessed through the menu item File Customers.

The Airbuyer.Dbf File

If the user flags the customer record as a buyer, we enable the Buyer Records button on the form. If it is clicked, we can peruse and/or edit the buyer records attached to this customer. A buyer can be interested in more than one type of aircraft, and he may be willing to spend differing amounts on different types. We're setting up a many to one relationship with the customer record on the one hand and the Airtypes file on the other.

Field Name	Type	Length	Decimals	
b_code	C	6	0	customer code
b_cat	C	3	0	aircraft category
b_desc	C	35	0	make and model
b_low	N	8	0	low price range
b_high	N	8	0	high price range

The file is indexed on b_code + b_cat to airbuy1.ntx, and b_cat + b_code to airbuy2.ntx. We will use both sequences in our different joins when we try to match buyers to sellers or sellers to buyers.

The Aircraft.Dbf File

If the user flags the customer record as a seller, we enable the Aircraft Button on the customer form. Only one aircraft record is allowed to a customer. Clicking the Aircraft button on the customer form takes us directly to an aircraft description form (VXFORM5). This form is duplicated as VXFORM6 with different buttons for use with the Aircraft display module accessed by the File Aircraft menu item.

Field Name	Type	Length	Decimals	
c_code	C	6	0	customer code
c_nno	C	6	0	aircraft identifier
c_cat	C	3	0	aircraft type
c_desc	C	35	0	make and model
c_price	C	8	0	asking price
c_year	C	2	0	model year
c_annual	C	4	0	year-month annual due
c_ttsn	N	6	0	total time since new
c_smoh	N	4	0	time since major o/haul
c_spoh	N	4	0	time since prop overhaul
c_stoh	N	4	0	time since top overhaul
c_gwt	N	5	0	gross weight
c_ewt	N	5	0	empty weight
c_fuelcap	N	4	0	fuel capacity
c_net	N	8	0	net to broker
c_navcom1	L	1	0	1st of 16 avionics flds
...				which answer the
...				question "Is this
...				equipment installed?"
c_deice	L	1	0	last avionics field
c_memo	M	10	0	memo about the aircraft

The file is indexed on c_code + c_nno to aircraf1.ntx and c_cat + c_code to aircraf2.ntx.

<u>The Forms</u>

VXFORM0 is the startup form.
VXFORM1 is the menu form and system controller.
VXFORM2 is the Airtypes record editing form.
VXFORM3 is the Aircust record editing form.
VXFORM4 is the Airbuyer record editing form.
VXFORM5 is the Aircraft record editing form.

VXFORM6 is the Aircraft detail display form.
VXFORM7 is a sample form that shows you how to extract xBase field and file details using vxBase commands.
VXFORM8 is an example of using the vxRecord function to extract the contents of an xBase record.
VXFORM9 shows how to use vxCtlFormat to control data entry and data verification in Visual Basic text boxes.
VYFORM0 is an example of using database alias names and vxSetRelations.
VYFORM1 shows you how to use printer enumeration and printer selection functions.
VYFORM2 uses a single column vxCtlBrowse with a linked picture box to illustrate bitmap displays.

The Link Menu

These functions show off the power of vxBrowse and visual joins to best advantage. Bring up the Buyers to Sellers item and hit the Join menu item. Its magic! With an absolute minimum of effort we can link potential buyers to sellers and vice versa.

Running the Sample Application

The name of the project is vxbtest.mak. It should reside in the \vb\ vxbtest directory you were asked to set up when you installed vxBase. Remember to run vxload.exe before starting Visual Basic. Open the vxbtest project and run, or make an .EXE and run it.

Tips and Techniques

Entry and Exit Strategies

Please study the methods of form loading/unloading and exit procedures in the sample application and emulate these methods in your own application. Remember that in a Windows environment we can shut down a running application from a number of areas - your own Exit menu item, the application's system menu, or even shut down Windows entirely while your application is running. It is imperative that xBase files that have undergone changes are closed properly to ensure no loss of data, header information, or index corruption. Always include a vxInit call as the first statement in your vxBase application. Always include a vxDeallocate call as the last statement before terminating your Visual Basic application. This call ensures that memory allocated by vxBase is released when in design mode and that it is safe to unload the application when running as an .EXE.

The sample application allows the form with the system menu on it to remain visible. We use global flags that are set when a form is loaded and reset when it is unloaded to test whether there are any active forms running when an exit is taken from this top level window.

Access to Form Menus

vxBase requires parent windows to draw upon. If your Visual Basic parent form contains menus, remember that the menu items will be available to the user when a vxBrowse table is being shown and program accordingly. For example, it would be foolhardy to pack a file that was already open and unlocked and being displayed in a browse table. The application WILL crash when the user attempts to access the browse table again. See the sample code in vxPack for a method of checking the open status of files before performing critical operations on them. You can also disable menu items temporarily before beginning the browse. Almost every sub-function in the sample application disables one or more menu items. Alternatively, use modal forms in critical operations that could be led astray by other program functions.

Data Entry

xBase programmers have become accustomed to a get system that effectively defines what data is entered, how it looks, and how long it is. vxBase provides a somewhat similar facility thorugh the vxCtlFormat funcion.

The sample application has many examples of manual data validation as an alternate method. The sample uses these methods attached to each edit control to achieve some logical flow for you as the programmer to use as a guide without getting lost in a maze of global subroutines.

Most of the methods would be better served as global functions. Over time, you should be able to build your own library of data validation routines to make life simpler for your next application.

Particularly examine the GotFocus and KeyPress events attached to the various edit controls in the sample application. I think you'll get some good ideas there for limiting data entry length, case conversion,

and numeric validation.

Logical fields have been much ignored among xBase programmers but I think they'll make a comeback considering how effective they are in controlling Windows' check boxes and radio buttons.

Release 1.07c: See the new vxCtlFormat Function to control data entry and validation.

Parents for vxBase Windows

Windows created with vxBase functions (vxBrowse and vxMemoEdit) absolutely require an active Visual Basic form to act as parent. Their default sizes are calculated based upon the size of the active window.

Data Paths

The sample application has data paths for the files hard coded into each vxUse. You would be well advised to set up a system that solicited a path that you could save and prepend to each file name for each command that requires it. vxBase acts like other xBase systems in that it does not find data files that are simply in the system path. You have to tell vxBase where the files are.

Controlling Multiple Windows

vxBase maintains an internal task-window manager that registers database select areas with windows if certain rules are followed. Always include a vxSelectDbf (or vxUseDbf) statement accessing the first database you will be working on in any form as the first statement in the Form_Load procedure and as the first statement in the Form_Paint procedure (see Multitasking issues discussed below). The Form_Load select registers the database as the default for the application; the Form_Paint select registers the database with the window.

If you are going to leave a window visible that contains access to menu items (as in the sample application), carefully disable menu items that could adversely affect the data currently displayed on the form. For example, if you had a record editing form visible for FileX, you would not want the user to select a pack or reindex item from a background menu that could compromise the status of the current file (especially if the pack or reindex function closes the file when it terminates).

You should also always disable the menu item that brought you to the current form or make the current form a modal one. Always remember that you don't know what the user is liable to do, so disable those functions that could compromise your current position.

Browse Windows

vxBrowse windows contain their own message loops. Conflicts could arise when running a vxBase application in Design Mode. For example, when a browse window is displayed in VB Design Mode, it is possible to click on the VB Menu Window and select RUN again (because vxBrowse is not dependent on the VB controlled message loop). If you do this, the system will hang.

Dataworks

Dataworks is a dictionary based xBase file management system for Windows that allows you to interactively create dbf/ntx files, import your own files, display, join, modify xBase structures, and so on. It is an excellent visual additive tool for the vxBase programmer - much like the dBase dot prompt was to a whole generation of xBase programmers. It was written by the same author as vxBase and is available for $49.95. An unregistered shareware version is probably in the same library if you obtained vxBase from a bulletin board - the name of the file is dworks.zip. See the form at the back of this manual for ordering information.

MultiTasking and Multiuser Considerations

MultiTasking

vxBase supports multitasking. You can run a number of applications using vxBase all at the same time. You can run multiple instances of the same program. You can have multiple windows visible each accessing a different database (in the same instance of the program) or the same databases (in multiple instances of the same program or other programs). As a programmer, you don't know what the user is liable to do. He can easily compromise a database by injudicious use of the Windows multitasking environment. You can make every effort to disable menu items that could harm the current window data, but these efforts could be circumvented by a user playing with multiple instances of your program. You may wish to limit Windows by only allowing one instance of your program. You can do this by implementing a window test scheme in the form load procedure of the first form in your application.

If you don't wish to place artificial limits on the user, you may wish to create separate file maintenance programs for packing and reindexing files that won't run if your main application is running. This is probably your best course of action.

The first call to vxBase from your application must be to vxInit and the last statement in the application must be a call to vxDeallocate. vxInit registers the task with a multitask list maintained by the DLL. If the task registered is the first to load vxBase, it controls the database memory that will be shared by all other vxBase tasks running at the same time. This task must be unloaded last. If it unloaded prior to other concurrently running vxBase tasks, all the database memory goes with it - and the other vxBase tasks crash with an Unrecoverable Application Error or General Protection Fault as soon as their windows get the focus. We ensure that it is unloaded last by calling vxDeallocate when we are making an exit. If it is the controlling task, and there are other tasks using vxBase that were loaded after it, the user is informed via an error message box that a task closure sequence error has occurred. It is up to the programmer to test the result of vxDeallocate to determine whether we can safely unload the task. See the writeups in the function section of this manual for vxInit and vxDeallocate for examples of correct procedures.

If running your program in Visual Basic Design Mode, see the section entitled "Visual Basic and VXLOAD.EXE".

To implement a vxBase application in a multitasking environment, vxBase places minimal restrictions on the programmer other than the vXinit and vxDeallocate calls described above. The user might have two or more windows open each displaying different data and he may move back and forth between them at will. In a normal xBase application, there may be only one active database select area. In a Windows environment, however, we may have three or four or five active windows with different databases represented in each, representing the same program or different programs (a task). Every time the user moves from one open window to another, as a programmer in the old xBase tradition, you would have to ensure the proper database was selected. vxBase removes this

onus by maintaining a task-window-select area table that automatically selects the correct database when the window controlling that database receives the input focus. vxBase also maintains a default select area for the task that it uses to register databases with windows if no database has been selected for that window. As a programmer, you are required to insert three vxBase calls into every separate form procedure that accesses a database:

(1) vxSelectDbf() or vxUseDbf() the first database accessed by the form as the first line in the Form_Load procedure. This registers the database as the default database for the task.
(2) vxSelectDbf() the first database accessed by the form as the first line in the Form_Paint procedure. This registers the database with the window associated with the current task.
(3) issue the vxWindowDereg command in your form unload procedure to remove the task-window-select entry from the vxBase task management table. This table is limited to 96 entries and could overflow if you fail to deregister the windows. Issue the vxWindowDereg command after closing any databases you wish to close in the Form_Unload procedure.

vxSelectDbf or vxUseDbf register databases with windows. These are the only two vxBase functions that register databases with windows.

vxBase field functions should also have field names qualified with an alias (see vxSetAlias). The alias names ensure proper database selection in all circumstances.

If printing a report from a database, always reselect that database after the Printer.EndDoc has been issued because the database becomes attached to the print task if any vxBase functions are called from within the body of the print routine.

While testing, if you get an "Invalid field name" error message from vxBase, and you know the field exists in the database (i.e., the name is correctly spelled), in all likelihood the wrong database is active because of vxBase's automatic selection. To correct the problem, simply insert a vxSelectDbf statement for the database you want in front of the statement that conatins the field reference or set up an alias name for the database with vxSetAlias. Some things go on in the background that you are hardly aware of (e.g., Form painting), and if you have the required select statement in the Form_Paint procedure then a reference to a field in another database may be invalid if a Form Paint has taken place since you last accessed the file you thought was still active. See the code examples in the sample application for the Help buttons (e.g., CustStateHelp in VXFORM3) for a perfect instance of the above. Disabling and then re-enabling a background form for a help browse causes a Form_Paint message to be issued, which selects a different database than the one we just used, so we have to re-select the database to access its fields.

If the database has been registered with a window, any call to a vxBase function that accesses a database will result in a search in the task management table for the window id of the window that currently owns the input focus. If found, the database is automatically selected.

If no entry is found, the database selected will be the task default. The user may then have multiple forms open and switch between them at will. It doesn't matter to you as the programmer which window is selected or where the program instruction pointer is residing when the user switches to another window. The correct database is automatically selected if the simple rules outlined above are followed.

Windows allows up to 20 file handles per task (15 useful handles). Use the vxSetHandles function to increase the number of file handles available to a task if you will have more than 15 files open simultaneously. Only one select area is assigned to a database in any given task (unless a file is opened an another area with vxUseDbfAgain). The select area contains critical information about the state of the database (e.g., current record number, filter, table definition, etc.). Opening the file in the same task again will change this information for the subtask that opened the file in the first place. Bear this in mind and disable access to functions that could change the state of the database when you don't want it changed. Use the sample application as a guide.

The same database may be opened in different tasks (multiple instances of the same program are different tasks as well) and each different task gets its own select area for the database. Changes made to records by one task are reflected in the other task as soon as the records come into view. Records currently in view, such as in a browse table, won't reflect the changes until the view window has been repainted. Because each task has its own select area, changes to record positions, tables, etc. in one task do not affect the state of the database in the other task (or tasks).

MultiUser

On a local area network, many workstations can run the same Visual Basic program using vxBase at the same time, all accessing the same files on a network drive. Obviously, there are no internal conflicts between the allocated memory buffers residing on the individual workstations. There may, however, be file and record conflicts when more than one user attempts to access the same record (or file) as another user.

Always install vxbase.dll on each workstation local drive rather than on the server drive.

If vxBase applications are going to run concurrently with other xBase applications (e.g., those written in Clipper), always use vxWriteHdr after appending a record in vxBase to ensure that the other applications are kept informed of the state of the database.

Traditional (i.e., Clipper style) record and file locking is provided by vxBase by using the vxSetLocks(FALSE) function. vxBase file and record locking is perfectly compatible with Clipper (version 5.1 and below). Clipper recognizes vxBase locks and vice versa.

If you use the default vxSetLocks setting (which is TRUE), it is not necessary for the user or the programmer to be concerned with explicit

file and/or record locking, although these functions are provided as part of the vxBase command set. Commands that obviously require a file lock (such as vxPack or vxReindex) are automatically locked by vxBase during the processing of the command. Records that occupy any given workstation buffer are also automatically locked if vxSetLocks() is TRUE, as opposed to Clipper (or vxSetLocks(FALSE)) which allows simultaneous access to the same record and therefore also allows simultaneous updating of the record while it resides in each workstation's record buffer. In this case, the last update always wins and the user who wrote the record out first loses his changes.

In a multiuser environment, it is usually necessary to provide a network signature flag on any record that could be affected by simultaneous updates. The signature is simply a number that is incremented each time the record is updated. When a user reads in the record for updating, he saves the contents of the signature field and he moves the contents of every other field in the record to working storage. When the update on the working storage variables is finished, it is necessary to re-read the record and check to see that the signature field has not changed since he first read the record. If it is the same, he locks the record, replaces required fields with his changed data, increments the signature field, and then unlocks the record. If the signature field had changed since he first read the record, it would be necessary to re-do the update because the other user could have changed sensitive data.

With vxSetLocks(TRUE), any record currently occupying a workstation buffer automatically locks out other users from accessing that record. The programmer must be aware of this fact when designing a system for multiuse. A signature system such as the one described above could easily be implemented as follows:

```
If vxSeek("ABC") Then       ' find the record to update
   RecNum& = vxRecNo()          ' save the record number
   Sig% = vxInteger("CustSig") ' and the signature
   Name.text = vxField("Name")  ' store the form vars
   Status.text = vxField("Stat")

   ' now unlock the record
   ' ---------------------
   j% = vxUnlock()

   ' now perform the update on the vis basic form
   ' --------------------------------------------
   CustRecordUpdate

   ' now retrieve the record and test if anyone else
   ' has changed it
   ' -----------------------------------------------
   j% = vxGo(RecNum&)
   If Sig% <> vxInteger("CustSig") Then
      MsgBox "Another user beat you to it. Redo!"
   Else
      Call vxReplString("Name", (Name.text))
      Call vxReplString("Stat", (Status.text))
      Call vxReplInteger("CustSig", (Sig% + 1))
   End If
   j% = vxUnlock()
End If
```

The only real difference between a Clipper implementation (vxSetLocks(FALSE)) and the vxBase default lock procedure (vxSetLocks(TRUE)) is that with the TRUE locks you must explicitly unlock the record instead of locking it. If you fail to do so, other users even attempting to browse in the same area of the file will have to wait until the user who has the locked record finishes his update.

The sample code attached to VXFORM2 contains complete protocols for unlocking the database in a multiuser environment when the default locking mechanism is used. Signature fields are not used, however, for simplicity's sake. Bear in mind that for a robust multiuser system they should be attached to all master files that could be affected by simultaneous updates.

Note that records displayed via a vxBrowse table are not locked. Only when a selection has been made from a vxBrowse table does a locked record occupy the workstation buffer space if vxSetLocks is TRUE.

Files may be accessed in Read Only mode. If network server files are marked with the read only attribute for a specific user, vxUseDbf will fail. You must use vxUseDbfRO to open files for reading only. Note that files opened with vxUseDbfRO are not required to have the read only attribute. vxBase will simply not perform any function that produces a file write on the database or any of its related files (i.e., index and memo files).

Share

If attempts at running vxBase applications on a LAN fail with file/record locking errors, or if they fail on a single workstation when vxSetHandles is used, ensure that SHARE.EXE is loaded by each workstation that will be running vxBase.

International Issues

The following functions all deal with the problem of a database that contains characters from the high end of the ANSI or OEM character sets, which is commonplace if the database stores data in a language other than English.

vxSetAnsi(FALSE) properly handles databases that were created with a DOS based application (such as Clipper). These databases are OEM databases. Characters with diacritical marks in the high end of the OEM character collating sequence are NOT the same as the ANSI characters. It is necessary for vxBase to translate the characters to ANSI (both Windows and vxBase native mode) before they can be used in a vxBase application. They also must be translated back again when they are written.

The default value of vxSetAnsi is TRUE (no translation takes place). If the database was created and is maintained by vxBase (or DataWorks), and the database is going to be used exclusively by Windows applications, vxSetAnsi should be TRUE.

vxCollate allows the programmer to create his own collating sequence table (for EITHER an ANSI database or an OEM database). The OEM character set in particular does not use any kind of logical collating sequence for characters with diacritical marks. The ANSI table handles these characters more intelligently - but its sequence is also incorrect for some languages.

It is necessary to define a collating sequence table to properly build an index that uses these characters.

vxSetCollate can toggle a defined collating table on or off.

vxAppendBlank

Declaration

```
Declare Function vxAppendBlank Lib "vxbase.dll" () As Integer
```

Purpose

Append a blank record to the physical end of the database file in preparation for using the vxReplx functions to replace the fields with your data.

Parameters

None.

Returns

TRUE if record successfully appended.

FALSE if not successfully appended. Always FALSE if the file was opened as Read Only with vxUseDbfRO.

Usage

Always append a blank record to receive fields for a new record that is being inserted into the database. If you forget to do this, the current record will be changed instead.

Always close the database before exiting your application. Use vxCloseAll in your exit routine to ensure that all records are flushed to disk and the xBase header is updated correctly.

Multiuser Considerations

All active index files associated with the selected database are locked until the record is written. You must explicitly lock the appended database record. The record is written either by performing an explicit vxWrite command or implicitly by performing some other action on the file such as vxClose, vxSkip, or vxGo.

Example

```
If AddMode Then
  If Not vxAppendBlank() Then
         MsgBox "Append Error"
  Else
     j% = vxLockRecord()
          Call vxReplString("Field1","New Field")
     j% = vxWrite()
     j% = vxUnlock()
  End If
End If
```

See Also

vxWrite, vxReplxxx

vxAppendFrom

Declaration

Declare Function vxAppendFrom Lib "vxbase.dll" (ByVal FromFile As String) As Integer

Purpose

Append all of the records from the named database onto the currently selected database.

Parameters

FromFile is either a string variable that contains the name of the file that will be appended from (including an optional path specification) or a literal string. If no file extension is supplied, vxAppendFrom defaults to ".dbf". This file does not have to be open for the operation to succeed. If it is open, it will be closed when the function returns to your program.

Returns

TRUE if the operation was successful or FALSE if it was not. Always FALSE if the target file was opened as Read Only with vxUseDbfRO.

Usage

Useful for processing transactions in a batch and then, after verification, appending the transactions to a master file. For example, in a general ledger application, it would be commonplace to collect transactions in a batch. The user could enter and edit transactions at will in one or more sessions. When the user decides to post the transactions, they would then be applied to the general ledger, added to the master transaction file with the vxAppendFrom function, and then the records in the batch file would be deleted to protect the integrity of the audit trail. In this case, the structure of the transaction batch file would probably be the same as the structure of the master file.

This function would also be used when transferring fields from one file to fields that have the same name and type in another file. Any fields in the From file that match in name and type to fields in the current database are appended record by record to the current selection. Truncation in the receiving file occurs on the right for character fields and on the left for numeric fields if the lengths of the fields differ. If the field is numeric and the number of decimals differs, truncation occurs on the right if the number of decimals in the receiving field is less than the sending field. Memos and bitmaps stored in FromFile dbt files are also appended to the receiving file dbt.

Files that duplicate current structures may also be dynamically created at run time with the vxCopyStruc function, used as batch files, appended to master files, and then deleted.

Note that filters on either the FromFile (if it happens to be open) or on the currently selected database have no effect. All records, including deleted records, in the FromFile are appended.

Multiuser Considerations

Both databases are locked for the duration of the operation. When the function completes, the current selection is the same as on entry, and the record pointer is pointing to the top record in the file, which is locked if vxSetLocks is TRUE.

Example

```
' open transaction batch file
' ---------------------------
TransDbf% = vxUseDbf("Transbat.dbf")
TransNtx% = vxUseNtx("Transbat.ntx")
j% = vxDbfSelect(Transdbf%)

' call transactions editing procedure
' ----------------------------------
CollectTrans

' if posting now, append transactions to
' master file after they have been posted
' and then clear the batch file in preparation
' for the next editing session
' ------------------------------------------
j% = MsgBox("Post Now?", 52)
If j% = 6 Then
   PostTrans
   TrMasterDbf% = vxUseDbf("Transmas.dbf")
   TrMasterNtx% = vxUseNtx("Transmas.ntx")
   j% = vxSelectDbf(TrMasterDbf%)
   j% = vxAppendFrom "Transbat.dbf"
   j% = vxClose()      ' close master file

   ' reopen transaction batch because the From
   ' file is closed by vxAppendFrom
   ' -----------------------------------------
   TransDbf% = vxUseDbf("Transbat.dbf")
   TransNtx% = vxUseNtx("Transbat.ntx")
   j% = vxDbfSelect(TransDbf%)
   j% = vxZap()        ' clear the batch
End If
j% = vxClose()              ' close the batch
```

See Also

vxCopy, vxCopyStruc

vxAreaDbf

Declaration

Declare Function vxAreaDbf Lib "vxbase.dll" (ByVal DbfName As String) As Integer

Purpose

Extracts the select area assigned by vxBase to the named database file when it was opened with vxUseDbf. The select area is an integer greater than zero.

Many vxBase functions take a select area as a parameter when identifying a dbf and its related ntx files instead of using the file name.

This function would be used primarily to test the open status of a .dbf. Under normal conditions, the select area integer is assigned to a global variable when the file is opened with vxUseDbf.

Parameters

DbfName is either a string variable that contains the name of the file (including an optional path specification) or a literal string. If no file extension is supplied, vxAreaDbf defaults to ".dbf".

Returns

An integer identifying the select area of the named file that was assigned by vxBase when the file was opened with vxUseDbf. A number > 0 identifies an open file. If the file is not open, FALSE is returned. Note that you cannot test the return value with a NOT expression because a number greater than zero is NOT TRUE (but neither is it FALSE) according to Visual Basic. Store the return value in a variable and explicitly test it for FALSE.

Note that this function will return the lowest numbered select area of a dbf if it has been opened more than once (with vxUseDbfAgain on subsequent opens).

Usage

Use the returned value to check on whether a file is open or not.

This function may be used to test the open status of a file that is about to undergo a critical operation (such as vxPack or vxReindex). A FALSE return indicates that no active task (not just the current one) has the file open.

Note that since this function returns the area selected by any task that is active, you cannot rely on it to return a value that you can use in your application.

Example

```
' See if file is open before packing
' --------------------------------
NamesDbf% = vxAreaDbf("c:\database\names.dbf")
If NamesDbf% = FALSE Then
   j% = vxUseDbfEX("c:\database\names.dbf")
   j% = vxPack()
   j% = vxClose()
Else
   MsgBox "File is open. Function aborted."
End If
```

See Also

vxDbfCurrent, vxPack, vxSelectDbf, vxSetHandles, vxUseDbf, vxUseDbfAgain, vxUseDbfEX, vxUseDbfRO

vxAreaNtx

Declaration

Declare Function vxAreaNtx Lib "vxbase.dll" (ByVal NtxName As String) As Integer

Purpose

Extracts the select area assigned by vxBase to the named index file when it was opened with vxUseNtx. The select area is an integer greater than zero.

Many vxBase functions take a select area as a parameter when identifying an open index instead of using the file name.

This function would be used rarely. Under normal conditions, the select area integer is assigned to a global variable when the file is opened with vxUseNtx.

Parameters

NtxName is either a string variable that contains the name of the file (including an optional path specification) or a literal string. If no file extension is supplied, vxAreaNtx defaults to ".ntx".

Returns

An integer identifying the select area of the named file that was assigned by vxBase when the file was opened with vxUseNtx. A number > 0 identifies an open file. If the file is not open, FALSE is returned. Note that you cannot test the return value with a NOT expression because a number greater than zero is NOT TRUE (but neither is it FALSE) according to Visual Basic. Store the return value in a variable and explicitly test it for FALSE.

Usage

Use the returned value to check on whether the file is open or not.

Note that since this function returns the area selected by any task that is active you cannot rely on it to return a value that you can use in your application.

Example

```
NamesNtx% = vxAreaNtx("c:\database\names.ntx")
If NamesNtx% = FALSE Then
  MsgBox "NAMES.NTX is not open"
End If
```

See Also

vxAreaDbf, vxNtxCurrent, vxSelectNtx, vxSetHandles, vxUseNtx

vxBlobRead

Declaration

```
HANDLE FAR PASCAL  vxBlobRead (memofieldname)
char*  memofieldname;          /* name of memo field holding blob */
```

Purpose

Read a binary large object (blob) from a memo file that was stored with vxBlobWrite.

vxBlobRead and vxBlobWrite are primarily for the c programmer. To handle bitmaps in Visual basic, see vxPictureImport and vxPictureRead.

Parameters

memofieldname is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *memofieldname* may be qualified with a valid alias name that points to any open database.

vxBase uses the large memory model. All char * are far pointers and must be explicitly cast as such if you are using a different memory model.

Returns

A HANDLE (HGLOBAL) to global memory allocated by vxBase. The memory contains the blob. It is the programmer's responsibility to release the memory with GlobalFree when he is done with it.

Usage

Any binary object may be stored in a memo field with vxBlobWrite and a HANDLE (HGLOBAL) to that object may be extracted with vxBlobRead. The vxPicture functions are limited to standard BMPs and variants thereof.

Example

```
/* ****************************************** */
/* DrawBlob uses a GIF library to place an       */
/* image into the window passed to the function  */
/* The blob is retrieved from the memo fieldname */
/* stored in the current record                  */
/* ****************************************** */
BOOL DrawBlob(HWND hwnd, char *fieldname)
{
  HANDLE     hPicBuffer;      // mem handle to blob
  LPSTR      lpPicBuffer;     // string addr of blob

  // get the handle to the blob
  // if NULL, return FALSE
  // --------------------------
  if (NULL == (hPicBuffer = vxBlobRead(fieldname))
     return(FALSE);

  // convert the handle to a string address
  // --------------------------------------
  lpPicBuffer = GlobalLock(hPicBuffer);
```

```
  // Draw the image
  // --------------
  if (!GifConverter(hwnd, lpPicBuffer))
     {
     GlobalUnlock(hPicBuffer);
     GlobalFree(hPicBuffer);
     return(FALSE);
     }

  // release the memory
  // ------------------
  GlobalUnlock(hPicBuffer);
  GlobalFree(hPicBuffer);
  return(TRUE);
}
```

See Also

vxBlobWrite, vxIsPicture, vxPictureImport, vxPictureRead, vxMemoClear

vxBlobWrite

Declaration

```
int FAR PASCAL  vxBlobWrite (hGlobal, memofieldname)
HANDLE hGlobal;           // global handle to mem storing blob
char*  memofieldname;     // name of memo field to put it in
```

Purpose

Write a binary large object (blob) into a memo file.

vxBlobRead and vxBlobWrite are primarily for the c programmer. To handle bitmaps in Visual basic, see vxPictureImport and vxPictureRead.

Parameters

hGlobal is a handle to a global memory object that stores a binary large object.

memofieldname is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *memofieldname* may be qualified with a valid alias name that points to any open database.

vxBase uses the large memory model. All char * are far pointers and must be explicitly cast as such if you are using a different memory model.

Returns

TRUE if the operation was successful and FALSE if not.

Usage

Any binary object may be stored in a memo field with vxBlobWrite and a HANDLE (HGLOBAL) to that object may be extracted with vxBlobRead. The vxPicture functions are limited to standard BMPs and variants thereof.

vxBase does not free the global memory after writing the blob. That is the programmer's responsibility.

Example

```
/* ****************************************** */
/* GetBlob stores a GIF image into a memo field   */
/* The blob is retrieved from the filename passed */
/* to this function                               */
/* ****************************************** */
BOOL GetBlob(char *filename, char *fieldname)
{

  int        hFile;           // dos file handle
  DWORD      dwLength;        // import file length
  HANDLE     hPicBuffer;      // mem handle to blob
  LPSTR      lpPicBuffer;     // string addr of blob

  //  verify existence of import file
  // --------------------------------
  if ((hFile = myFileOpen(filename, 0) < 0)
     return(FALSE);

  // if zero length, set error
  // -------------------------
```

```
  if (!(dwLength = myFileLength(hFile))
     return(FALSE);

  // if unable to allocate memory, set error
  // ---------------------------------------
  if (NULL == (hPicBuffer = GlobalAlloc(GHND, (DWORD)dwLength)))
     {
     _lclose(hFile);
     return(FALSE);
     }

  // read image
  // ----------
  lpPicBuffer = GlobalLock(hPicBuffer);
  _hread(hFile, lpPicBuffer, (DWORD) dwLength); // huge read
  _lclose(hFile);

  // write image to memo file
  // ------------------------
  GlobalUnlock(hPicBuffer);
  if (!vxBlobWrite(hPicBuffer, fieldname))
     {
     GlobalFree(hPicBuffer);
     return(FALSE);
     }

  GlobalFree(hPicBuffer);
  return(TRUE);
}
```

See Also

vxBlobRead, vxIsPicture, vxPictureImport, vxPictureRead, vxMemoClear

vxBof

Declaration

```
Declare Function vxBof Lib "vxbase.dll" () As Integer
```

Purpose

Test if beginning of file has been reached in the currently selected database.

Parameters

None.

Returns

TRUE if an attempt was made to skip backwards beyond the first record in the file. Otherwise FALSE.

Usage

When skipping through a file backwards, always use vxBof to test if the top of the file has been reached. Once the condition has been satisified, it remains true until the record pointer is repositioned with a call to vxGo, vxTop, vxBottom, vxSkip, or vxSeek. It is never possible to skip to a record prior to the first record in the file. If vxBof is true, the record buffer will contain the elements of the first record. (It is possible, however, to skip beyond the end of the file to an empty record buffer.)

Example

```
' skip back one record
' ----------------------
Do
  j% = vxSkip(-1)
  If j% = FALSE Then
     MsgBox "Error on Skip Previous. Try Reindex."
     Exit Sub
  End If
  If vxBof() Then Exit Do
Loop Until Not vxDeleted()

' test for beginning of file
' --------------------------
If vxBof() Then
  Beep
  TypeStatus.text = "Beginning of File!"
  j% = vxTop()   ' make sure we've got a record
Else
  TypeStatus.text = "Skipped to " + LTrim$(Str$(vxRecNo()))
End If
```

See Also

vxEof, vxSkip

vxBottom

Declaration

Declare Function vxBottom Lib "vxbase.dll" () As Integer

Purpose

Position record pointer to the last record in the currently selected file. If an index is active, this is the last logical record in the file. If no index is in use, it is the last physical record in the file.

Parameters

None.

Returns

TRUE if the attempt was successful. Otherwise, it is FALSE. A FALSE condition can occur on an empty database or on a file with a corrupted index.

Usage

Useful when your program requires a forced end of file condition. See the example below.

If a filter is active, vxBottom will attempt to find the last record in the file that satisfies the filter.

Multiuser Considerations

If vxSetlocks(TRUE), then the last record in the file is locked.

Example

```
If vxSeek("ABC") Then
   Do While Not vxEof()
      j% = vxSkip(1)
      If vxField("CustCode") <> "ABC" Then
         PrintTotals
         j% = vxBottom()      ' Exit Do would work
         j% = vxSkip(1)       ' just as well but this is
      Else                    ' an example
         PrintRecord
      End If
   Loop
End If
j% = vxUnlock()
```

See Also

vxSetLocks, vxTop

vxBrowse

Declaration

Declare Sub vxBrowse Lib "vxbase.dll" (ByVal Hwnd As Integer, ByVal DbfArea As Integer, ByVal NtxArea As Integer, ByVal EditMode As Integer, ByVal AllowFilter As Integer, ByVal EditMenu As Integer, ByVal StartRec As Long, ByVal Caption As String, RetVal As Long)

Purpose

Create and display a table of records using the defined database and index. This is a very powerful function that eliminates the need for a grid control or huge arrays to display a data table. Combined with the vxTable functions and the vxJoin function it gives the programmer an extremely useful tool with little effort.

For a variation of vxBrowse, see vxCtlBrowse as well. vxBrowse runs in its own window and must be closed before any user actions performed on the browse can be evaluated by the programmer. vxCtlBrowse allows the use of a browse grid within a form control. In vxCtlBrowse, the user and the programmer can retrieve information from and direct the actions of the browse window interactively.

Parameters

Hwnd is the hWnd property of an active window which assumes the role of parent to the vxBrowse window. There must be an existing form to act as a reference point for the browse window.

DbfArea is the select area of an already opened database. If it is not currently selected, vxBrowse will make it the current selection. It will be the current selection when vxBrowse returns as well.

NtxArea is the select area of an index file attached to the DbfArea. If you do not wish to browse with an index, pass a 0 (zero) to the function.

EditMode is passed as TRUE or FALSE. If TRUE, when the user double clicks on any column in the table, the field attached to that row and column is presented for update. **If the user presses the ENTER key, the field is replaced and the record is written. If the user presses the ESC key, the update is cancelled.** Note that the only data validation possible with the onscreen edit is for type (i.e., numeric fields must contain numbers, etc.). If your data requires more sophisticated validation, never pass a TRUE to this function. If *EditMode* is FALSE, doubleclicking on a record will return the selected record number (which is the same result as Edit Update or pressing the ENTER key). If *EditMode* is TRUE, it would probably be a good idea to add the words "Edit Enabled" to your browse window caption to alert the user that onscreen editing is active.

If a vxTableDeclare has been issued to control your browse display, any column defined as an expression rather than as a field will not be available for edit (obviously). You can use this fact to your advantage if you wish to limit onscreen editing to only a few fields. All of the fields which would have editing disallowed could be defined in the table as expressions rather than fields (e.g., instead of displaying field

"category", you could define the column to display "substr(category,1,3)" (assuming the length of field "category" is 3), which would effectively rule out any editing on that field, or you could simply tell vxBase that the item is an expression with the VX_EXPR parameter (see vxTableField for more information).

AllowFilter is passed as TRUE or FALSE. If TRUE, an item on the vxBrowse menu will allow the user to invoke a dialog box that accepts a standard xBase expression as a filter string. If the expression passes the evaluation test (and that test ensures that the expression returns a logical result), then the filter will be applied to the current browse table. For example, areacode = '403' would be a valid filter expression if the file contained a character field named "areacode". The table would then only contain records whose areacode matched "403". Note that this filter applies only to the active browse window. It goes away when the window is closed and will not affect any program logic. It will, however, override any filter set by vxFilter before the browse is invoked. When the window is closed, the old vxFilter expression will once again take effect. If AllowFilter is FALSE, the user is not allowed to enter a filter when browsing. vxBrowse always filters out deleted records.

Use filters judiciously. A filter can slow the vxBrowse display in a large file enormously. See vxFilter for more details.

EditMenu is passed as TRUE or FALSE. If TRUE, an Edit menu item is presented on the vxBrowse menu bar. The Edit menu contains Update, Add, and Delete selections. If any of these are selected by the user, a code is passed back to the program in the *RetVal* parameter (see below) informing the program what the user wants to do. These three items are standard fare in maintaining files. If you are going to use the vxBrowse table as display only, or as a help window, then *EditMenu* would be passed as FALSE. It should also be passed as FALSE if you use vxMenuDeclare and vxMenuItem to define your own browse menus.

StartRec is a long integer that contains the starting record number for the browse. If passed as 0 (zero), then the record pointer is positioned to the first record in the file (either logical or physical depending on whether an index was specified or not). If you are interested in a subset of records in the file, it is your responsibility to position the record pointer to the first one that meets your criteria before beginning the browse. See the sample code attached to VXFORM3 (Proc BuyRecs_Click) for an example of using vxBrowse to display a record subset. If an invalid *StartRec* is passed, the browse will begin at the first record in the file.

Caption is a string that is used as a Window caption for the vxBrowse table.

RetVal must be dimensioned as a long integer before the browse commences. The result of the browse is passed back to the program in this parameter. Usually, the programmer will set up a number of GLOBAL RetVals (one for each file that will be browsed) and use these as prime movers in his logical flow. Study the code in VXFORM2 and the use of the TypeReturn variable to control the flow of logic surrounding the

AirTypes file.

The values returned in *RetVal* are defined as Global constants in the vxbase.txt file.

BROWSE_CLOSED: The user closed the window with the System menu or Alt-F4. He doesn't want to do anything with this browse.

BROWSE_EDIT: The user selected the Update function from the Edit menu. The record pointer is positioned at the record that was highlighted on the browse table immediately prior to the menu selection.

BROWSE_ADD: The user selected the Add item from the Edit menu. The record pointer is positioned at the record that was highlighted on the browse table immediately prior to the menu selection.

BROWSE_DELETE: The user selected the Delete item from the Edit menu. No action is taken by vxBase on the selection. Instead, it is the programmer's responsibility to ensure that the delete is handled properly. This usually involves a confirmation window and cross-referencing logic to remove related records from other files. The record pointer is positioned at the record that was highlighted on the browse table immediately prior to the menu selection.

BROWSE_ERROR: An error occurred when attempting to start the browse. For example, the defined database or index area is invalid.

In addition to these constants, BROWSE_USER is also defined to handle circumstances known only to the programmer. BROWSE_USER could be used if the *RetVal* parameter is indeed the prime mover behind your logical flow. See an example of its use in the VXFORM2 Form_Unload procedure.

If the user presses the ENTER key, or doubleclicks a record (when *EditMode* is FALSE), *RetVal* will contain the record number that was highlighted in the browse table immediately prior to the user action. All of the BROWSE_ constants are negative numbers. If *RetVal* is greater than zero, then you know what action the user took.

Returns

See the *RetVal* parameter above.

Usage

vxBrowse and vxCtlBrowse are intended to be the primary tools you will use to create vxBase applications. You can display only the data you want in the table by using the vxTable functions. You can define visual relationships between one file and another (and another and another) with the vxJoin command that are absolutely splendid in execution (try the Link items in the sample system and let your imagination flow). vxJoin links are only possible with vxBrowse - not vxCtlBrowse.

The entire set of sample programs revolves around the use of vxBrowse. Use them freely as templates for your own applications.

vxBrowse is also very handy in implementing help lists. For example, suppose a form control required the entry of a valid customer code. You can set up a help button beside the customer code control that activates a browse window on the customer file. When the user finds the record he wants, he simply doubleclicks it or presses the ENTER key to pass the record back to you. You can then extract the required field data and place it directly into the control without the need for typing the data.

The vxTable functions allow you customize your browse tables as to column heads and the sequence and format of the data you display. If no table is declared, vxBrowse provides a raw data display with the field names as column heads. Numeric fields are right justified in columns and dates are formatted as "mm/dd/yy" (default) or whatever format has been set with vxSetDate.

The vxMenu functions allow you to define custom menus on the browse table.

Function vxBrowsePos allows you to position and size the browse window. If this function is not called prior to beginning a browse, the size and position are dependent on the size and position of the underlying window.

Data from more than one database may be displayed in a horizontal row if a relationship is set up with vxSetRelations and column contents are defined with vxTableFieldExt.

Quick Key

Quick Key searches are a standard feature of a vxBrowse window. Usually, you will set up a browse with the vxTableDeclare function and place the index key field first in the column array. If an index is active during a browse. the user simply presses the sequence of characters he is looking for and the browse table reacts accordingly. The status of the Quick Key field is shown in the window caption.

For example, if the user had a browse table active consisting of customer codes and names, and the file was keyed on the code, then pressing the "T" key would position the table to the first record that had a customer code beginning with the letter "T". Subsequent key presses without intervening actions (such as pressing an arrow key or using the vertical scroll bar) will expand the quick key and narrow the search. If a quick key item is not found, the table will be positioned to the next higher record and the quick key adjusted accordingly (for example, if "TH" was entered and no code existed that began with these two letters, but a code existed that began with the letters "TI", then the table would be positioned there, and the quick key in the caption would show "TI" instead of the "TH" that the user entered).

One limitation on Quick key access becomes evident if you have a filter defined. If the partial key entered matches a filtered record, vxBrowse makes no attempt to find a record past that to satisfy the the logic in the paragraph above. Instead, a single beep is sounded and we stay where we are.

NOTE: All key presses directed at the quick key algorithm are converted to upper case before the seek is performed. You should ALWAYS use the UPPER() xBase function when indexing character fields.

The column head that contains the quick key may be marked with "*" by specifying its relative position in vxBrowseSetup.

Vertical Scrolling

Records that are displayed in a browse table with a controlling index react to a movement in the vertical scroll bar thumb in two ways. First, the relative position of the thumb in the scroll bar is ascertained to determine where, approximately, the display should start. For example, if the thumb was positioned halfway down the bar, the display should begin at the halfway point in the file. Because the file is indexed, we cannot simply go to the halfway record (i.e., if there were 5000 records in the file, we cannot go to 2500 and start there). Instead, we must find the 2500th index pointer so we read 2500 index keys to get the start record. Second, we use the record number attached to the key to get the first actual record and we're away. Obviously, if the file is very large, using the thumb to move around in the file will be on the slow side. The quickest way to traverse the records in a browse table is to use the Quick Key feature or the Page Keys (or click on the paging area in the vertical scroll bar).

The method for finding the record which relates to the position of the scroll thumb may be controlled with the *Threshold* parameter of vxBrowseSetup.

Other Menu Items

The browse table always has a menu bar unless turned off with vxBrowseSetup. If vxBase menus are not turned off with vxBrowseSetup, items that always remain on the menu bar are Query and Utilities.

In the query dialog box that is brought up when Query Search is selected, the user may enter any string. The search is case insensitive. It is also field insenstive. If the string is found anywhere in the record (even crossing field boundaries), that record is highlighted. The Query Find Next command simply finds the next occurrence of the same string. The standard Find Next accelerator key, F3, may be used instead of the menu item.

The utilities provide a lowercase toggle. When checked (the default value), the records in the table are displayed in all lowercase. This makes a cleaner and more readable display. If the user wishes to display the records exactly as entered, he toggles the lowercase switch off. The default case used in the browse window may be changed with vxBrowseCase.

The utilities Print option prints all records that vxBrowse would display. Defined tables are used to supply headings and the printout is exactly in the same format as the display. Use vxTableDeclare with vxBrowse to format quick reports. The position of the Print menu item may be changed with vxBrowseSetup.

The About File item tells the user a little bit about the file - its

name, size, etc. - and a listbox displaying the field structure.

User defined menus may also be created and displayed on the browse table with vxMenuDeclare and vxMenuItem.

vxBrowse Limitations

Up to 8 vxBrowse windows may be active at a time (total for all active tasks using vxBase). vxBrowse windows attached to a task must be closed in the reverse sequence of opening. vxBase maintains an internal stack of browse windows and informs the user about the closure sequence if he picks the wrong one to close.

There is a reason for this. vxBrowse is a function and as such it maintains a return address to the program line following the original call. In C or Assembler, it is a simple matter to extract this address and maintain an internal stack to always go back from whence you came, no matter what the sequence of function return.

Unfortunately, Visual Basic maintains a program area for a call to a DLL function in only one place in its structure. Therefore every call to vxBrowse from Visual Basic emanates from the same program location and returns to the instruction following the call. Visual Basic maintains its own internal stack of return addresses and pops the address of the LAST call to vxBrowse off of this stack and returns to the instruction following that call. It always returns to the instruction following the last call to vxBrowse.

The popping of the return address by Visual Basic follows a whole lot of other things which essentially restores the Visual Basic state to what it was before the call. What this means to us is that a function such as vxBrowse, which does not return to Visual Basic immediately after the call to it, and which may be called again in the Windows environment while other vxBrowses still have not completed, must be terminated in the reverse sequence of call in order for Visual Basic to return to the instruction following each vxBrowse.

On exit from a vxBase application, no vxBrowse table may be active. See the example shown in vxCloseAll for an exit protocol that ensures both windows and files are closed properly, and that allocated memory is released.

Use vxCtlBrowse if you require a table that is always visible and that does not have the limitations described above.

Multiuser Considerations

No records are locked by vxBrowse unless and until the user makes a record selection and vxSetLocks is TRUE (the default). If other users lock records that will be displayed by the browse, the browse will wait until the file is free unless vxSetLocks is set to FALSE. If the user selects a record for update or deletion that is already in use, he is informed immediately via a message box that the record is locked and he can retry the operation or abort and carry on with the browse.

Focus Issues

If a browse window disappears behind another because of a loss of focus, and it apparently cannot be made to reappear, either double-click on the background window screen (Windows system desktop), or, if that is not visible, press CTL-ESC to bring up the task management window. The

browse window that was invisible will appear on this list and it may be restored.

Example

```
  j% = vxSelectDbf(AirtypesDbf)    ' select database
  j% = vxSelectNtx(AirtypesNtx)

  TypeReturn = 0                   ' Browse return value
                                   ' declared as GLOBAL

' An active form must be visible because we need a
' parent for our browse
' ------------------------------------------------------
  If Not VXFORM1.Visible Then VXFORM1.Show

' Execute the browse routine (will use table declared
' in TypesOpen - in sample file VXBMOD.BAS)
' ------------------------------------------------------
  Call vxBrowse(VXFORM1.hWnd, AirtypesDbf, AirtypesNtx,
       TRUE, TRUE, TRUE, 0, "Aircraft Types", TypeReturn)
' (the above would be on one line)

' Browse returns a code or record number in TypeReturn var.
' If an edit menu item is selected, a code is returned.
' If the enter key is pressed, the rec number is returned.
' Double clicks when EditMode is true allow edit onscreen.
' (return codes defined in global vxbase.txt)
' ------------------------------------------------------
  Select Case TypeReturn

     Case BROWSE_ERROR
        MsgBox "Error in AirTypes Browse!"
        Exit Sub

     ' user closed browse with sys menu
     ' --------------------------------
     Case BROWSE_CLOSED
        j% = vxSelectDbf(AirtypesDbf)
        Call vxTableReset
        j% = vxClose()
        Exit Sub

     ' all other choices are processed by VXFORM2
     ' ------------------------------------------
     Case Else
        VXFORM1.Hide
        VXFORM2.Show
  End Select
```

See Also

vxBrowseCase, vxBrowsePos, vxBrowseSetup, vxCtlBrowse, vxJoin, vxMenuDeclare, vxMenuItem, vxSetDate, vxSetLocks, vxSetRelations, vxTableDeclare, vxTableField, vxTableFieldExt

vxBrowseCase

Declaration

```
Declare Sub vxBrowseCase Lib "vxbase.dll" (ByVal DefCase As Integer)
```

Purpose

Set the default case for ALL vxBrowse displays.

Parameters

DefCase is one of VX_UPPER or VX_LOWER as defined in vxbase.txt.

Returns

Nothing.

Usage

The default case used to display data in vxBrowse and vxCtlBrowse tables is VX_LOWER (i.e., lower case). The user can change the display to reflect the exact contents of the database (as entered) by unchecking the Utilities Lowercase menu item on the vxBrowse menu bar. The programmer may change the default to VX_UPPER, which displays the data exactly as entered, in both upper and lower case.

This is a SYSTEM WIDE function. All vxBrowse/vxCtlBrowse displays for all active tasks will be affected. It would normally be issued in your startup form FORM_LOAD procedure.

Example

```
Call vxBrowseCase(VX_UPPER)
```

See Also

vxBrowse, vxBrowsePos, vxBrowseSetup, vxCtlBrowse, vxCtlBrowseMsg

vxBrowsePos

Declaration

Declare Sub vxBrowsePos Lib "vxbase.dll" (ByVal StartX As Integer, ByVal StartY As Integer, ByVal xWidth As Integer, ByVal yHeight As Integer)

Purpose

Set the start position and size of an upcoming browse window that will be opened using the currently selected database file.

Parameters

All parameters to this function use familiar character units in the x dimension and line height units in the y dimension. The units are converted to the average character width and height of the standard Windows system font (or the font selected through vxBrowseSetup) and are therefore device independent.

StartX is the start position of the browse window in characters from the left edge of the screen.

StartY is the start position of the top of the browse window from the top of the screen.

xWidth is the start width of the browse window in characters.

yHeight is the height of the browse window (including caption and menu bar) in lines.

Returns

Nothing.

Usage

Browse window start position and size are defaulted according to the size of the underlying window (the *Hwnd* parameter passed to vxBrowse) if this command is not issued. If this command is issued, the position and size are relative to the entire screen.

If a file is browsed with vxBrowse and not closed, and the browse is called again, the second and subsequent window positions and sizes will be as they were when the window was closed (i.e., if the user changes size and/or position, this information is retained with the selected database).

Note: If you are changing the font and/or font size, call vxBrowseSetup BEFORE calling this function. vxBrowsePos uses the current font settings for the database to convert the passed coordinates to screen coordinates. If vxBrowsePos is called before vxBrowseSetup, coordinate calculations will use the Windows System font.

This function will size and position a vxBrowse window ONLY for the current database.

Example

```
' The proc below will set up an initial size and
' position for the browse window
' ------------------------------------------------
Call vxBrowsePos(10, 5, 50, 15)
' the coordinates are in familiar character and line
' units. The first param is x (characters in from left),
' the second param is y (lines down from top), the third
' param is the width of the window in characters, and the
' last param is the window height in lines

' if the user movers or sizes the window, and subsequent
' vxBrowse calls are made without an intervening close of the
' file, the window will retain its last position and size.
```

See Also

vxBrowse, vxBrowseCase, vxBrowseSetup

vxBrowseSetup

Declaration

Declare Sub vxBrowseSetup Lib "vxbase.dll" (ByVal Menus As Integer, ByVal PrintMenu As Integer, ByVal QCol As Integer, ByVal V3D As Integer, ByVal FontName As String, ByVal FontSize As Integer, ByVal Weight As Integer, ByVal Italic As Integer, ByVal Hdr As Integer, ByVal MinMax As Integer, ByVal Thresh As Integer)

Purpose

Controls the appearance and some of the functionality of a vxBrowse or vxCtlBrowse.

Parameters

Menus controls the standard menus added to a vxBrowse window. If FALSE, both the Utility menu and the Query menu are suppressed. This parameter has no effect on a vxCtlBrowse because a vxCtlBrowse has no menus.

PrintMenu controls the placement of the "Print" menu item on the vxBrowse menus. If the value is 0 (zero), no print menu item will be added; if 1, the print menu item appears on the standard edit menu (the default); if 2, the print menu item will appear on the Utilities menu. This parameter has no effect on a vxCtlBrowse.

QCol is the number of the column (relative to 1) that responds to Quick Key seeks. If this parameter is specified, an asterisk "*" is placed in front of the header text for that column to indicate that this column is seekable with quick key strokes.

V3D if TRUE will display the browse table in 3d format on a gray background. If FALSE, the browse is displayed as conventional black text on a white background. Each record and column is separated with a light gray line to give the appearnce of a grid. The default value is TRUE.

FontName is the name of an available font that will be used to display the browse table. It must be a valid name. A good place to look at an enumeration of your fonts is in the list box of fontnames on the VB properties bar for a text box.

FontSize is the size of the font in points. This number is device dependent to some extent (on your video resolution). Experiment before assuming that a given font size will yield the desired result.

Weight is a vxBase Global constant that specifies the weight of the font. The following constants are defined in vxbase.txt:

```
Global Const VX_DONTCARE = 0
Global Const VX_THIN = 100
Global Const VX_EXTRALIGHT = 200
Global Const VX_LIGHT = 300
Global Const VX_NORMAL = 400
Global Const VX_MEDIUM = 500
Global Const VX_SEMIBOLD = 600
Global Const VX_BOLD = 700
Global Const VX_EXTRABOLD = 800
Global Const VX_HEAVY = 900
```

Italic if TRUE will display the font in italic. If FALSE (the default), the display is not in italic.

If you do not wish to change the font (from the default Windows System font), pass the *FontName* as a space, the *FontSize* as 0, the *Weight* as 0, and *Italic* as FALSE.

Hdr defines the type style used in the column headers. The default is FALSE (which is shadowed text). If passed as TRUE, the header text is displayed in a flat style.

MinMax defines whether or not minimize and maximize buttons will appear on the browse window. If FALSE (the default) there are no buttons; if TRUE, the buttons appear. This parameter has no effect on vxCtlBrowse.

Thresh defines the number of records used as a threshold for implementing the vxBase relative scroll thumb positioning algorithm. The default value is 5000 (which is what you get if you specify 0).

Threshold Explanation: If the user positions the vertical scroll thumb, the browse display will begin at a point that is relative to the proportion of the new thumb position to the vertical scroll bar length. In other words, if the thumb is positioned to the middle of the vertical scroll bar, the record pointer is moved to the middle of the file.

This is easy if no indexes are being used (and in this case the threshold does not apply). We simply take the number of records and divide by 2 and that's where we start the display.

If the file is indexed, however, we must position the record pointer to the logical middle of the file. This means we have to count keys (just like vxNtxRecNo does) until we reach the middle key and then position the record pointer to the physical record pointed to by the ntx key entry.

This can take time if the file is large. An optimum size that yields a respectably short time is about 5000 records (which is the default threshold). If the file is larger than this, vxBase uses a key analysis algorithm to determine the approximate position of the file (high key value minus low key value times the scroll thumb proportion plus the low key value equals an approximate key we can softseek on). This algorithm works quite well on a database that is regularly sequenced (for example,

a name and address file with a fairly regular distribution of names throughout the alphabet will yield a key close to "M" to start the display at if the thumb is positioned in the middle of the scroll bar).

If your database contains more than 5000 records, and the distribution of keys is irregular (e.g., many duplicate keys or duplicate starting portions of keys or lots of A's and Z's with nothing in between), then you will likely wish to increase the threshold value to a number greater than the number of records in the file (maximum 32,767) to use the exact relative positioning algorithm. If there are more than 32,767 records in the file, the approximation alorithm will be used.

What's more important? Speed or an accurate thumb? This is a question for the ages.

Returns

Nothing.

Usage

Use this procedure to fine tune the appearance and functionality of your browse tables (both of the vxBrowse and the vxCtlBrowse variety).

Note: the database you are going to be browsing must be open and selected when the call to this procedure is issued.

If setting up for a vxBrowse window that will be positioned and sized with vxBrowsePos, call vxBrowseSetup BEFORE vxBrowsePos to ensure that the correct font is used to determine the window coordinates.

Example

```
' the browse must be set up either prior to
' or during the load of the form that contains
' the text box that will hold the browse
' ------------------------------------------
Sub Form_Load ()
  vxClientDbf = vxUseDbf("\ab2\abacus\sam\vxuser.dbf")
  vxCl1Ntx = vxUseNtx("\ab2\abacus\sam\vxuser.ntx")

  Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 6)
  Call vxTableField(1, "Serial", "vxser", VX_FIELD)
  Call vxTableField(2, "Name", "vxname", VX_FIELD)
  Call vxTableField(3, "Company", "vxcompany", VX_FIELD)
  Call vxTableField(4, "Phone", "vxphone", VX_FIELD)
  Call vxTableField(5, "City", "vxcity", VX_FIELD)
  Call vxTableField(6, "Country", "vxcountry", VX_FIELD)

  Call vxBrowseCase(VX_UPPER)
  Call vxBrowseSetup(0, 0, 1, 1, "Arial Narrow", 15, VX_SEMIBOLD,
                     FALSE, 0, 0, 0)
End Sub
```

See Also

vxBrowse, vxBrowseCase, vxBrowsePos, vxCtlBrowse, vxCtlBrowseMsg, vxMenuDeclare, vxMenuItem, vxSetLanguage, vxSetRelation, vxTableDeclare, vxTableField, vxTableFieldExt, vxTableReset

vxChar

Declaration

Declare Function vxChar Lib "vxbase.dll" (ByVal FieldName As String) As String

Purpose

Extract the first character from a defined field.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

A visual basic string that contains the first character of the field.

Usage

Commonly used to test the contents of a field whose data format is known.

Example

```
If UCase$(vxChar("PersonSex")) = "M" Then
   MaleProcess
Else
   FemaleProcess
End If
```

See Also

vxEmpty, vxField

vxClose

Declaration

```
Declare Function vxClose Lib "vxbase.dll" () As Integer
```

Purpose

Close the currently selected database.

Parameters

None.

Returns

TRUE if the close was successful, FALSE if not. A FALSE return could mean that one of the index files associated with the database had an error in closing.

Usage

A dbf file opened with vxDbfUse or one of its variants must always be closed. This ensures that any changes to the xBase header info become permanent as well as freeing any memory allocated to store the database structure, file structures, record buffer, table declarations and table joins. If an attempt is made to close a file that resides in an active browse window (for example, by another task that is using the file), the file is not closed but the result reported to the current task is TRUE and the file is no longer available to be selected from the task that initiated the close without another vxUseDbf being issued.

If the record buffer has been changed and not yet written, it is written to disk.

All open index files associated with the dbf are also closed. It is not necessary to explicitly close the index files.

After a file has been closed, it must be opened again with vxUseDbf or one of its variants before it may be accessed again.

Example

```
j% = vxSelectDbf(AirtypesDbf)
If Not vxClose() Then
   MsgBox "Error in Airtypes close"
End If
```

See Also

vxCloseAll, vxCloseNtx, vxDbfDate, vxJoinReset, vxTableReset, vxUseDbf, vxUseDbfAgain, vxUseDbfEX, vxUseDbfRO, vxUseNtx

vxCloseAll

Declaration

Declare Function vxCloseAll Lib "vxbase.dll" () As Integer

Purpose

Close all open database and index files.

Parameters

None.

Returns

TRUE if the operation is successful, otherwise FALSE. The operation will always return FALSE if there are any active browse windows open. The user is informed that the browse windows must be closed before an exit is allowed. In your exit strategy, follow the protocol shown in the example below (which comes directly from the sample application) to ensure that everything is cleaned up properly when an exit is requested.

Usage

Normally called when an application exit is taken to ensure that all record buffers, index nodes, and xBase headers are written and all associated memory is released.

Example

```
' ----------------------------------------
' This routine is activated from either the
' Exit menu item on VXFORM1 or by selecting
' the Close item from the system menu.
'
' We MUST test the vxCloseAll result in
' case there are any active browse windows
' that require closure before we can
' terminate the application
'
' If the close operation is successful, any
' open databases are closed (which updates
' the database header information) and all
' attached memory objects (Tables and Joins)
' are released.
' ----------------------------------------
Sub Form_Unload (Cancel As Integer)
  If Not vxCloseAll() Then
     Cancel = -1
     VXFORM1.Show       ' redraw top level form
     Exit Sub
  Else
     ' we MUST test the result of vxDeallocate
     ' to ensure that the task is not controlling
     ' memory for any other vxBase tasks that
     ' might be running at the same time as this one
     ' ----------------------------------------
     If Not vxDeallocate() Then
        Cancel = -1
        VXFORM1.Show
     Else
        vxCtlGrayReset
     End If
  End If
End Sub
```

See Also

vxClose, vxCloseNtx, vxDeallocate, vxInit

vxCloseNtx

Declaration

Declare Function vxCloseNtx Lib "vxbase.dll" (ByVal NtxArea As Integer) As Integer

Purpose

Close a previously opened index file.

Parameters

NtxArea is the select area of the index you wish to close. This number is returned by vxUseNtx when the file is opened or by vxAreaNtx after it has been opened.

Returns

TRUE if the operation is successful and FALSE if not.

Usage

A dbf file is normally opened with all of its index files if there is any chance that the file may change in the current procedure. This will ensure that all index files are updated if any key fields are altered or records are appended. A file opened for display only may be used with one index, and then another requirement may necessitate the closure of that index and the opening of one or more other index files (or none if freeing a file handle is your intention) as the case may be. If a dbf file is going to be left open, ensure that its index files are also open if it may be altered.

Example

```
MastFile% = vxUseDbf("Transfil.dbf")
MastIndex% = vxUseNtx("Transfil.ntx")
DisplayRecords
j% = vxCloseNtx(MastIndex%)
MastIndex2% = vxUseNtx("Transfi2.ntx")
```

See Also

vxClose, vxCloseAll, vxNtxDeselect, vxSetHandles

vxCollate

Declaration

Declare Sub vxCollate lib "vxbase.dll" (CharMap As Integer)

Purpose

Define a collating sequence table to be used for indexing other than the native collating table (ANSI or OEM depending on the setting of vxSetAnsi).

Parameters

CharMap is the first element in an array of 256 integers. Each integer represents the new collating sequence number for the character that would normally occupy that slot (array index - 1).

The collating sequence table is composed of 256 characters that range in value from zero to 255. Consequently, the index (minus 1) into the character map represents the current native character. By placing a different number into the integer at that spot we change its collating sequence to the new number.

For example, suppose you wanted a space character to be first (lowest) in your new collating sequence. A space is represented by decimal 32 (it is the 33rd character in the set which begins at zero) in both the ANSI and OEM character sets. To make a space the lowest value in your index collating sequence, you would place a zero in CharMap(33). The index number is the same as the decimal value of the character you wish to change plus one (for relative zero).

Note that the first integer in the array is passed BY REFERENCE rather than by value.

Returns

Nothing.

Usage

This function is used primarily for non-English language databases. The collating sequence of characters with diacritical marks that are used heavily in languages other than English is certainly incorrect in the OEM character set (for any language) and could be incorrect for certain languages if you are using ANSI databases as well (e.g., Swedish). To maintain index keys in a sequence that the user can understand, this function must be used to build a collating table.

The example below shows you how to build a true descending index in English (the DESCEND() xBase function simply complements the bits in the key using 2s complement arithmetic and creates a normal ascending index that results in descending order).

vxCollate can also be used for purposes like this but you must be careful to toggle the use of the table on and off with vxSetCollate. The table shown in the example would only be turned on for the file and index that it applied to. If there is more than 1 index for the file, using a table like this WILL be disastrous.

The new collating sequence table passed to vxCollate() MUST contain 256 elements. If fewer than 256, Windows will crash with a GPF.

Example

```
Dim CharMap(256)  As Integer

Sub Form_Load ()
   Call vxInit
   Call vxCtlGraySet
   Call vxCtlGraySet
   Call vxSetLanguage(VX_GERMAN)
   Call vxSetLocks(FALSE)
   Call vxSetString(0)
   j% = vxCloseAll()

   ' using OEM databases
   ' -------------------
   Call vxSetAnsi(FALSE)

   ' create descending collating sequence table
   ' ------------------------------------------
   i% = 255
   For j% = 1 To 256
      CharMap(j%) = i%
      i% = i% - 1
   Next j%
   Call vxCollate(CharMap(1))

   ' build descending index
   ' ----------------------
   vxDbf = vxUseDbf("\vb\vxuser.dbf")
   vxBackNtx = vxCreateNtx("\vb\vxback.ntx", "upper(vxname)")
   j% = vxClose()

   ' turn off table usage until required
   ' -----------------------------------
   Call vxSetCollate(FALSE)
End Sub
```

See Also

vxSetAnsi, vxSetCollate

vxCopy

Declaration

Declare Function vxCopy Lib "vxbase.dll" (ByVal NewDbfName As String) As Integer

Purpose

Make an exact copy or a filtered copy of the currently selected database.

Parameters

NewDbfName is the name of the new database file that receives the copy. The parameter may be a literal string or a string variable. It may include a complete path name. If an extension is not specified, vxBase defaults it to ".dbf". If a file exists with the same name it is overwritten. File names must begin with a letter.

Returns

TRUE if the operation is successful and FALSE if not.

Usage

A copy is made of the selected database that excludes deleted records. Memo files (that can include both textual memos and bitmaps) attached to the database are also copied to *NewDbfName.dbt*. Any file that matches *NewDbfName* is overwritten without warning.

This function is useful for sorting and packing data files without losing the originals, and for compressing memo files.

vxCopy respects filters defined with vxFilter as well. File subsets may be created by setting a filter and then using vxCopy to build the smaller file.

Multiuser Considerations

The currently selected database and its index files are locked for the duration of the operation. When it terminates, the record pointer is reset to its value before the function was called and that record is locked if vxSetLocks is TRUE.

Example

```
CustDbf% = vxSelectDbf("Custmast.dbf")
CustNtx% = vxSelectNtx("Custmast.ntx")
If vxCopy("Custcopy") Then
   MsgBox "Copy OK"
Else
   MsgBox "Copy Failed"
End If
```

See Also

vxAppendFrom, vxCopyStruc, vxCreateDbf, vxCreateNtx, vxFilter, vxPack, vxSetLocks

vxCopyStruc

Declaration

Declare Function vxCopyStruc Lib "vxbase.dll" (ByVal NewDbfName As String) As Integer

Purpose

Create an empty file whose structure is the same as the currently selected database.

Parameters

NewDbfName is the name of the new database file that is created. The parameter may be a literal string or a string variable. It may include a complete path name. If an extension is not specified, vxBase defaults it to ".dbf". An existing file with the same name is overwritten. File names must begin with a letter.

Returns

TRUE if the operation is successful and FALSE if not.

Usage

Commonly used to create a temporary batch file that will be used to capture data. The captured data would then be appended to a master file and the batch file erased. We can modify the sample code shown under vxAppendFrom to dynamically create a batch file instead of using a permanent file to hold temporary records.

Example

```
' create transaction batch file with the same
' structure as the master file
' -------------------------------------------
BatchName$ = "Tr" + SignOnId$
FileSpec$ = MyPath$ + BatchName$ + ".dbf"
IndexSpec$ = MyPath$ + BatchName$ + ".ntx"

' if file exists, error
' --------------------
If vxFile(FileSpec$) Then
   MsgBox "Error. Batch file exists!"
   Exit Sub
Else
   ' if no error, create empty transaction file
   ' ------------------------------------------
   TrMasterDbf% = vxUseDbf("Transmas.dbf")
   TrMasterNtx% = vxUseNtx("Transmas.ntx")
   j% = vxSelectDbf(TrMasterDbf%)
   If Not vxCopyStruc(BatchName$) Then
      MsgBox "Error in batch file creation"
      j% = vxClose()
      Exit Sub
   Else
      ' now create index same as master file
      ' ------------------------------------
      IndexExpr$ = vxNtxExpr(TrMasterNtx%)
      If Not vxCreateNtx(BatchName$, IndexExpr$) Then
         MsgBox "Error in index creation"
         Kill FileSpec$
         j% = vxClose()
         Exit Sub
      End If
```

```
   End If
End If
j% = vxClose()         ' close master file
TransDbf% = vxUseDbf(BatchName$)
TransNtx% = vxUseNtx(BatchName$)

' call transactions editing procedure
' -----------------------------------
CollectTrans

' if posting now, append transactions to
' master file after they have been posted
' and then clear the batch file in preparation
' for the next editing session
' -------------------------------------------
j% = MsgBox("Post Now?", 52)
If j% = 6 Then
   PostTrans
   TrMasterDbf% = vxUseDbf("Transmas.dbf")
   TrMasterNtx% = vxUseNtx("Transmas.ntx")
   j% = vxSelectDbf(TrMasterDbf%)
   j% = vxAppendFrom(BatchName$)
   j% = vxClose()      ' close master file
   Kill FileSpec$      ' erase batch file
   Kill IndexSpec$     ' and index
   Exit Sub
End If
j% = vxClose()             ' close the batch
```

See Also

vxAppendFrom, vxCopy, vxCreateDbf, vxCreateNtx

vxCreateDbf

Declaration

Declare Function vxCreateDbf Lib "vxbase.dll" (ByVal NewDbfName As String, ByVal NumFields As Integer, FStructure As FileStruc) As Integer

Purpose

Create a new database file.

Parameters

NewDbfName is the name of the new database file that is created. The parameter may be a literal string or a string variable. It may include a complete path name. If an extension is not specified, vxBase defaults it to ".dbf". An existing file with the same name is overwritten. File names must begin with a letter. Their length is limited by DOS to 8 characters.

NumFields is the number of fields the new database will contain.

FStructure is a user defined type that is filled in by the programmer with the data about the fields required to build the new database. The *FileStruc* type is defined in vxbase.txt (which should be included in your Global module). The type may be modified to suit your needs by adding or deleting "Fldnn" definitions to conform to the largest database (in number of fields) that your application will create.

The *FileStruc* type is composed of fixed length strings (each 16 characters in length) that represent the field definitions in your new file. Each string is named Fldnn where nn represents the field number. The structure supplied in vxbase.txt is defined with 32 fields. Add more if necessary.

The fixed length string that defines the field structure is composed of the following elements:

field name 10 characters
field type 1 character
field width 3 characters
field decimals 2 characters

The field type must be one of "C" for character, "N" for numeric, "L" for logical, "D" for date, or "M" for memo (or bitmap). A logical field length cannot exceed 1 character, a date field must be 8 characters wide, and a memo field length is 10 characters. If your new file definition contains a memo field, a file with the same name as NewDbfName will be created with a ".dbt" extension.

A numeric field cannot exceed 19 characters in width, which includes the decimal point and sign position if the number can be negative. If a numeric field has a number of defined decimals, the minimum length of the field is the number of decimal positions plus 2 (1 for the decimal point and 1 for a leading zero). If there is a possibility that the number may be negative, add another for the sign.

Field names must begin with a letter. The other nine positions can be letters, numbers, or the underscore character (not a hyphen) and may not contain embedded spaces. Trailing spaces of course are allowed (the field name can be from 1 to 10 characters in length).

The field structure for a new database is passed to vxBase as a user defined type because the elements in the structure must be contiguous in memory. Visual Basic string array elements are not necessarily contiguous in memory so we can't use an array. The fixed length requirement for the elements of the structure simplifies and speeds up the parsing vxBase performs to create your new database.

Returns

TRUE if the operation is successful and FALSE if not.

Usage

Your application could be shipped without any supporting database or index files. The first time it is run, you could create your files in a directory specified by the user.

Always test the result of vxCreateDbf to ensure the database was properly created before you attempt to use the new file.

Example

```
  Dim CustFile As FileStruc
  Dim NumFields As Integer

'                    1234567890123456  (alignment ruler)
  CustFile.Fld01 = "NAME      C 30 0"
  CustFile.Fld02 = "ADDRESS   C 30 0"
  CustFile.Fld03 = "CITY      C 20 0"
  CustFile.Fld04 = "PHONE     C 13 0"
  CustFile.Fld05 = "AMTOWING  N 15 2"

  NumFields = 5

  If Not vxCreateDbf("custfile", NumFields, CustFile) Then
     MsgBox "Error in database creation"
  End If
```

See Also

vxAppendFrom, vxCopy, vxCopyStruc, vxCreateNtx, vxCreateSubNtx

vxCreateNtx

Declaration

Declare Function vxCreateNtx Lib "vxbase.dll" (ByVal NewNtxName As String, ByVal NtxExpr As String) As Integer

Purpose

Create a new index file.

Parameters

NewNtxName is the name of the new index file that is created. The parameter may be a literal string or a string variable. It may include a complete path name. If an extension is not specified, vxBase defaults it to ".ntx". An existing file with the same name is overwritten. File names must begin with a letter. Their length is limited by DOS to 8 characters.

NtxExpr is a valid xBase expression (which may be as simple as a field name) that is passed as either a literal string or as a string variable. The expression must evaluate to a string. The expression must also, of course, reference field names in the currently selected database.

Returns

The new index is created, selected, and attached to the current database. The *NxtArea* is returned as an integer greater than zero if the operation was successful. If the operation was not successful, FALSE is returned. Always test the return value.

Note that you cannot test the return value with a NOT expression because a number greater than zero is NOT TRUE according to Visual Basic. Use the test format shown in the example below.

Usage

The index expression must evaluate as a string. If elements of your index are numeric or date fields, use the xBase STR() and DTOS() expressions to convert the fields to strings within the expression. Always use the UPPER() xbase function when indexing character fields. This allows instant Quick Key access in browse windows and correct alphabetical order being maintained in the index.

"custcode + datefield + numfield" is an invalid index expression if datefield and numfield are date and numeric fields respectively. If we assume the numeric field has a format of length 11 with 2 decimals, to create a valid index out of the same elements, we would use "custcode + dtos(datefield) + str(numfield,11,2)".

You can use this function to create new indexes for new databases created with the vxCreateDbf function (or however) or to create temporary indexes that you require for a one-shot report that is rarely run. Remember to explicitly close one-shot indexes and kill them after you are done with them.

A descending index may be built using the xBase DESCEND() function

within your index expression (see vxDescend).

If you are soliciting an expression from the user, always use the vxEval function on the user supplied expression to test its validity before creating the index.

If vxSetMeters is TRUE, a meter bar charting the progress of the index creation is presented to the user. The programmer may alternatively specify his own gauge control with vxSetGauge.

Example

```
Sub TestCopy_Click ()
  Dim NtxExpr As String
  Dim Ret As Long

  AirtypesDbf = vxUseDbf("\vb\vxbtest\airtypes.dbf")
  AirTypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")

  ' get index expression from master file
  ' -------------------------------------
  NtxExpr = vxNtxExpr(AirTypesNtx)

  If Not vxCopyStruc("\vb\vxbtest\testcopy.dbf") Then
     MsgBox "Error in database copy struc"
     Exit Sub
  End If
  j% = vxSelectDbf(AirtypesDbf)
  j% = vxClose()

  TDbf% = vxUseDbf("\vb\vxbtest\testcopy.dbf")

  ' index create opens and selects new index and
  ' returns the index select area. Zero (FALSE)
  ' is returned if there was an error
  ' -----------------------------------------
  TNtx% = vxCreateNtx("\vb\vxbtest\testcopy.ntx", NtxExpr)
  If TNtx% = FALSE Then
     MsgBox "Error in index create"
     j% = vxClose()
     Exit Sub
  End If

  If Not vxAppendFrom("\vb\vxbtest\airtypes.dbf") Then
     MsgBox "Error in append from"
     j% = vxClose()
     Exit Sub
  End If

  Call vxBrowse(VXFORM1.hWnd, TDbf%, TNtx%, 0, 0, 0, 0,
                "Test", Ret)
  j% = vxClose()
End Sub
```

See Also

vxCopy, vxCopyStruc, vxCreateDbf, vxCreateSubNtx, vxDescend, vxEval, vxNtxExpr, vxSetGauge, vxSetMeters

vxCreateSubNtx

Declaration

Declare Function vxCreateSubNtx Lib "vxbase.dll" (ByVal NewNtxName As String, ByVal NtxExpr As String, ByVal ForCond As String) As Integer

Purpose

Create a permanent subindex that represents a defined subset of records in the main dbf file. Only records that pass the test of the defined conditional logical expression are included in the index.

Parameters

NewNtxName is the name of the new index file that is created. The parameter may be a literal string or a string variable. It may include a complete path name. If an extension is not specified, vxBase defaults it to ".ntx". An existing file with the same name is overwritten. File names must begin with a letter. The file name length is limited by DOS to 8 characters.

NtxExpr is a valid xBase expression (which may be as simple as a field name) that is passed as either a literal string or as a string variable. Maximum length of the expression is 255 characters.

ForCond is a valid xBase expression that evaluates as a logical TRUE or FALSE. The index built by vxCreateSubNtx is composed of keys built from records that satisfy the *ForCond* expression. The *ForCond* expression may be passed as either a literal string or as a string variable. Maximum length of the expression is 255 characters.

Usage

A subindex is an index that represents a defined subset of records in the main dbf file. The indexing expression is in no way related to the conditional expression that determines whether or not the record will be represented in the index. In other words, the condition that determines the presence or absence of a key does not depend upon the value of the key. For example, a subindex may be created using the expression "upper(custname)" as the key. The conditional expression could be "(left(vxcountry,6)="CANADA") .or. (left(vxcountry,6)="U.S.A.". This would produce an index that represented customers in North America only. An open subindex in an index list attached to a database is maintained just as like other index. When a record is added, an index key for the record is only added if the conditional expression evaluates as TRUE. If a record is updated, and the update data invalidates the record for inclusion in the subindex, the key is deleted.

If you regularly filter data based upon a condition such as the one above, a permanent subindex makes data retrieval MUCH faster. If the file is large, and you need to set a temporary filter that may result in very long record retrieval times, it is probably faster to create a temporary subindex instead. A subindex makes it APPEAR that the database contains only records that satisfy the conditional logical expression.

If vxSetMeters is TRUE, a meter bar charting the progress of the index creation is presented to the user. The programmer may

alternatively specify his own gauge control with vxSetGauge.

Warning

If you are editing a file that is being controlled by a subindex (i.e., the index currently selected), field changes or additions that result in the for condition returning FALSE will leave the record pointer in an undefined state after the record is saved. If the record is an update, the key will be removed from the index. If the record is an addition, the key will not be added to the index. IT IS YOUR RESPONSIBILTY to position the record pointer to a valid record if this should occur.

There are a number of ways this can be accomplished. If you know the condition, you can test if the new data will qualify the record for inclusion in the index. Or you may simply seek for the record again after writing. If it does not exist, you can position the record pointer to someplace you have prepared to go to before you began the update routine. This is the strategy used below:

Example (Updating a subindexed file safely)

```
' Validate data when save button is pressed
' ----------------------------------------
Sub CustSave_Click ()
  ' verify something in the field
  ' ----------------------------
  j% = vxSelectDbf(vxClientDbf)
  SeekKey$ = CustCode.Text
  If EmptyString(SeekKey$) Then
     MsgBox "vxSer Field cannot be empty"
     Exit Sub
  End If

  ' reread the record
  ' -----------------
  j% = vxSeek(SeekKey$)
  ThisRec& = vxRecNo()

  ' now get previous record in case
  ' in a subindex situation the changes
  ' the user makes removes this record
  ' from the index.
  ' The subindex condition here is
  ' vxCountry = 'CANADA' .or. vxCountry = 'U.S.A.'
  ' If the user has changed the country to
  ' something else, this record will disappear
  ' from the index when it is written so we
  ' must plan on what to do of this happens.
  ' -----------------------------------------------
  j% = vxSkip(-1)
  If vxBof() Then
     j% = vxTop()
  End If
  PrevRec& = vxRecNo()

  ' put record pointer back to update
  ' ---------------------------------
  j% = vxGo(ThisRec&)

  ' Data passed. Put it away
  ' ------------------------
  j% = vxLockRecord()
```

```
  Call vxReplString("vxcompany", (CustCompany.Text))
  Call vxReplString("vxname", (CustName.Text))
  Call vxReplString("vxaddress1", (CustAddress.Text))
  Call vxReplString("vxaddress2", (CustAddress2.Text))
  Call vxReplString("vxcity", (CustCity.Text))
  Call vxReplString("vxstate", (CustState.Text))
  Call vxReplString("vxcountry", (CustCountry.Text))
  Call vxReplString("vxzip", (CustZip.Text))
  Call vxReplString("vxphone", (CustPhBus.Text))
  Call vxReplString("vxfax", (CustFax.Text))
  j% = vxWrite()
  j% = vxWriteHdr()
  j% = vxUnlock()

  ' Update status box
  ' -----------------
  VXFORM1.StatBar.Text = "Record " + LTrim$(Str$(ThisRec&)) + " saved"

  ' Now see if the record still exists in this index.
  ' If it does not exist, the country has been changed
  ' so we will go to the Previous record we saved
  ' above and load the form data with that. Otherwise,
  ' this record data will remain on the form.
  ' ------------------------------------------------
  If Not vxSeek(SeekKey$) Then
     j% = vxGo(PrevRec&)
     CustDataLoad
  End If

  CustReturn = BROWSE_EDIT
  RecChange = False
End Sub
```

Example (Creating a subindex)

```
 ' we open a subindex just as we do a normal index
 ' UserFname$ contains the path and name of the subindex
 If Not vxFile(UserFname$) Then
    vxCl1Ntx = vxCreateSubNtx(UserFname$, "vxser",
                              "left(vxcountry,6)='CANADA'
                              .or. left(vxcountry,6)='U.S.A.'")
 Else
    vxCl1Ntx = vxUseNtx(UserFname$)
 End If
```

See Also

vxCreateNtx, vxIsSubNtx, vxNtxSubExpr, vxNumRecsSub, vxUseNtx, vxSetGauge, vxSetMeters

vxCtlBrowse

Declaration

Declare Function vxCtlBrowse Lib "vxbase.dll" (ByVal ControlHwnd As Integer, ByVal DbfArea As Integer, ByVal NtxArea As Integer, ByVal EditMode As Integer, ByVal StartRec As Long, ByVal MemoHwnd As Integer, ByVal MemoField As String) As Integer

Purpose

Place a browse table into a multiline text box control. The browse table is bounded by the confines of the text box. The browse reacts to standard events (mouse pointing and clicking, quick key presses, etc.) in the same fashion as vxBrowse. Communication with the Browse table is accomplished though the use of button controls (or menu items) on the main form and the use of vxCtlBrowseMsg.

The main differences between vxBrowse and vxCtlBrowse are:

> vxBrowse is a popup window unto itself (even a task), and is controlled with its own message loop. It may be resized, moved, minimized, etc. After it is started, communication between the calling program and vxBrowse is a one way street; vxBrowse can tell the calling program what the user did but the calling program cannot interrogate anything that happens during the browse. After the browse window is closed, vxBrowse reports the user action to the calling program.
>
> vxCtlBrowse is a child window that resides entirely within the confines of a bounded text box on a main form. Communication works both ways; the browse window can report certain events through standard procedures (e.g., key presses and key downs), it can report its state (what record number is highlighted?), and it can be controlled by the programmer and by the user (e.g., redraw thyself starting someplace else). It does not have to go away in order for the programmer to react.

vxBrowse is perfect for help pick lists and as a primary tool to view a set of records in tabular format. As a pick list help tool, the window pops up over top of another window, the user picks something (or not), and the window goes away - leaving the programmer with the user's choice (or not). vxBrowse can also be used to initiate and display dynamic one to many relationships. vxCtlBrowse cannot do this dynamically - only under programmer control by creating another vxCtlBrowse window and manually implementing the browse.

vxCtlBrowse is much more flexible in that it can stay around and react to and be affected by user (and programmer) actions. Its not as good as a help pick list and its no good at all as a quick and dirty report generator.

vxCtlBrowse can also dynamically display memos in a second text box! (highlight a record that has a memo and the memo appears in the defined text box).

Parameters

ControlHwnd is the window handle of the multiline text box that the browse is going to inhabit. The window handle is not directly available from Visual Basic. It must be extracted with vxCtlHwnd (see the example in vxCtlBrowseMsg below).

DbfArea is the select area of an open database. If it is not currently selected, vxCtlBrowse will make it the current selection.

NtxArea is the select area of an index file attached to DbfArea%. Pass a 0 (zero) if no idex is to be used.

EditMode is passed as TRUE or FALSE. If TRUE, a mouse doubleclick on a browse row/column will pop up an edit window with the text of the selected field ready for edit. **If the user presses the ENTER key, the field is replaced and the record is written. If the user presses the ESC key, the update is cancelled.** Note that the only data validation possible with the onscreen edit is for type (e.g., numeric fields must contain numbers). If your data requires more sophisticated validation, never pass a TRUE in this parameter.

If *EditMode* is FALSE, a mouse doubleclick will be converted to an ENTER key value and may be interrogated in the textbox_keypress event procedure. Before the key is passed, the record pointer is positioned to the currently highlighted record so it is automatically available to vxRecNo() if desired.

See the writeup in vxBrowse under *EditMode* for information on the relationship between this parameter and vxTableDeclare/vxTableField.

StartRec is a long integer that contains the browse table starting record number. If passed as 0 (zero), the display will commence at the top of the file. If a record subset has been defined with vxTableDeclare, it is the programmer's responsibility to ensure that the pointer is positioned to the correct starting record prior to calling vxCtlBrowse.

MemoHwnd is the window handle of a multiline text box (for textual memos) or a picture box (for bitmaps) that may contain the contents of a dynamic memo link. The window handle is not directly available from Visual Basic. It must be extracted with vxCtlHwnd (see the example in vxCtlBrowseMsg below). If there is no memo link, pass this parameter as a 0 (zero).

MemoField is the name of the memo field that will be dynamically linked to the browse table. Whenever a record in the browse table receives the highlight, and that record contains a memo reference in this field, then the memo (or bitmap) will be displayed in the MemoHwnd% text or picture box. Pass this parameter as a space (" ") if there is no memo link. If a relationship has been set up and is being displayed by the browse, the memo field must belong to the parent file. NO ALIAS NAMES ALLOWED.

Bitmap displays automatically size the picture box to the size of the bitmap with an anchor at the upper left corner of the picture box. If the bitmap results in a size that overflows the form, it is clipped

on the right and/or on the bottom.

Returns

TRUE if the browse was successfully set up. FALSE is returned for one of the following reasons:

(1) no current database.
(2) DbfArea is invalid.
(3) no more browse windows available (maximum of 16 active at once in all concurrent vxBase tasks - NOT including any vxBrowse windows).
(4) ControlHwnd% is invalid.
(5) The browse has already been set up in ControlHwnd (you don't have to worry about calling the same browse twice into the same window - it simply returns FALSE).
(6) The file selected with DbfArea is empty.
(7) Invalid memo field name.
(8) Memo field name is not a memo.
(9) MemoHwnd is invalid.
(10) Out of memory.

Usage

A wonderful tool for displaying and activating file editing procedures. Users expect data to be presented in tabular format. That's why phone books are so successful. Use vxCtlBrowse to provide a gross view of the data, and then use vxCtlBrowseMsg to react to the user's requests.

Please see the sections entitled "Quick Key", "Vertical Scrolling", and "Multiuser Considerations" under vxBrowse.

NOTE: The text box that is created to hold the browse must be given the multiline property. If scroll bars are required to allow all of the browse data to be viewed (horizontal, vertical, or both), vxCtlBrowse automatically provides them.

In the Form_load procedure of the form that calls vxCtlBrowse, you should change the MousePointer property of the text box that is going to hold the browse to an arrow as follows:

```
BrowseBox.MousePointer = 1
```

Alternatively, you can set the MousePointer property for the text box at Design Time.

Changing the cursor shape will stop an annoying flicker that results from vxCtlBrowse constantly changing the mouse pointer from an I-Beam to an Arrow whenever the mouse is moved.

vxCtlBrowse may not be called from a Form_Load procedure. The text box that is to hold the browse has not been created yet so no window handle may be passed to vxCtlBrowse. The best place to call it is in the Form_Paint procedure. Form_Paint of course may be called many times during the life of the form but vxCtlBrowseMsg will not invoke itself any more than once for a defined text box control. See the example in vxCtlBrowseMsg below.

Browse Navigation

The browse may be perused vertically with the mouse and the scroll bar, the Page Up and Page Down keys, the Home and End Keys, and the up and down arrow keys.

The horizontal aspect may be controlled with the mouse and the scroll bar, the right and left arrow keys, and Ctrl-Left and Ctrl-Right to move horizontally a page at a time.

Example

SEE vxCtlBrowseMsg BELOW.

See Also

vxBrowse, vxBrowseCase, vxBrowseSetup, vxCtlBrowseMsg, vxSetRelation, vxTableDeclare, vxTableField, vxTableFieldExt, vxTableReset

vxCtlBrowseMsg

Declaration

Declare Function vxCtlBrowseMsg Lib "vxbase.dll" (ByVal Hwnd As Integer, ByVal Msg As Integer, Param As Any) As Long

Purpose

Communicate with a vxCtlBrowse text box. Messages and directives to the browse are passed via this function, usually via a button or menu item click event. The browse can return requested information or react to a directive issued by vxCtlBrowseMsg.

The vxCtlBrowse may also send messages back to the KeyPress and KeyDown event procedures for the text box.

Parameters

Hwnd is the window handle of the multiline text box that the browse resides in. The window handle is not directly available from Visual Basic. It must be extracted with vxCtlHwnd (see the example below).

Msg is one of the following Global Constants as defined in vxbase.txt:

```
Global Const VXB_REFRESH = 0
Global Const VXB_FILTERDLG = 1
Global Const VXB_FILTERPRG = 2
Global Const VXB_GETCURRENTREC = 3
Global Const VXB_GETTOPREC = 4
Global Const VXB_STATS = 5
Global Const VXB_CASE = 6
Global Const VXB_SEARCHDLG = 7
Global Const VXB_SEARCHPRG = 8
Global Const VXB_SEARCHAGAIN = 9
Global Const VXB_SEEK = 10
Global Const VXB_CLOSE = 11
Global Const VXB_QUICKDISPLAY = 12
Global Const VXB_GO = 13
Global Const VXB_SKIP = 14
```

Each message is discussed under *Param* below.

NOTE: If any of the constants above are not defined and explicit chcking is not turned on, Visual Basic will send a 0 (VXB_REFRESH) to the browse. Ensure that all message values are defined.

Param is a parameter that accompanies a message to the vxCtlBrowse. Each message that requires a *Param* must have that *Param* passed ByVal. If a message does not require a *Param*, it may be passed as 0 (zero).

```
* ------------------------*
  DON'T FORGET THE BYVAL.
* ------------------------*
```

VXB_REFRESH: redraws the browse window. *Param* must be passed as a

long integer that contains a display start record number. This message would be sent after editing, adding, or deleting a record that affects the visible browse display. If you wish to start the display at the same position, use VXB_GETTOPREC to set the start record number passed with this message. If the record number is 0 (zero) or greater than the number of records in the file (vxNumRecs()), no refresh takes place. The return value may be ignored.

VXB_FILTERDLG: invokes the same vxBase filter dialog box that is used by the vxBrowse Filter menu command. The user may enter his own xBase filter expression. This message should be reserved for expert users only. Use the next message (VXB_FILTERPRG) after extracting filter parameters from the user and building the xBase expression under program control. *Param* is passed as 0 (zero). The return value may be ignored.

VXB_FILTERPRG: Sets a filter on the browse table. *Param* is passed BYVAL as a complete xbase expression that evaluates as a logical TRUE or FALSE. Note that deleted records are always filtered out of a vxCtlBrowse display. This is the preferred method of filter setting because things like field names and xBase function syntax can be controlled by the programmer (what user expects an address field to be named "A1"?). The return value may be ignored.

A filter set on the file with vxFilter() prior to the browse will be overridden by a filter set by either VXB_FILTERDLG or VXB_FILTERPRG (even though it is in effect when the browse commences). The filter will take effect again when the browse is closed.

A filter that has been set with VXB_FILTERPRG may be cancelled by passing *Param* as ByVal 0& as follows:

```
j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_FILTERPRG, ByVal 0&)
```

VXB_GETCURRENTREC: Retrieves the physical record number of the record that is currently highlighted in the browse display. *Param* is passed as 0 (zero). A long integer containing the record number is returned. This message would normally be used in response to a button press that invoked a record edit procedure. The user clicks the "Edit" button; the programmer reacts in the EditButton_Click event procedure by first going to the record using this message, extracting the record contents and presenting the data in text boxes on the same or another form for editing.

VXB_GETTOPREC: Retrieves the physical record number of the record that sits at the top of the browse display. *Param* is passed as 0 (zero). A long integer containing the record number of the top record is returned. After editing, adding, or deleting records, you probably want the display to restart at the same place that the user left off (to provide some continuity to

the session). See VXB_REFRESH above.

VXB_STATS: Presents a file statistics dialog box - its name, size, number of records, and a list box containing the field structure of the file. *Param* is passed as 0 (zero). The return value may be ignored.

VXB_CASE: The case of the display is toggled. If it started out as VX_UPPER (see vxBrowseCase), it becomes all lower, and vice versa. Note that VX_UPPER means as it was entered - not necessarily all upper case. *Param* is passed as 0 (zero). The return value may be ignored.

VXB_SEARCHDLG: Invokes a search dialog box that prompts the user for a string. If the string exists in the table, the record that contains the string is highlighted. If the browse was set up with vxTableDeclare, only columns defined with vxTableField are searched. Field boundaries are respected. If the browse is a raw data display, the entire record is searched for the string. If a match is found that crosses field boundaries or not, the record is highlighted. *Param* is passed as 0 (zero). The return value may be ignored.

VXB_SEARCHPRG: Searches for a string passed ByVal in *Param*. This search is under programmer control. The same search algorithm as above is used. The return value may be ignored.

VXB_SEARCHAGAIN: Search for the same string (as passed via VXB_SEARCHDLG or VXB_SEARCHPRG) again - skipping forward one record first. *Param* is passed as 0 (zero). The return value may be ignored.

VXB_SEEK: Perform a softseek on the index (see vxSeekSoft). *Param* is passed ByVal as a string. A record is highlighted if there is a partial match or exact match. If there is no match, but a record exists with a key higher than the search key, it is highlighted instead. The return value may be ignored.

VXB_CLOSE: This is a very important message that must be issued to the vxCtlBrowse window when the form containing the text box is unloaded. It reclaims memory, clears the data structure that was set up vxCtlBrowse, and clears the edit box that the browse lived in. You may issue this message any time (not just when unloading the form) to start a new browse on a different file, or with a different index, or whatever. Just remember that there shouldn't be any active vxCtlBrowses left over when you unload the form (or end the program). *Param* is passed as 0 (zero). The return value may be ignored.

VXB_QUICKDISPLAY: Extracts the current contents of the user entered quick key value. A quick key entry will normally result in a the record pointer being moved so the KeyCode middle button down event generated by vxCtlBrowse whenever a new record is highlighted may be used to conveniently ask for the current

quick value. *Param* is passed as a long integer that contains the handle of the text box being used to display the quick value. The new message may be used to display the current quick value in a text box as follows:

```
Sub BrowseBox_KeyDown (KeyCode As Integer, Shift As Integer)
  '   Debug.Print KeyCode
  If KeyCode = 4 Then
     QWinLong& = vxCtlHwnd(QuickBox)
     ' Note: vxCtlHwnd normally returns an integer but you
     ' must explicitly cast its value as a long integer
     ' in order to fulfill the vxCtlBrowseMsg parameter
     ' requirements
     k& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_QUICKDISPLAY,
                         ByVal QWinLong&)
  End If
End Sub
```

Note: A VB label MAY NOT BE USED to display the quick value because a label is a "graphical object" and therefore does not have a window handle.

VXB_GO: Positions the highlight to a specific record contained in the current browse display. This is especially useful if you are updating a record on a detail form and, after you are done, you wish to refresh the browse table with the new information and then position the highlight to the record that was just updated (VXB_REFRESH always positions the highlight to the top record in the browse). Param is passed BYVAL as a dbf record number to the browse. You must ensure that the record number passed is indeed pointing to a record on the current browse display. The return value may be ignored.

Example:

```
Sub ButtonSave_Click ()
   SaveRecNo& = CurrentRec
   TopRec& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_GETTOPREC, 0)
   j% = vxSelectDbf(vxClientDbf)
   j% = vxGo(CurrentRec)
   j% = vxLockRecord()
   Call vxReplString("vxnamekey", (BoxNAMEKEY.Text))
   Call vxReplString("vxcompany", (BoxCOMPANY.Text))
   j% = vxWrite()
   j% = vxWriteHdr()
   j% = vxUnlock()
   k& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_REFRESH, ByVal TopRec&)
   k& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_GO, ByVal SaveRecNo&)
End Sub
```

VXB_SKIP: Moves the highlight up or down in the browse table (if param is passed as a negative number the movement is UP; 0 or a positive number moves DOWN). If the highlight is positioned at the top or bottom record in the display, the browse will scroll. This is useful where you have a browse table on one side of the screen and detail info about the highlighted record on the other side of the screen or even on a different form. If the detail side has a Previous or Next button on it, you can synchronize the highlight on the browse table to the detail record data. The return value may be ignored.

Example:

```
Sub ButtonNext_Click ()
   If RecChange Then
      j% = MsgBox("Record changed. Save?", 52)
      If j% = 6 Then
         ButtonSave_Click
      End If
   End If

   CurrentRec = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_GETCURRENTREC, 0)
   j% = vxGo(CurrentRec)

   ' skip forward one record
   ' -----------------------
   j% = vxSelectDbf(vxClientDbf)
   j% = vxSkip(1)

   ' test for end of file
   ' --------------------
   If vxEof() Then
      Beep
      VXFORM1.StatBar.Text = "End of File!"
      j% = vxGo(CurrentRec)
   Else
      VXFORM1.StatBar.Text = "Skipped to record " + LTrim$(Str$(vxRecNo()))
      CurrentRec = vxRecNo()
      CurrentRec = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_SKIP, ByVal 1&)
      ButtonEdit_Click
   End If
End Sub
```

Returns

The only two messages that result in a return of any value are VXB_GETCURRENTREC and VXB_GETTOPREC. These return record numbers as long integers. Returns from all other messages may be ignored.

Usage

vxCtlBrowseMsg is the only way you have of communicating with a vxCtlBrowse. vxCtlBrowse also communicates with you through the KeyPress and KeyDown event procedures attached to the text box.

If the user presses the ENTER key, or doubleclicks on a record when the EditMode parameter of vxCtlBrowse is FALSE, an value of 13 is passed through the KeyAScii parameter of the TextBox_KeyPress event. This is usually a signal that the user wishes to do something with the record that currently has the highlight (expand it, edit it, etc. - its up to you). YOU MUST SET KEYASCII TO 0 (ZERO) BEFORE THE KEYPRESS EVENT PROCEDURE EXIT AFTER RECEIVING AN ENTER KEY SO IT DOESN'T GET THROUGH TO THE TEXT BOX.

The ESCAPE key is also passed to the KeyPress event procedure as KeyAscii value 27.

The TextBox_KeyDown event procedure receives INSERT presses (as KeyCode 45) and DELETE presses (as KeyCode 46) as well. You may react or not react to these events as you wish.

The KeyDown event procedure also receives a KEY_MBUTTON (KeyCode as 4) whenever a record is highlighted in the Browse box. You can dynamically link a detail form display to this event.

Focus Issues

The browse display receives the focus automatically when it is created. The focus is also automatically shifted back to the browse after vxCtlBrowseMsg completes its task. Whenever the focus leaves the browse window, the column header row is inverted (red becomes cyan, etc.) The user can reset the focus if it is gone by tabbing to the text box, clicking on it, etc. - all the normal ways. You can also reset the focus with the SetFocus Method under program control.

Memos and Bitmaps

Dynamic memo field links result in the display of a defined text memo or a bitmap whenever one exists that is attached to a highlighted record. If a text memo, you may allow the user to edit the memo or not - save it or not (with vxReplMemo). If displaying a text memo, the control must be fully enabled to allow the user to scroll in the memo text box if the box is not large enough to hold the entire contents of the memo.

Bitmaps are displayed through this facility by passing the handle of a picture box rather than a multiline text box. The picture box is dynamically resized (whether the AutoResize property is set on or not) to provide an exact fit for the bitmap. Do not use vxCtlStyle to provide 3d effects to the picture box because the dynamic resizing will leave shadow lines all over your form.

If you wish to have the bitmap displayed in a static picture box (AutoResize property FALSE), you may trap the middle button event in the Key_Down procedure of the browse box and extract the bitmap yourself with vxPictureRead. The Aircraft sample application has some commented code that shows you how to do this in VYFORM2.

Example

The following example is the actual code used to alpha test the vxCtlBrowse function. The VB form had the following elements:

```
  BrowseBox: multiline text box with vertical and horizontal
      scroll bars attached.

  MemoBox: multiline text box with a vertical scroll bar attached.

  Buttons:
     ButtonAgain     to test VXB_SEARCHAGAIN
     ButtonCancFilt  to test VXB_FILTERPRG, ByVal 0&
     ButtonCase      to test VXB_CASE
     ButtonCurRec    to test VXB_GETCURRENTREC
     ButtonExit      to unload form and test VXB_CLOSE
     ButtonFilter    to test VXB_FILTERDLG
     ButtonPrgFilt   to test VXB_FILTERPRG
     ButtonRefresh   to test VXB_REFRESH
     ButtonSearch    to test VXB_SEARCHDLG
     ButtonStrSearch to test VXB_SEARCHPRG
     ButtonTopRec    to Test VXB_GETTOPREC
```

Note the use of vxCtlHwnd to convert a VB control handle into a Window Handle (in vxCtlBrowse and vxCtlBrowseMsg calls).

```
' ------------------------------------------
' the browse must be set up either prior to
' or during the load of the form that contains
' the text box that will hold the browse
' ------------------------------------------
Sub Form_Load ()
  vxClientDbf = vxUseDbf("\ab2\abacus\sam\vxuser.dbf")
  vxCl1Ntx = vxUseNtx("\ab2\abacus\sam\vxuser.ntx")

  Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 6)
  Call vxTableField(1, "Serial", "vxser", VX_FIELD)
  Call vxTableField(2, "Name", "vxname", VX_FIELD)
  Call vxTableField(3, "Company", "vxcompany", VX_FIELD)
  Call vxTableField(4, "Phone", "vxphone", VX_FIELD)
  Call vxTableField(5, "City", "vxcity", VX_FIELD)
  Call vxTableField(6, "Country", "vxcountry", VX_FIELD)

  Call vxBrowseCase(VX_UPPER)
  Call vxBrowseSetup(0, 0, 1, 1, "Arial Narrow", 15, VX_SEMIBOLD,
                     FALSE, 0, 0, 0)
  ' the fontsize param of 15 comes out as about 8 point type
  ' on an SVGA at 1024/768 res

  ' change the mousepointer for the text box to an arrow
  BrowseBox.MousePointer = 1
End Sub


' call vxCtlBrowse from the form_paint after
' the form has been initialized and displayed
' ------------------------------------------
Sub Form_Paint ()
  j% = vxSelectDbf(vxClientDbf)
  Call vxFormFrame(VXFORMX.hWnd)
  Call vxCtlStyle(BrowseBox, VX_RECESS)
  j% = vxCtlBrowse(vxCtlHwnd(BrowseBox), vxClientDbf, vxCl1Ntx,
                   TRUE, 0, vxCtlHwnd(MemoBox), "vxmemo")
End Sub

' to redraw formframe if resized
' ------------------------------
Sub Form_Resize ()
  VXFORMX.Refresh
End Sub

' important to close vxCtlBrowse in form unload
' ---------------------------------------------
Sub Form_Unload (Cancel As Integer)
  k& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_CLOSE, 0)
  j% = vxSelectDbf(vxClientDbf)
  j% = vxClose()
  vxWindowDereg (VXFORMX.hWnd)
End Sub


' message testing functions
' invoked when buttons clicked
' ----------------------------
Sub ButtonExit_Click ()
  Unload VXFORMX
End Sub

Sub ButtonSearch_Click ()
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_SEARCHDLG, 0)
End Sub

Sub ButtonAgain_Click ()
```

```
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_SEARCHAGAIN, 0)
End Sub

Sub ButtonCase_Click ()
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_CASE, 0)
End Sub

Sub ButtonCurRec_Click ()
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_GETCURRENTREC, 0)
  Debug.Print j&
End Sub

Sub ButtonTopRec_Click ()
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_GETTOPREC, 0)
End Sub

Sub ButtonFilter_Click ()
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_FILTERDLG, 0)
End Sub

Sub ButtonSeek_Click ()
  SeekKey$ = InputBox$("vxBase License?", "SearchKey", "")
  If EmptyString(SeekKey$) Then
     Exit Sub
  End If
  SeekKey$ = UCase$(SeekKey$)
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_SEEK, ByVal SeekKey$)
End Sub

Sub ButtonRefresh_Click ()
  SeekRec$ = InputBox$("goto record?", "Refresh", "")
  If EmptyString(SeekRec$) Then
     Exit Sub
  End If
  GoRec& = Val(SeekRec$)
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_REFRESH, ByVal GoRec&)
End Sub

Sub ButtonStrSearch_Click ()
  SeekStr$ = InputBox$("Search string?", "Search For String", "")
  If EmptyString(SeekStr$) Then
     Exit Sub
  End If
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_SEARCHPRG, ByVal SeekStr$)
End Sub

Sub ButtonPrgFilt_Click ()
  Filt$ = "trim(vxcountry)='U.S.A.'"
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_FILTERPRG, ByVal Filt$)
End Sub

Sub ButtonCancFilt_Click ()
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_FILTERPRG, ByVal 0&)
End Sub

' ------------------------------------------------
' KEY EVENTS Passed on to VB
' Enter (13) and escape (27) key presses initiated
' during the browse may be interrogated here.
' ------------------------------------------------
Sub BrowseBox_KeyPress (KeyAscii As Integer)
  If KeyAscii = 13 Then
     MsgBox "Enter key pressed"
     ' do your update or expansion routine here
     Debug.Print vxRecNo()
     KeyAscii = 0
  Else
```

```
     Debug.Print KeyAscii
  End If
End Sub

' Insert and Delete keys (45 and 46) will show up here
' As well as KeyCode 4 when a rec is highlighted
' ------------------------------------------------
Sub BrowseBox_KeyDown (KeyCode As Integer, Shift As Integer)
  ' insert key?
  If Keycode = 45 Then
     AddRec
  End If

  ' delete key?
  If KeyCode = 46 Then
     DeleteRec
  End If

  ' record highlighted in browse?
  If Keycode = 4 Then
     vxGo(vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_GETCURRENTREC, 0))
     DisplayRec
  End If
End Sub
```

See Also

vxBrowse, vxBrowseSetup, vxCtlBrowse, vxSetRelation, vxTableDeclare, vxTableField, vxTableFieldExt, vxTableReset

vxCtlFormat

Declaration

Declare Function vxCtlFormat Lib "vxbase.dll" () (ByVal TextLen As Integer, ByVal Picture As Integer, ByVal Decimals As Integer) As Integer

Purpose

Control the format of text entry into Visual Basic form text boxes.

Parameters

TextLen is an integer defining the number of characters that may be entered into the text box. Maximum length for VX_UPPER/VX_ALPHA fields is 255. Maximum numeric field length is 19. Maximum data field length is 8. If these lengths are exceeded, vxBase sets the default lengths to the maximum.

Picture is an integer that describes the type of data that may be entered into the text box. The data types are defined as Global identifiers in VXBASE.TXT.

VX_UPPER = 0: converts all lowercase characters to uppercase as they are typed.
VX_CHAR = 1: accepts all characters with no conversion (see also vxCtlLength).
VX_ALPHA = 2: only accepts alhabetic characters (both upper and lower case).
VX_NUM = 3: accepts only numbers, a minus sign, and a decimal point. Numeric fields must also be entered in the correct format (i.e., the only characters other than numbers that may be entered into a numeric box are a minus sign (-) in the first position aand a decimal character as defined in the internation section of the WIN.INI file). Only one sign and 1 decimal are allowed.
VX_DATE = 4: accepts and validates dates in a format as defined by vxSetDate (default VX_AMERICAN MM/DD/YY).
VX_PASSWORD = 5: displays all typed characters as asterisks (*) but accepts any character.

Decimals is an integer that defines the number of decimal places allowed in a numeric field. The maximum number of decimals allowed is 17.

Returns

TRUE if the format was successful and FALSE if not. FALSE is returned if the maximum number of active controls is exceeded (256) or if we run out of memory.

Usage

Use vxCtlFormat in the GotFocus() event procedure for the text box in which you wish to control the format.

The maximum number of active controls that may be formatted with vxCtlFormat is 256.

IMPORTANT NOTE: Always use vxWindowDereg in your Form Unload

procedure to release the memory vxBase allocates to the formatting routine and to reset the control procedure address.

If text is formatted as VX_UPPER, VX_ALPHA, VX_CHAR, or VX_PASSWORD, characters are converted as they are typed. VX_DATE and VX_NUM formats only allow numbers and delimiters as typing occurs.

VX_DATE and VX_NUM formats are validated when the control loses the focus. An error message box is presented if the entered data does not pass and the focus is reset to the offending control. For example, the number 123.45- is accepted in a numeric field as it is typed but when the control loses the focus, the user is informed that the sign must precede the number and focus is reset to the control. If an invalid date is entered, a date mask as defined by vxSetDate is inserted into the control after the user has been informed of the error.

PARAMETERS	CHARACTERS TYPED	RESULT
6,VX_UPPER,0	abC34F	ABC34F
6,VX_CHAR,0	abC34F	abC34F
8,VX_NUM,2	-123.456	-123.45 (decimals truncated)
	23	23.00 (trailing .00 added)
	23.	23.00 (trailing 00 added)
	123456.7	123456.70 (truncated left)
	123.4567	123.45 (truncated right)
8,VX_DATE,0	4/1/92	04/01/92 (leading zeroes added)
6,VX_PASSWORD,0	abC34F	****** (as viewed)
		abC34F (contents of text box)

Example

```
Sub NumField_GotFocus()
  j% = vxCtlFormat(vxFieldSize("numfield"), VX_NUM, 2)
End Sub
```

See Also

vxCtlLength, vxSetDate, vxWindowDereg

vxCtlGrayReset

Declaration

Declare Sub vxGrayReset Lib "vxbase.dll" ()

Purpose

Reset Windows Gray color for disabled items back to the system standard.

Parameters

None.

Returns

Nothing.

Usage

Only used if vxCtlStyle and vxFormFrame are called to give your application a metallic, three-dimensional look (VGA/SVGA only). When using this style of form, the backgrounds of both forms and controls are painted light gray - the same light gray used by Windows to show that text and controls have been disabled. Disabled items therefore disappear into the background.

At the start of our application, we issue a vxCtlGraySet to set the disabled color to a darker gray and we use vxCtlGrayReset to set it back when we exit. The disabled gray color is a Windows System Color and as such it affects every other application you may have running as well.

Note: This command has no effect if the system is not running on a VGA or SVGA monitor.

Example

```
Sub Form_Unload (Cancel As Integer)
   If Not vxCloseAll() Then
      Cancel = -1
      VXFORM1.Show       ' redraw top level form
      Exit Sub
   Else
      ' we MUST test the result of vxDeallocate
      ' to ensure that the task is not controlling
      ' memory for any other vxBase tasks that
      ' might be running at the same time as this one
      ' -------------------------------------------
      If Not vxDeallocate() Then
         Cancel = -1
         VXFORM1.Show
      Else
         vxCtlGrayReset
      End If
   End If
End Sub
```

See Also

vxCtlGraySet, vxCtlPenWidth, vxCtlStyle, vxFormFrame

vxCtlGraySet

Declaration

Declare Sub vxCtlGraySet Lib "vxbase.dll" ()

Purpose

Set the Windows System color for disabled items to dark gray.

Parameters

None.

Returns

Nothing.

Usage

Only used if vxCtlStyle and vxFormFrame are called to give your application a metallic, three-dimensional look (VGA/SVGA only). When using this style of form, the backgrounds of both forms and controls are painted light gray - the same light gray used by Windows to show that text and controls have been disabled. Disabled items therefore disappear into the background.

At the start of our application, we issue a vxCtlGraySet to set the disabled color to a darker gray and we use vxCtlGrayReset to set it back when we exit. The disabled gray color is a Windows System Color and as such it affects every other application you may have running as well.

The gray settings are done at the start and end of the application because the entire screen is repainted whenever we set a system color.

Note: This command has no effect if the system is not running on a VGA or SVGA monitor.

vxCtlGraySet affects all Windows system colors. As such, every running application has its windows repainted when this command is issued. Some background tasks (such as Norton Desktop for Windows) will come to the foreground and overlay your vxBase task when it is run as an .EXE. To stop this, simply issue two calls to vxCtlGraySet in succession.

Example

```
' register task and
' set system gray color with the
' first form we load so disabled
' items on our gray forms will not
' disappear
' -------------------------------
Sub Form_Load
   Call vxInit
   Call vxCtlGraySet
End Sub
```

See Also

vxCtlGrayReset, vxCtlPenWidth, vxCtlStyle, vxFormFrame

vxCtlHwnd

Declaration

Declare Function vxCtlHwnd Lib "vxbase.dll" (ControlName As Any) As Integer

Purpose

Convert a Visual Basic control handle into a Window handle.

Parameters

ControlName is the name of a Visual Basic control.

Returns

An integer that contains the window handle of the control.

Usage

Must be used to pass a window handle to vxCtlBrowse and vxCtlBrowseMsg so they can do their duty. May also be used to access Windows API calls that only work on window handles and not VB control handles.

Restriction

For Visual Basic users only.

Example

```
Sub ButtonStrSearch_Click ()
  SeekStr$ = InputBox$("Search string?", "Search For String", "")
  If EmptyString(SeekStr$) Then
     Exit Sub
  End If
  j& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_SEARCHPRG,
                      ByVal SeekStr$)
End Sub
```

See Also

vxCtlBrowse, vxCtlBrowseMsg

vxCtlLength

Declaration

Declare Sub vxCtlLength Lib "vxbase.dll" (ByVal FieldName As String)

Purpose

Set the maximum number of characters that can be entered by the user in a data entry box equal to the xBase field size.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

Nothing.

Usage

If used, this function must be placed in the GotFocus event procedure for each control to set the maximum number of characters that can be entered into a text box. The text box must of course be associated with a vxBase field.

If you require fomatted text, use vxCtlFormat instead, which also sets the maximum text length.

Restriction

This function is restricted to Visual Basic users only.

Example

```
Sub TypeCode_GotFocus ()
   ' set up text length limit
   ' ------------------------
   Call vxCtlLength("category")
End Sub
```

See Also

vxCtlFormat, vxSetAlias

vxCtlPenWidth

Declaration

Declare Sub vxCtlPenWidth lib "vxbase.dll" (ByVal PenWidth As Integer)

Purpose

Control the depth of recessed or raised controls when using vxCtlStyle.

Parameters

Penwidth is either 1, 2, or 3. The default value is 2.

Returns

Nothing.

Usage

Primarily to make vxBase styled text boxes look the same as other third party controls (e.g., 3dWidgets from Sheridan Software).

Example

```
Call vxInit
Call vxCtlGraySet
Call vxCtlPenWidth(1)
```

See Also

vxCtlStyle

vxCtlStyle

Declaration

Declare Sub vxCtlStyle Lib "vxbase.dll" (ControlName As Any, ByVal Mode As Integer)

Purpose

Draw a frame around a control that gives it a three-dimensional look.

Parameters

ControlName is the name of your form control.

Mode is one of the Global Constants defined in vxbase.txt that defines the drawing style. VX_RECESS (value 1) gives the control a recessed look. VX_RAISE (value 0)raises the control away from the form, VX_CREASE (value 2) gives the control a creased border, and VX_FLAT (value 3) flattens recessed or raised controls.

Returns

Nothing.

Usage

Gives your application a metallic, three-dimensional look (on VGA/SVGA monitors only). The three-dimensional depth is controlled via vxCtlPenWidth. Follow these steps in designing a form with this style.

(1) Lay out your form as usual, in black and white. Group boxes and related items (even groups of buttons) may be placed inside picture boxes and then the picture boxes may be raised for effect.

(2) When satisfied with your item placement and font selection, color the backgound of the form and every control a light gray with the Window Color Palette. You may wish to make the text of labels a color other than black to distinguish them from the data entered in their related text boxes.

(3) remove the borders from picture boxes and text boxes that you are going to paint with vxCtlStyle. You can't remove borders from list boxes and group boxes. It is not absolutely necessary to do this. I just think it looks better. If you disagree, leave the borders on. Try it both ways. (If your application is run on an EGA monitor, vxCtlStyle draws black borders around the controls instead of making them appear three-dimensional).

(4) use the Form_Paint procedure to draw the controls as in the example below. If any form in your application contains disabled controls, make sure you use vxCtlGraySet at the start of your application to change the disabled color to a darker gray or the text of your disabled controls will disappear into the light gray background.

When using a Form_Paint procedure, it is important to understand the sequence of painting events that results in the completed display.

The Form_Load procedure is executed first. Your Form_Load procedure

does not display the form. You normally use this procedure to initialize values that will appear in the form data boxes.

After the Form_load procedure, controls that have had values assigned are given the focus and the data is inserted in the boxes.

Windows issues an internal WM_PAINT message to draw the form before Visual Basic receives a Form_Paint message.

After your form has been painted, we can use the Form_Paint procedure to enhance our controls.

What this really means is that you cannot use a Control_GotFocus event to do anything that will affect the appearance of the form. For example, if you had a browse table up and the user selected the Delete record item from the browse menu, a good place to test for this would be in the GotFocus event procedure for the first control on the form. We could then solicit a Deletion Confirmation from the user. We wouldn't test if Delete had been selected in the Form_Load procedure because the data hasn't been displayed yet and we would like the user to see the record he is deleting before we ask for verification. But if we are using the enhanced controls that vxCtlStyle provides, the Visual Basic Form_Paint event hasn't occurred yet so we would get a flat form overlaid by our Confirmation message box. Not pretty.

Instead, we can test for the Delete message in the Form_Paint procedure itself after the control borders have been drawn by vxCtlStyle, as in the example below. Keep this sequence in mind when you contemplate initialization procedures during any event that occurs after the first Windows painting of the form and before Visual Basic is informed of the Form_Paint event.

Restriction

This function is restricted to Visual Basic users only.

Example

```
Sub Form_Paint ()
   Call vxFormFrame(VXFORM2.hWnd)
   Call vxCtlStyle(TypeCode, VX_RECESS)
   Call vxCtlStyle(TypeDesc, VX_RECESS)
   Call vxCtlStyle(TypeStatus, VX_RAISE)

   ' if delete request from browse, do it now
   ' because we must let enhanced controls
   ' paint before asking for delete confirmation
   ' ------------------------------------------
   If TypeReturn = BROWSE_DELETE Then
      TypeDelete_Click
   End If
End Sub
```

See Also

vxCtlGrayReset, vxCtlGraySet, vxCtlPenWidth, vxFormFrame

vxDateFormat

Declaration

Declare Function vxDateFormat Lib "vxbase.dll" (ByVal DateField As String) As String

Purpose

Convert an xBase date field to a Visual Basic date format that can be used by Visual Basic date arithmetic and formatting functions.

xBase dates are stored as 8 character long strings in the format CCYYMMDD.

Parameters

DateField is either a string variable or a literal string that contains a valid date field name from the currently selected database. *DateField* may be qualified with a valid alias name that points to any open database.

Returns

A Visual Basic string in the format DD-MMM-CCYY. For example, if the DTOS(date) in the database field is "19910722" then the returned value will be 22-Jul-1991. If the field name does not represent a date, or if it is empty, the value returned will be 01-Jan-1980.

Usage

This function must be used to convert a date into a format which Visual Basic can understand. Visual Basic contains a full complement of functions that perform date arithmetic so there is no need for vxBase to duplicate those functions.

Example

```
' vxDateFormat() routine returns a date in the
' format dd-mmm-yyyy, which the Visual Basic
' DateValue function inderstands. We will put
' the creation date into a variable so we can
' perform some date arithmetic on it to determine
' the number of days on file
' ------------------------------------------------
DateCreate$ = vxDateFormat("a_cdate")
DaysOnFile% = (DateValue(Date$) - DateValue(DateCreate$))
              + 1

CustCdate.text = DateCreate$
CustRdate.text = vxDateFormat("a_rdate")
CustDays.text = Format$(DaysOnFile%, "###0")
```

See Also

vxDateString, vxReplDate, vxReplDateString, vxSetAlias, vxSetDate

vxDateString

Declaration

Declare Function vxDateString Lib "vxbase.dll" (ByVal DateField As String, ByVal DateType As Integer) As String

Purpose

Convert an xBase date field to a standard display style date formatted according to country specific conventions.

Parameters

DateField is either a string variable or a literal string that contains a valid date field name from the currently selected database. DateField may be qualified with a valid alias name that points to any open database.

DateType is a country identifier as defined in vxbase.txt. It is one of the following:

```
VX_AMERICAN format mm/dd/yy
VX_ANSI     format yy.mm.dd
VX_BRITISH  format dd/mm/yy
VX_FRENCH   format dd/mm/yy
VX_GERMAN   format dd.mm.yy
VX_ITALIAN  format dd-mm-yy
VX_SPANISH  format dd-mm-yy
```

Returns

A Visual Basic string in the format of the country specified. These dates may not be used with Visual Basic date arithmetic routines because the results are always ambiguous. Use vxDateFormat to extract an unambiguous date if date arithmetic is to be performed on the date. If the field name does not represent a date, or if it is empty, the value returned will be January 1, 1980.

Usage

Use this function only for form display purposes and for filling a data entry box with a string that will have its format controlled with vxCtlFormat.. Use vxDateFormat if date arithmetic is to be performed on the converted date.

Example

```
    ' format a date for display in a VB
    ' form textbox
    ' ---------------------------------
  CustRdate.text = vxDateString("a_rdate", VX_AMERICAN)
```

See Also

vxCtlFormat, vxDateFormat, vxDbfDate, vxReplDate, vxReplDateString, vxSetAlias, vxSetDate, vxSetLanguage

vxDbfCurrent

Declaration

```
Declare Function vxDbfCurrent lib "vxbase.dll" () As Integer
```

Purpose

Get the current database select area.

Parameters

None.

Returns

The current database select area (as reported by vxUseDbf or one of its variants when the file was opened). If there is no current select area active, FALSE (0) is returned.

Usage

Can be used to ensure that the database select area you THINK is active is really active when you are about to perform critical tasks (such as vxZap() or even vxClose()).

Example

```
If vxDbfCurrent() = MasterDbf Then
   j% = vxClose()
End If
```

See Also

vxAreaDbf, vxNtxCurrent, vxSelectDbf, vxUseDbf, vxUseDbfAgain, vxUseDbfEX, vxUseDbfRO

vxDbfDate

Declaration

```
Declare Function vxDbfDate Lib "vxbase.dll" () As String
```

Purpose

Extract the date of the last read/write access performed on the current database.

Parameters

None.

Returns

A Visual Basic string that contains the date that the file was last opened with vxUseDbf (and successfully closed with vxClose or vxCloseAll). The date returned is formatted according to the current vxSetDate value (default VX_AMERICAN MM/DD/YY).

Usage

Usually for display or print purposes.

Example

```
UpdateDate.text = vxDbfDate()
```

See Also

vxDbfName, vxSetDate

vxDbfName

Declaration

```
Declare Function vxDbfName Lib "vxbase.dll" () As String
```

Purpose

Extract the name of the currently selected database file.

Parameters

None.

Returns

A Visual Basic string that contains the name of the database file as it was passed to the vxUseDbf function or one of its variants when it was opened.

Usage

Usually for display or print purposes.

Example

```
NameControl.text = vxDbfName()
```

See Also

vxDbfDate, vxNtxName

vxDeallocate

Declaration

Declare Function vxDeallocate Lib "vxbase.dll" () As Integer

Purpose

Release global memory allocated to VB.EXE by vxBase when in Design Mode and test task closure sequence if multitasking vxBase applications.

Parameters

None.

Returns

TRUE if it is safe to unload the task and FALSE if not. FALSE is returned if this is the memory controlling task for a number of concurrently running vxBase tasks.

Usage

Always used as the last statement in your VB application. In Visual Basic Design Mode, this function releases memory allotted to VB.EXE. If this statement does not appear as the last statement in your application, repeated test runs while in design mode will cause VB.EXE to grow in memory by about 130k per repetition until you eventually run out of memory unless VXLOAD.EXE is running as suggested. This procedure only works if the Visual Basic Design mode window (which includes the VB main menu) is visible on your screen when the Run command (or F5) is issued. If running a compiled vxBase application, this function tests if it is safe to unload the current application. See the Multitasking and Multiuser Considerations section for a complete explanation of vxBase memory sharing. Also see the section entitled "Visual Basic and VXLOAD.EXE".

Example

```
Sub Form_Unload (Cancel As Integer)
  If Not vxCloseAll() Then
     Cancel = -1
     VXFORM1.Show       ' redraw top level form
     Exit Sub
  Else
     ' we MUST test the result of vxDeallocate
     ' to ensure that the task is not controlling
     ' memory for any other vxBase tasks that
     ' might be running at the same time as this one
     ' -------------------------------------------
     If Not vxDeallocate() Then
        Cancel = -1
        VXFORM1.Show
     Else
        vxCtlGrayReset
     End If
  End If
End Sub
```

See Also

vxCloseAll, vxInit

vxDecimals

Declaration

Declare Function vxDecimals Lib "vxbase.dll" (ByVal FieldName As String) As Integer

Purpose

Extract the number of decimal positions defined for the specified field.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. The field should be numeric, although a zero will be returned for any other field type. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

An integer that contains the number of decimal positions.

Usage

Usually extracted to help in data validation.

Example

```
Sub BuyHigh_KeyPress (KeyAscii As Integer)
  ' Treat enter key as a tab
  ' ------------------------
  If KeyAscii = 13 Then
     KeyAscii = 0
     SendKeys "{Tab}"
     Exit Sub
  End If

  ' if there are any decimals defined, allow decimal point
  ' -----------------------------------------------------
  If vxDecimals("b_high") > 0 And KeyAscii = Asc(".") Then
     Exit Sub
  End If

  ' limit key presses to numbers
  ' ---------------------------
  If KeyAscii < Asc("0") Or KeyAscii > Asc("9") Then
     KeyAscii = 0
     Beep
  End If
End Sub
```

See Also

vxFieldSize, vxFieldType, vxSetAlias

vxDeleted

Declaration

```
Declare Function vxDeleted Lib "vxbase.dll" () As Integer
```

Purpose

Determine whether a record from the currently selected database has been logically deleted or not.

Parameters

None.

Returns

TRUE if the record has been deleted, and FALSE if not.

Usage

When xBase records are deleted with the vxDeleteRec function, they are only logically deleted. Every record has a Deletion Flag field as the first byte in the record. If the vxDeleteRec function is used to delete the record, the flag is changed from a space to an asterisk "*". vxBrowse automatically filters these records. If the programmer is using other record movement schemes, it is his responsibilty to ensure that deleted records are ignored when they are supposed to be, or to report the fact that the record has been deleted to the end user.

Deleted records are physically removed from a file only by packing it or copying it using vxCopy.

A filter can be set to ignore deleted records with the vxFilter function.

Example

```
' standard skip loop
' ------------------
Do
   j% = vxSkip(1)
   If j% = FALSE Then
      MsgBox "Error on Skip. Try Reindex."
      Exit Sub
   End If
   If vxEof() Then Exit Do
Loop Until Not vxDeleted()
```

See Also

vxCopy, vxDeleteRange, vxDeleteRec, vxPack, vxRecall, vxZap

vxDeleteRange

Declaration

Declare Function vxDeleteRange Lib "vxbase.dll" (ByVal StartRec As Long, ByVal EndRec As Long) As Integer

Purpose

Physically remove the specified range of records from the currently selected database.

Parameters

StartRec is the record number of the first record to delete. *EndRec* is the last record number in the range.

Returns

TRUE if the operation was successful and FALSE if not. Always FALSE if the file was opened as Read Only with vxUseDbfRO.

Usage

StartRec must be less than or equal to **EndRec**. The record numbers refer to the physical locations of the records. If an index is in use, it is deselected prior to the commencement of the operation. If one or more indexes are in use, the file is reindexed after the range of records has been removed.

Multiuser Considerations

The file and its indexes are locked for the duration of the operation.

Example

```
j% = vxBottom()
OldLastRec& = vxRecNo()
j% = vxAppendFrom("Transfil.dbf")
j% = vxBottom()
NewLastRec& = vxRecNo()
j% = MsgBox("Everything OK?", 52)
If j% = 6 Then
   j% = vxClose()
   Kill "Transfil.dbf"
Else
   j% = vxDeleteRange(OldLastRec& + 1, NewLastRec&)
   j% = vxClose()
End If
```

See Also

vxDeleteRec, vxPack, vxZap

vxDeleteRec

Declaration

Declare Function vxDeleteRec Lib "vxbase.dll" () As Integer

Purpose

Logically delete the current record from the currently selected database.

Parameters

None.

Returns

TRUE if the operation was successful and FALSE if not. Always FALSE if the file was opened as Read Only with vxUseDbfRO.

Usage

This function sets the Delete Flag field that is present at the front of every xBase record to '*', which logically deletes the record. The record is still available for use by every function except vxBrowse, which filters all deleted records.

The record may be recalled with the vxRecall function.

Records deleted with vxDeleteRec may be physically removed from the file with function vxPack or function vxCopy.

The programmer is responsible for skipping by deleted records when moving the record pointer. Alternatively, a filter may be set on the file with vxFilter that masks deleted records from the vxSkip and vxSeek functions.

Example

```
Sub TypeDelete_Click ()

  ' get user confirmation of delete
  ' -------------------------------
  j% = MsgBox("Confirm Delete", 52)
  If j% = 6 Then
     If vxDeleteRec() Then
        TypeDataClear
        TypeStatus.text = "Rec " + LTrim$(Str$(vxRecNo()))
                          + " Deleted"
     Else
        TypeStatus.text = "Delete failed"
     End If
  Else
     TypeStatus.text = "Delete cancelled"
  End If
End Sub
```

See Also

vxCopy, vxDeleted, vxDeleteRange, vxPack, vxRecall, vxZap

vxDescend

Declaration

Declare Function vxDescend Lib "vxbase.dll" (ByVal KeyString As String) As String

Purpose

Create a search key for use in seeking records indexed with the xBase DESCEND() function.

Parameters

KeyString is either a literal string or string variable that contains the value to be converted into DESCEND() format.

Returns

A Visual Basic string containing a complemented representation of KeyString.

Usage

This function must be used to create search keys if you are attempting to find records in an index built with the xBase DESCEND() function.

If you are browsing a file with a key built with the DESCEND() function as the first part of the key, be sure to turn Quick Key off in vxTableDeclare (set parameter Quick to zero). Quick Key searches in a browse window will not work on DESCENDing key elements.

Example

```
AirbuyerDbf = vxUseDbf("\vb\airtypes.dbf")
DescNtx = vxCreateNtx("\vb\airdown.ntx", "DESCEND(UPPER(b_cat))")
If vxSeek(vxDescend("P15")) Then
   DisplayBuyRec
Else
   MsgBox "Record Not Found"
End If
```

See Also

vxCreateNtx, vxSeek, vxSeekSoft

vxDouble

Declaration

Declare Sub vxDouble Lib "vxbase.dll" (ByVal FieldName As String, DblAmount As Double)

Purpose

Convert a numeric field to a Visual Basic double value.

Parameters

FieldName is either a string variable or a literal string that contains a valid numeric field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

DblAmount is a predimensioned double value that will receive the result of the function. See the example below.

Returns

A double value in the *DblAmount* parameter.

Usage

Unlike other field reference functions, this is a procedure that must be CALLed. The user is responsible for passing a predefined double variable to vxDouble, which receives the result of the procedure call.

The format of this function has to do with Borland C++, phantom parameters, and Bad DLL Calling Conventions, which you probably don't want to know about. Unfortunately, this is the only way I could get it to work.

Example

```
Sub BuyerDataLoad ()
  Dim b_low As Double
  Dim b_high As Double

  CursorWait
  EnableBuyerData
  Call vxDouble("b_low", b_low)
  Call vxDouble("b_high", b_high)
  BuyLow.text = Format$(b_low, "#######0")
  BuyHigh.text = Format$(b_high, "#######0")
  BuyType.text = vxField("b_cat")
  BuyTypeDesc.text = vxField("b_desc")
  BuyCode.text = vxField("b_code")
  CursorArrow
End Sub
```

See Also

vxField, vxFieldTrim, vxInteger, vxLong, vxReplDouble, vxReplString, vxSetAlias

vxEmpty

Declaration

Declare Function vxEmpty Lib "vxbase.dll" (ByVal FieldName As String) As Integer

Purpose

Test if a character field is filled with spaces or if a numeric field is zero.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

TRUE if the character field has nothing but spaces in it or if a numeric field evaluates to zero. FALSE if the field contains something. The function will actually work on any kind of field (including date, logical, and memo fields) and return TRUE if the field is composed entirely of spaces.

Usage

Normally used to control processing of controls depending on whether something has been entered or not.

Example

```
' if the code has already been entered, don't
' allow the user to edit it
' ----------------------------------------
If vxEmpty("buy_code") Then
   BuyCode.Enabled = TRUE
   BuyCode.text = ""
Else
   BuyCode.Enabled = FALSE
   BuyCode.text = vxField("buy_code")
End If
```

See Also

vxChar, vxField, vxFieldTrim, vxSetAlias

vxEof

Declaration

```
Declare Function vxEof Lib "vxbase.dll" () As Integer
```

Purpose

Test for end of file.

Parameters

None.

Returns

TRUE if the record pointer has been moved past the last record in the file and FALSE if not.

Usage

When skipping through a file in the forward direction, always use vxEof to test if the last record has been read. If vxEof is TRUE, the record buffer will point to an empty record (which can't be used for anything).

Example

```
' skip forward one record
' ------------------------
Do
   j% = vxSkip(1)
   If j% = FALSE Then

      ' if skip error, only allow exit
      ' ------------------------------
      MsgBox "Error on Skip Next. Try Reindex."
      TypeDataClear
      Exit Sub
   End If
   If vxEof() Then Exit Do
Loop Until Not vxDeleted()

' test for end of file
' --------------------
If vxEof() Then
   Beep
   TypeStatus.text = "End of File!"
   j% = vxBottom()  ' go back to last record
Else
   TypeStatus.text = "Skipped to record " +
                     LTrim$(Str$(vxRecNo()))
End If
TypeDataLoad
```

See Also

vxBof

vxErrorTest

Declaration

Declare Function vxErrorTest Lib "vxbase.dll" (ErrorStructure As vxErrorStruc) As Integer

Purpose

Test if an error occurred in a vxBase function. This function only works if vxSetErrorMethod is set to TRUE. vxSetErrorMethod and vxErrorTest provide an alternate error handling procedure to the standard vxBase error routine. If vxSetErrorMethod is FALSE, vxBase errors are reported through an immediate run time message box.

Parameters

ErrorStructure is of type vxErrorStruc as defined in the global module (see Example below).

Returns

TRUE if an error occurred in the previous call to vxBase and FALSE if no error occurred.

Usage

Visual Basic 1.0 does not provide a method for a DLL to trigger a Visual Basic Error. As a result, the standard ON ERROR method will not work to trap errors found by a DLL. Visual Basic 2.0 does provide a standard hook into the ON ERROR method and therefore simplifies the processing of the alternate error method.

Visual Basic 1.0

By using vxSetErrorMethod(TRUE) and vxErrorTest(vxError), the programmer may trap errors and execute an error procedure instead of being caught in a no win situation (such as a series of field extraction commands that are issued to a non-selected database which results in a seemingly never-ending sequence of the same error message box). If an error occurs within a vxBase function, you may exit to a special vxBase error handling procedure and, depending on the error reported, either END the program or continue by returning to the statement following the vxErrorTest. vxErrorTest must be called after every vxBase function call that could result in a runtime error.

Visual Basic 2.0

If vxSetErrorMethod is set to TRUE with Visual Basic 2.0, the standard ON ERROR method event is triggered and vxErrorTest is only called in the defined error routine instead of after every call to vxBase that could result in a runtime error. Your defined error routine must still interrogate the vxErrorStructure to determine the type of error that occurred.

Errors that occur within high level vxBase functions (vxBrowse, vxMemoEdit, vxCtlFormat) are NOT trappable with vxErrorTest. Errors that occur in these functions cannot be foreseen by the programmer (such as an invalid date entry into a vxCtlFormat date text box) and must be dealt with at the user level.

The error structure defined as Global var vxError (as type vxErrorStruc) has the following elements:

vxError.ErrorNum	vxBase error number as listed in Appendix A
vxError.ErrorMsg	vxBase error message as listed in Appendix A or in the language selected with vxSetLanguage. This is a FIXED string and must be RTRIMmed before use.
vxError.DbfArea	the currently selected dbf area (0 if none)
vxError.NtxArea	the currently selected ntx area (0 if none)
vxError.DbfName	the name of the current dbf (blank if none)
vxError.NtxName	the name of the current ntx (blank if none)
vxError.BadParm	an extra information field that contains variable data depending on the type of error that occurred. For example, if an invalid field name is passed to vxBase, that invalid field name will appear here. If there is no extra info that would have any validity, the element is blank. If the extra info is longer than 79 characters (e.g., xbase expressions) it is truncated on the right.

A call to vxErrorTest resets the internal vxBase error flag. Subsequent calls to vxErrorTest when no error has occurred will always return FALSE. Only the LAST error that occurred is available at any given time in vxError unless you wish to save the returned error structure in different variables.

Example

The error structure, function declaration, and global variable must be defined in the global module as follows (and in the same order):

```
Type vxErrorStruc
   ErrorNum As Integer
   ErrorMsg As String * 80
   DbfArea As Integer
   NtxArea As Integer
   DbfName As String * 80
   NtxName As String * 80
   BadParm As String * 80
End Type

Declare Function vxErrorTest Lib "vxbase.dll"
       (ErrorStructure As vxErrorStruc) As Integer
' (above declaration on ONE line)

Global vxError As vxErrorStruc
```

Visual Basic 1.0

```
' Trapping the error in the FORM code
' ----------------------------------
Call vxSetErrorMethod(TRUE)
jj% = vxUseNtx("\vb\vxbtest\testerr.ntx")
If vxErrorTest(vxError) Then
   ProcessError
```

```
  End If
  Call vxSetErrorMethod(FALSE)

  ' processing the error in a General Procedure
  ' ------------------------------------------
  Sub ProcessError ()

     Select Case vxError.ErrorNum
        ' 620 File Open
        Case 620
           MsgBox "vxBase TEST: file open error"
           END

        ' 944 Invalid Field Name
        Case 944
           MsgBox "Bad Field " + RTrim$(vxError.BadParm)
           END

        Case Else
           MsgBox RTrim$(vxError.ErrorMsg)
     End Select

  ' see Appendix A in the vxBase manual
  ' for a description of all errors

  ' identify what you feel are catastrophic
  ' errors (like a 620 error) and abort
  ' the program run entirely with an END
  ' statement

  End Sub
```

Visual Basic 2.0

```
  ' VB 2.0 uses the standard VB On Error method
  ' ------------------------------------------
  On Error GoTo VBErrorRtn
  Call vxSetErrorMethod(True)
  jj% = vxUseNtx("\vb\vxbtest\testerr.ntx")
  ...
  ...
  ...
  Call vxSetErrorMethod(False)
  ...
  ...
  ...

VBErrorRtn:
  MsgBox "vxBase error encountered"
  If vxErrorTest(vxError) Then
     ProcessError
  End If
  Resume Next
End Sub
```

See Also

vxSetErrorMethod

vxEval

Declaration

Declare Function vxEval Lib "vxbase.dll" (ByVal xBaseExpr As String) As Integer

Purpose

Test if a user entered xBase expression will properly evaluate.

Parameters

xBaseExpr is a character expression formatted using xBase syntax.

Returns

TRUE if the expression will evaluate and FALSE if it contains syntax errors or errors of any other type that the vxBase parser uncovers.

Note that this function does not return the result of the expression. Rather, it simply checks to see if the expression will evaluate correctly once it is in use.

Usage

You may solict xBase expressions from the user in the form of filters, index expressions, browse table column definitions, etc. If an error is uncovered during expression evaluation, the user is informed via vxBase error message windows but the programmer has no way of knowing that the expression doesn't pass unless he parses it first with vxEval.

NOTE: The record buffer must be filled with a valid record from the database that the expression applies to BEFORE this function is called. The database must be open and currently selected.

Example

```
' test user entered filter expression
' ------------------------------------
j% = vxGo(RecNum&)
If Not vxEval((UserFilter.Text)) Then
   MsgBox "Re-Enter filter"
   UserFilter.SetFocus
   Exit Sub
Else
   Call vxFilter((UserFilter.Text))
   Call BrowseFile
End If
```

See Also

vxCreateNtx, vxEvalDouble, vxEvalLogical, vxEvalString, vxFilter, vxTableField

vxEvalDouble

Declaration

Declare Function vxEvalDouble Lib "vxbase.dll" (ByVal xBaseExpr as String, DblAmount As Double) As Integer

Purpose

Evaluate an xBase expression that returns a numeric value and store the result of the evaluation in a predefined Visual Basic double variable.

Parameters

xBaseExpr is a valid xBase expression that will return a numeric result.

DblAmount is a predefined double value that will receive the result of the xBase expression.

Returns

TRUE if the expression is a valid NUMERIC xBase expression. FALSE is returned if the expression cannot be parsed or if the expression type is not numeric.

NOTE: The record buffer must be filled with a valid record from the database that the expression applies to BEFORE this function is called. The database must be open and currently selected.

Example

```
' use vxEvalDouble to calc capacity in
' lbs by subtracting empty weight from gross
' in an xBase expression instead of Vis Bas
' ----------------------------------------
NumVal = 0
If vxEvalDouble("c_gwt - c_ewt", NumVal) Then
   AirEmpty.text = Format$(NumVal, "####0")
Else
   MsgBox "Numeric expression eval error"
End If
```

See Also

vxEval, vxEvalLogical, vxEvalString

vxEvalLogical

Declaration

Declare Function vxEvalLogical Lib "vxbase.dll" (ByVal xBaseExpr as String, ByVal TrueFalse As String) As Integer

Purpose

Evaluate an xBase expression that returns a logical value and store the result of the evaluation in a predefined Visual Basic string. The result will be either ".T." for TRUE or ".F." for FALSE.

Parameters

xBaseExpr is a valid xBase expression that will return a logical result.

TrueFalse is a predefined string that is 4 characters long that will receive the result of the xBase expression (either ".T." or ".F".).

Returns

TRUE if the expression is a valid LOGICAL xBase expression. FALSE is returned if the expression cannot be parsed or if the expression type is not logical.

Notice that the integer return value is NOT the same as the result of the expression evaluation. If the expression is logical and evaluates as ".F.", the function will return TRUE but the value in the predefined Visual Basic string variable will be ".F.".

Usage

NOTE: The record buffer must be filled with a valid record from the database that the expression applies to BEFORE this function is called. The database must be open and currently selected.

Example

```
' example of xBase logical expression evaluation
' ----------------------------------------------
Eval$ = String$(4, 0)
If vxEvalLogical("left(category,1)='C'", Eval$) Then
   EvalBox.Text = Eval$
Else
   MsgBox "Error in logical expression evaluation"
End If
```

See Also

vxEval, vxEvalDouble, vxEvalString

vxEvalString

Declaration

Declare Function vxEvalString Lib "vxbase.dll" (ByVal xBaseExpr as String, ByVal StringVal As String) As Integer

Purpose

Evaluate an xBase expression that returns a string value and store the result of the evaluation in a predefined Visual Basic string.

Parameters

xBaseExpr is a valid xBase expression that will return a character result.

StringVal is a predefined string that will receive the result of the xBase expression. It MUST be long enough to hold the xBase result.

Returns

TRUE if the expression is a valid CHARACTER xBase expression. FALSE is returned if the expression cannot be parsed or if the expression type is not character.

Usage

NOTE: The record buffer must be filled with a valid record from the database that the expression applies to BEFORE this function is called. The database must be open and currently selected.

Example

```
' example of xBase string expression evaluation
' -------------------------------------------
EvStr$ = String$(64, 0)
If vxEvalString("trim(catname)+' is category '+category",EvStr$) Then
   JoinBox.Text = EvStr$
Else
   MsgBox "Error in string expression evaluation"
End If
```

See Also

vxEval, vxEvalDouble, vxEvalLogical

vxExactOff

Declaration

```
Declare Sub vxExactOff Lib "vxbase.dll" ()
```

Purpose

Turns the vxExactOn requirement OFF when using vxSeek.

Parameters

None.

Returns

Nothing. Sets an internal switch only.

Usage

Sets the ExactOn switch to OFF. OFF is the default value of this switch. See vxExactOn for more details on exactly what it does.

Example

```
vxExactOn
If vxSeek("ABC") Then
   UpdateProcedure
Else
   AddProcedure
End If
vxExactOff
```

See Also

vxExactOn, vxFound, vxSeek

vxExactOn

Declaration

```
Declare Sub vxExactOn Lib "vxbase.dll" ()
```

Purpose

Sets the internal Exact switch ON.

Parameters

None.

Returns

Nothing. Internal switch setting only.

Usage

The status of the Exact switch controls whether or not vxBase will report a successful vxSeek on a record if a partial key match is found. For example, assume you have a customer key in the form "ABCDEF". The vxSeek parameter could be "A", "AB", "ABC" etc. up to "ABCDEF" and it will report the record found (if its the only one with an "A" in the first position). In other words, vxSeek("A") will find the first record in the file whose key begins with "A" if you pass it a single letter "A", no matter how long the key is. There are times when you may wish to only find a record whose key matches the vxSeek parameter exactly. This is when you use vxExactOn. Don't forget to turn it off or things won't work out exactly as you had planned.

If vxExactOn is TRUE, then a partially matched key will cause vxSeek to return FALSE, and vxFound will also return FALSE. The record pointer, however, will be set at the record whose key matched partially if that was the case and vxEOF will be FALSE. If no part of the key was found, vxEOF will be TRUE, vxFound will be FALSE, and the record pointer will be pointing nowhere.

Example

```
vxExactOn
If vxSeek("ABC") Then
   UpdateProcedure
Else
   AddProcedure
End If
vxExactOff
```

See Also

vxExactOff, vxSeek, vxSeekSoft

vxField

Declaration

Declare Function vxField Lib "vxbase.dll" (ByVal FieldName As String) As String

Purpose

Extract an xBase field and convert it to a Visual Basic string.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

A Visual Basic string that contains the contents of the defined field.

Usage

Mostly used to get the contents of a character type field. Note, however, that all xBase data is kept in character format, so you can use this function to extract any field - including numeric, date, and logical fields (and even a memo block reference if you wish). You could then use Visual Basic data conversion functions to create the type of data you are interested in.

NOTE 1: The maximum length of a field that can be extracted using vxField is 255 characters. If the field is longer than 255, use vxRecord to extract the entire record contents into a defined record type or string.

NOTE 2: The record buffer must be filled prior to calling any vxBase field extraction function. When a file is opened, the record pointer is positioned to the top of the file and the first physical record is read into the buffer. If an index is opened, the first logical record is read into the buffer. Any field extraction operations performed on this buffer will always return the same values until another explicit operation is called that reads a record into the buffer (e.g., vxGo, vxTop, vxBottom, vxSkip, vxSeek, vxSeekSoft, etc.). In a multiuser situation user A could change the record and write it. User B may have the old record in his buffer and will never know about the changes until the record is re-read.

Example

```
Sub BuyerDataLoad ()
  Dim b_low As Double
  Dim b_high As Double

  CursorWait
  EnableBuyerData
  Call vxDouble("b_low", b_low)
  Call vxDouble("b_high", b_high)
  BuyLow.text = Format$(b_low, "#######0")
  BuyHigh.text = Format$(b_high, "#######0")
  BuyType.text = vxField("b_cat")
  BuyTypeDesc.text = vxField("b_desc")
  BuyCode.text = vxField("b_code")
  CursorArrow
End Sub
```

See Also

vxDouble, vxFieldTrim, vxInteger, vxLong, vxRecord, vxReplRecord, vxReplString, vxSetAlias, vxSetString

vxFieldCount

Declaration

```
Declare Function vxFieldCount Lib "vxbase.dll" () As Integer
```

Purpose

Extract the number of fields in the currently selected database.

Parameters

None.

Returns

An integer with the number of fields in the current database. If no database is selected, 0 is returned.

Usage

Use in conjunction with other field statistical functions to create listboxes of file structures, etc.

Example

```
' demonstration of file structure extraction
' ------------------------------------------
AircustDbf = vxUseDbf("\vb\vxbtest\aircust.dbf")
FileName.text = vxDbfName()
For j% = 1 To vxFieldCount()
   FieldName$ = vxFieldName(j%)
   FSize% = vxFieldSize(FieldName$)
   FType$ = vxFieldType(FieldName$)
   FDec% = vxDecimals(FieldName$)
   List1.AddItem FieldName$ + "     " + FType$ + "  " +
                 LTrim$(Str$(FSize%)) + "." +
                 LTrim$(Str$(FDec%))
Next
j% = vxClose

' note: the AddItem Method would be on one line
'       in the actual source code
' ------------------------------------------
```

See Also

vxDecimals, vxFieldName, vxFieldSize, vxFieldType

vxFieldName

Declaration

Declare Function vxFieldName Lib "vxbase.dll" (ByVal FieldNumber As Integer) As String

Purpose

Extract the name of the nth field in the field array of the current database.

Parameters

FieldNumber is an index into the field array that ranges from 1 to vxFieldCount.

Returns

A Visual Basic string that contains the name of the nth field.

Usage

Use in conjunction with other field statistical functions to create listboxes of file structures, etc.

Example

```
' demonstration of file structure extraction
' ------------------------------------------
AircustDbf = vxUseDbf("\vb\vxbtest\aircust.dbf")
FileName.text = vxDbfName()
For j% = 1 To vxFieldCount()
   FieldName$ = vxFieldName(j%)
   FSize% = vxFieldSize(FieldName$)
   FType$ = vxFieldType(FieldName$)
   FDec% = vxDecimals(FieldName$)
   List1.AddItem FieldName$ + "     " + FType$ + "  " +
                 LTrim$(Str$(FSize%)) + "." +
                 LTrim$(Str$(FDec%))
Next
j% = vxClose

' note: the AddItem Method would be on one line
'       in the actual source code
' ------------------------------------------
```

See Also

vxDecimals, vxFieldCount, vxFieldSize, vxFieldType

vxFieldSize

Declaration

Declare Function vxFieldSize Lib "vxbase.dll" (ByVal FieldName As String) As Integer

Purpose

Extract the size of the named field.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

An integer containing the field width.

Usage

Use in conjunction with other field statistical functions to create listboxes of file structures, etc.

Example

```
' demonstration of file structure extraction
' ------------------------------------------
AircustDbf = vxUseDbf("\vb\vxbtest\aircust.dbf")
FileName.text = vxDbfName()
For j% = 1 To vxFieldCount()
   FieldName$ = vxFieldName(j%)
   FSize% = vxFieldSize(FieldName$)
   FType$ = vxFieldType(FieldName$)
   FDec% = vxDecimals(FieldName$)
   List1.AddItem FieldName$ + "     " + FType$ + "  " +
                 LTrim$(Str$(FSize%)) + "." +
                 LTrim$(Str$(FDec%))
Next
j% = vxClose

' note: the AddItem Method would be on one line
'       in the actual source code
' ------------------------------------------
```

See Also

vxDecimals, vxFieldCount, vxFieldName, vxFieldType, vxSetAlias

vxFieldTrim

Declaration

Declare Function vxFieldTrim lib "vxbase.dll" (ByVal FieldName As String) As String

Purpose

Extract an xBase field, trim trailing spaces, and convert it to a Visual Basic string.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

A Visual Basic String (or ASCIIZ string if vxSetString is 1) that contains the contents of the defined field.

Usage

Get the contents of a character type field and trim trailing spaces. Trailing spaces should always be trimmed if the data is going into a text box for editing. If you use this function, it is not necessary to use the Visual Basic RTrim$ function to trim the string before placing it into a text box.

Note that all xBase data is stored in character format, so you can use this function to extract any field - including numeric, date, and logical fields (and even a memo block reference if you wish). You could then use Visual Basic data conversion functions to create the type of data you are interested in.

NOTE: The maximum length of a field that can be extracted with vxFieldTrim is 255. If a field is larger than this, use vxRecord to extract the entire record contents into a defined record type or string.

Example

```
BuyTypeDesc.Text = vxFieldTrim("b_desc")
```

See Also

vxField, vxInteger, vxLong, vxRecord, vxReplRecord, vxReplString, vxSetAlias, vxSetString

vxFieldType

Declaration

Declare Function vxFieldType Lib "vxbase.dll" (ByVal FieldName As String) As String

Purpose

Extract the type of the defined field from the current database.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

A Visual Basic string that contains the type code of the field. It will be one of "C" for character, "N" for numeric, "D" for date, "L" for logical, or "M" for memo.

Usage

Use in conjunction with other field statistical functions to create listboxes of file structures, etc.

Example

```
' demonstration of file structure extraction
' ----------------------------------------
AircustDbf = vxUseDbf("\vb\vxbtest\aircust.dbf")
FileName.text = vxDbfName()
For j% = 1 To vxFieldCount()
   FieldName$ = vxFieldName(j%)
   FSize% = vxFieldSize(FieldName$)
   FType$ = vxFieldType(FieldName$)
   FDec% = vxDecimals(FieldName$)
   List1.AddItem FieldName$ + "     " + FType$ + "  " +
                 LTrim$(Str$(FSize%)) + "." +
                 LTrim$(Str$(FDec%))
Next
j% = vxClose

' note: the AddItem Method would be on one line
'       in the actual source code
' ----------------------------------------
```

See Also

vxDecimals, vxFieldCount, vxFieldName, vxFieldSize, vxSetAlias

vxFile

Declaration

Declare Function vxFile Lib "vxbase.dll" (ByVal FileName As String) As String

Purpose

Determine if the named file exists.

Parameters

FileName is a literal string or string variable that contains a complete file name including an optional path.

Returns

TRUE if the file exists and FALSE if it does not.

Usage

Especially used in batch processing applications to determine whether or not a batch of transactions still exists. If the batch exists, in all likelihood it has not been processed yet and therefore a user request to create another batch file would be denied.

Example

```
' create transaction batch file with the same
' structure as the master file
' -----------------------------------------
BatchName$ = "Tr" + SignOnId$
FileSpec$ = MyPath$ + BatchName$ + ".dbf"
IndexSpec$ = MyPath$ + BatchName$ + ".ntx"

' if file exists, error
' --------------------
If vxFile(FileSpec$) Then
   MsgBox "Error. Batch file exists!"
   Exit Sub
Else
   ' if no error, create empty transaction file
   ' ---------------------------------------
   TrMasterDbf% = vxUseDbf("Transmas.dbf")
   TrMasterNtx% = vxUseNtx("Transmas.ntx")
   j% = vxSelectDbf(TrMasterDbf%)
   If Not vxCopyStruc(BatchName$) Then
      MsgBox "Error in batch file creation"
      j% = vxClose()
      Exit Sub
   Else
      ' now create index same as master file
      ' ------------------------------------
      IndexExpr$ = vxNtxExpr(TrMasterNtx%)
      If Not vxCreateNtx(BatchName$, IndexExpr$) Then
         MsgBox "Error in index creation"
         Kill FileSpec$
         j% = vxClose()
         Exit Sub
      End If
   End If
End If
j% = vxClose()         ' close master file
TransDbf% = vxUseDbf(BatchName$)
TransNtx% = vxUseNtx(BatchName$)
```

```
' call transactions editing procedure
' ---------------------------------
CollectTrans
```

```
' if posting now, append transactions to
' master file after they have been posted
' and then clear the batch file in preparation
' for the next editing session
' ------------------------------------------
j% = MsgBox("Post Now?", 52)
If j% = 6 Then
   PostTrans
   TrMasterDbf% = vxUseDbf("Transmas.dbf")
   TrMasterNtx% = vxUseNtx("Transmas.ntx")
   j% = vxSelectDbf(TrMasterDbf%)
   vxAppendFrom(BatchName$)
   j% = vxClose()      ' close master file
   Kill FileSpec$      ' erase batch file
   Kill IndexSpec$     ' and index
   Exit Sub
End If
j% = vxClose()              ' close the batch
```

See Also

vxAppendFrom, vxCopyStruc

vxFilter

Declaration

Declare Sub vxFilter Lib "vxbase.dll" (ByVal FilterString As String)

Purpose

Define a filter expression for use in masking unwanted records from displays, reports, etc.

Parameters

FilterString is a valid xBase expression that describes the records you wish to retain in the current procedure. It may be a literal string enclosed in quotes or a string variable.

Returns

Nothing. A pointer to the filter string is set up in the xBase descriptor block.

Usage

Declare filters to limit the range of records that will be displayed or printed. The most common filter is ".NOT. deleted()". A filter expression must evaluate to a logical result. Any declared filter affects the vxTop, vxBottom, vxSkip, vxSeek, and vxSum functions. vxGo ignores set filters.

vxBrowse automatically filters out deleted records. The filter set by vxFilter is in effect when a vxBrowse table is opened. If the user has access to the Filter menu item on the vxBrowse table, he can change the filter or remove it at will. The change or removal only effects the current browse and when vxBase returns to your Visual Basic program, the old filter is once again in effect.

Use filters judiciously. A filter set on a large file can slow processing enormously. For example, if a filter was set on a large names database to only show the name "BROWN", when the record pointer moved past the last "BROWN" (either through program control with vxSkip or with a down arrow by the user in a vxBrowse display), every record in the file would have to be evaluated until the end was reached before vxBase could determine there were no more "BROWN"s. If a filter is set on a large file, vxBrowse tables called on that file will take some time to initialize. vxBrowse must ascertain the number of records in the file that pass the filter to properly set the vertical scroll bar parameters. Study and use the SCOPE parameter available in vxTableDeclare instead.

NOTE: The record buffer must be filled with a valid record from the database BEFORE vxFilter is called. The database must be open and selected.

Complex Filter Expressions

A complex expression is one which contains two or more elements combined with a logical operator. For example, vxFilter("LastName = 'Smith' .and. AmtOwing > 100.00") is a complex expression which would result in only those records that satisfy both criteria being selected

for the operation. One must take care to recognize the precedence of logical operators. Use parentheses to group the elements of a complex expression if you are not sure of the potential result.

For example, the filter vxFilter(".NOT. deleted() .and. 'Tenholder' $ LastName") would appear to give us all records that contain "Tenholder" in the field LastName that are not deleted. In fact, the expression is evaluated as ".NOT. (deleted() .and. 'Tenholder' $ LastName)". The expression following the .not. will ALWAYS return false unless the record is both deleted and the last name contains "Tenholder" (which is not a record we want anyway). .NOT. FALSE is always TRUE; therefore, every record that is not deleted will be returned. The proper command would be vxFilter("(.NOT. deleted()) .and. ('Tenholder' $ LastName)").

Building Filter Expressions in String Variables

If the user is supplying one of the elements of a filter expression through a form text box, you may build a filter expression by concatenating the various elements into a string variable and then passing that variable to vxFilter. For example, suppose you solicit a city name from the user via a form textbox control and then wish to apply a filter to the database that contains only records with that city in Field "CityFld". The filter expression you want to pass to vxFilter may be vxFilter("'NEW YORK' $ CityFld"). The user enters the city name in control "TB_Detail".

```
Sub Btn_Find_Click
   j% = vxSelectDbf(PFind)
  j% = vxSelectNtx(DetIndex)
  FString$ = "'" + (TB_Detail.Text) + "' $ CityFld"
  Call vxFilter(FString$)
  j% = vxTop()
  ...
End Sub
```

Example

```
Dim CalifTotal As Double

' this routine adds up the amounts owing by customers
' in California
' ------------------------------------------------
Call vxFilter("(.NOT. deleted()) .AND. (state = 'CA')")
CalifTotal = 0
j% = vxTop()
Call vxSum("amtowing", CalifTotal)
TotalBox.text = Format$(CalifTotal, "#######0.00")
vxFilterReset
```

See Also

vxBrowse, vxEval, vxFilterReset

vxFilterReset

Declaration

```
Declare Sub vxFilterReset Lib "vxbase.dll" ()
```

Purpose

Removes a filter that was set with vxFilter and releases the memory allocated to hold the expression.

Parameters

None.

Returns

Nothing.

Usage

Always used to cancel a filter that was set to perform some specific procedure. Closing a file will also cancel the filter and release memory used to hold the filter expression.

Example

```
Dim CalifTotal As Double

' this routine adds up the amounts owing by customers
' in California
' ------------------------------------------------
Call vxFilter("(.NOT. deleted()) .AND. (state = 'CA')")
CalifTotal = 0
j% = vxTop()
Call vxSum("amtowing", CalifTotal)
TotalBox.text = Format$(CalifTotal, "#######0.00")
vxFilterReset
```

See Also

vxBrowse, vxFilter

vxFormFrame

Declaration

Declare Sub vxFormFrame Lib "vxbase.dll" (Hwnd As Integer)

Purpose

Draw a three dimensional frame inside the bounds of a form.

Parameters

Hwnd is the hWnd property of an active Visual Basic form.

Returns

Nothing.

Usage

Use in conjunction with vxCtlStyle to produce metallic, three-dimensional forms. The frame is drawn in gray scales that complement the look of control boxes enhanced with vxCtlStyle. Applicable to VGA and SVGA monitors only.

Always use the Visual Basic Refresh method in the Form_Resize event procedure to eliminate the old frame before a new one is drawn if the user has the ability to resize the form.

Example

```
Sub Form_Paint ()
  Call vxFormFrame(VXFORM2.hWnd)
  Call vxCtlStyle(TypeCode, VX_RECESS)
  Call vxCtlStyle(TypeDesc, VX_RECESS)
  Call vxCtlStyle(TypeStatus, VX_RAISE)

  ' if delete request from browse, do it now
  ' because we must let enhanced controls
  ' paint before asking for delete confirmation
  ' ----------------------------------------
  If TypeReturn = BROWSE_DELETE Then
     TypeDelete_Click
  End If
End Sub

Sub Form_Resize ()
  VXFORM2.Refresh
End Sub
```

See Also

vxCtlGrayReset, vxCtlGraySet, vxCtlStyle

vxFound

Declaration

```
Declare Function vxFound Lib "vxbase.dll" () As Integer
```

Purpose

Test the status of the last vxSeek or vxSeekSoft on the selected database.

Parameters

None.

Returns

TRUE if the last seek on the file resulted in a find, and false if not.

Usage

Even though vxSeek and vxSeekSoft immediately return the result of the operation, there are times when you want to know what the result of the last seek was well after the fact of the seek. Instead of saving the seek result in a variable, you can interrogate the status with vxFound. vxFound acts as a sort of global variable that retains the status of the last seek. It can even be interrogated from a module other than the one that issued the seek,

If the file is closed and then reopened, the status of the last seek is of course lost.

Example

```
j% = vxSeek("ABCDEF")
Call ChangeStatus
If vxFound() Then
   UpdateProc
Else
   AddProc
End If
```

See Also

vxSeek, vxSeekSoft

vxGetVersion

Declaration

```
Declare Function vxGetVersion lib "vxbase.dll" () As String
```

Purpose

Get a string containing the vxBase version number.

Parameters

None.

Returns

A Visual Basic String (or ASCIIZ string if vxSetString is 1) that contains the current vxBase version number.

Usage

It would be nice if you had a text box on your ABOUT form that displayed this number to aid vxBase tech support.

Example

```
VerBox.text = "vxBase Version " + vxGetVersion()
```

vxGo

Declaration

Declare Function vxGo Lib "vxbase.dll" (ByVal RecNum As Long) As Integer

Purpose

Position the record pointer to the defined record and read the record into the work buffer.

Parameters

RecNum is the physical record number to go to in the currently selected database.

Returns

TRUE if the operation was successful, or FALSE if not. FALSE will be returned if the record number is invalid, or if the record was locked by another user and the current user answered "NO" to the retry query. If FALSE is returned, the status of the record pointer and the data buffer are undefined.

Usage

This command is especially important in a multiuser environment. The current record number is usually saved prior to collecting edit data from a record and then the record is unlocked to allow other users to access it. After the edit operation, the record pointer is repositioned to the saved record number and the record is updated.

vxGo will find deleted records and records that don't satisfy a filter condition. In other words, if the record number is valid, it becomes the current record.

Multiuser Considerations

The record gone to is locked if vxSetLocks is TRUE (the default).

Example

```
' multiuser update example
' ------------------------
If vxSeek("ABC") Then         ' find the record to update
   RecNum& = vxRecNo()        ' save the record number
   Sig% = vxInteger("CustSig")  ' and the signature
   Name.text = vxField("Name") ' store the form vars
   Status.text = vxfield("Stat")

   ' now unlock the record
   ' ---------------------
   j% = vxUnlock()

   ' now perform the update on the vis basic form
   ' --------------------------------------------
   CustRecordUpdate
```

```
    ' now retrieve the record and test if anyone else
    ' has changed it
    ' ---------------------------------------------
    j% = vxGo(RecNum&)
    If Sig% <> vxInteger("CustSig") Then
       MsgBox "Another user beat you to it. Redo!"
    Else
       Call vxReplString("Name", (Name.text))
       Call vxReplString("Stat", (Status.text))
       Call vxReplInteger("CustSig", (Sig% + 1))
    End If
    j% = vxUnlock()
 End If
```

See Also

vxRecNo, vxSeek, vxSeekSoft, vxSetLocks, vxSkip

vxInit

Declaration

```
Declare Sub vxInit Lib "vxbase.dll" ()
```

Purpose

Register the current task with the vxBase DLL. If this is the first vxBase task, it controls the database shareable memory. If more than one vxBase task is running at the same time, only the first task allocates memory for use by all other tasks. It is necessary to register each vxBase task to ensure that the controlling task is not unloaded before any other vxBase task.

Parameters

None.

Returns

Nothing.

Usage

This procedure *must* be called before any other vxBase stetment is made (usually in the Form_Load procedure of your first form). It is used in conjunction with vxDeallocate to ensure that multitasking memory management flows smoothly. See the Multitasking and Multiuser Considerations section for more information.

Example

```
Sub Form_Load()
  vxInit
  vxCtlGraySet
End Sub
```

See Also

vxDeallocate

vxInteger

Declaration

Declare Function vxInteger Lib "vxbase.dll" (ByVal FieldName As String) As Integer

Purpose

Extract the defined field and convert the contents to an integer.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

An integer representing the contents of the field.

Usage

This function obviously works on numeric fields. If the field contains decimals, they are truncated. If the value of the field is greater than the integer maximum, the result is anybody's guess. This function also works on character fields that contain numbers.

Integer range is -32,768 to 32,767.

Example

```
j% = vxGo(RecNum&)
If Sig% <> vxInteger("CustSig") Then
   MsgBox "Another user beat you to it. Redo!"
Else
   Call vxReplString("Name", (Name.text))
   Call vxReplString("Stat", (Status.text))
   Call vxReplInteger("CustSig", (Sig% + 1))
End If
```

See Also

vxField, vxLong, vxReplInteger, vxSetAlias

vxIsMemo

Declaration

Declare Function vxIsMemo Lib "vxbase.dll" (ByVal FieldName As String) As Integer

Purpose

Determine whether there is a text memo attached to the defined field.

Parameters

FieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

TRUE if there is a memo in the .dbt file, and FALSE if not. FALSE will be returned if the memo block holds a bitmap instead of text.

Usage

Could be used to determine whether or not to display a memo for editing.

Example

```
If vxIsMemo("a_memo") Then
   SaveRec& = vxRecNo()
   Call vxMemoEdit(VXFORM2.hWnd, "a_memo")
   vxGo(SaveRec&)
End If
```

See Also

vxIsPicture, vxMemoEdit, vxMemoRead, vxSetAlias

vxIsPicture

Declaration

Declare Function vxIsPicture Lib "vxbase.dll" (ByVal MemoFieldName As String) As Integer

Purpose

Determine whether a bitmap is attached to the defined memo field.

Parameters

MemoFieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *MemoFieldName* may be qualified with a valid alias name that points to any open database.

Returns

TRUE if a bitmap is present and FALSE if not. Note that text attached to the field instead of a bitmap will return FALSE.

Usage

Could be used to determine whether or not to load a new bitmap.

Example

```
' the name of the bmp picture file is the
' same as the string in field "title" so
' we can import the bmps into the memo file
' by cocatenating ".bmp" to the trimmed field
' contents
' ----------------------------------------
j% = vxTop()
If Not vxIsPicture("pic") Then
   For i& = 1 To 13 ' there are 13 recs in the file
      j% = vxGo(i&)
      ftitle$ = vxFieldTrim("type")
      fname$ = "c:\magic\bmp\" + ftitle$ + ".bmp"
      If Not vxPictureImport(fname$, "pic") Then
         MsgBox "Import Failed"
      End If
   Next i&
   j% = vxClose() ' close ensures buffers flushed
   AirPicsDbf = vxUseDbf("\vb\vxbtest\airpics.dbf")
   j% = vxSelectDbf(AirPicsDbf)
End If
```

See Also

vxIsMemo, vxMemoClear, vxPictureImport, vxPicturePrint, vxPictureRead, vxSetAlias

vxIsRecLocked

Declaration

Declare Function vxIsRecLocked Lib "vxbase.dll" () As Integer

Purpose

Determine if the current record is locked or not.

Parameters

None.

Returns

TRUE if the record is locked and FALSE if not locked.

Usage

Test if a record is locked or not before attempting an update.

Example

```
' wait until rec is free before update
' -----------------------------------
Do While TRUE
   If Not vxIsRecLocked() Then
      j% = vxLockRecord
      vxReplString("field1", (Text1.text))
      j% = vxWrite()
      j% = vxWriteHdr()
      j% = vxUnlock
      Exit Do
   End If
Loop
```

See Also

vxLockDbf, vxLocked, vxLockRecord, vxSetLocks, vxUnLock

vxIsSubNtx

Declaration

Declare Function vxIsSubIndex Lib "vxbase.dll" (ByVal NtxArea As Integer) As Integer

Purpose

Determine whether or not the defined index is a subindex or a normal index.

Parameters

NtxArea is the select area of an index file returned by vxUseNtx or vxAreaNtx.

Returns

TRUE if the index is a subindex or FALSE if it is not.

Usage

It may be necessary to determine an update strategy or perhaps do something based upon the record count depending on whether or not the entire file is represented in the index.

Example

```
If vxIsSubNtx(vxNtxCurrent()) Then
   NumRecs& = vxNumRecsSub()
Else
   NumRecs& = vxNumRecs()
End If
```

See Also

vxCreateSubNtx, vxNtxCurrent, vxNtxSubExpr, vxNumRecsSub, vxUseNtx

vxJoin

Declaration

Declare Sub vxJoin Lib "vxbase.dll" (ByVal DbfArea As Integer, ByVal NtxArea As Integer, ByVal JoinExpr As String, ByVal KeyType As Integer, ByVal JoinTitle As String)

Purpose

Define a visual join window. This is truly one of the most exciting features of vxBase. You can set up chains of visual relationships that are activated through a vxBrowse window. In the sample application, the LINK menu items give you a taste of the possibilities.

xBase programmers will recognize this function as a variation on the SET RELATION TO command. We aren't limited to many to one relationships, however. We can go from one to many to many to one ad infinitum (or at least as far as our system will allow in terms of open files).

Parameters

DbfArea is the select area of an open database that will be joined to the currently selected database when its vxBrowse is activated.

NtxArea is the index to use with *DbfArea*. It also must be open. The file being joined to must be indexed, and an index expression must be able to be formed out of the field elements of the current database. We are in fact setting up a relationship between the current database and the database we are defining with this function.

JoinExpr is a valid xBase expression that defines the field or expression (both of which must contain field elements from the current database) that we will use to institute the join. Alias field qualifiers are NOT allowed in *JoinExpr*.

KeyType is one of the Global constants VX_FIELD or VX_EXPR that are defined in vxbase.txt. If the JoinExpr is simply a field, we use VX_FIELD; if an expression, we use VX_EXPR. We define this value to speed up the linking operation. If the join item is only a field, much less processing occurs when we institute the join.

JoinTitle is the caption of the joined window.

Returns

Nothing. We are attaching the join definition to the current database descriptor block and it will only take effect when we vxBrowse the current file.

Usage

Suppose we have a customer file that we will use as the parent browse window to our joins. We will define a table to limit the fields displayed in the window and then set up a join to a subledger file. The subledger file contains many records, each of which contains a customer code and invoice number as the key. There could be many records for each customer. We open the subledger file and also define a table to limit its browse. This browse will be activated when the user selects JOIN

from the vxBrowse menu bar attached to the customer browse table.

When we define the join for the customer file, we use the customer code field as key into the subledger file. This is the common element. When the join is activated by the user, a window opens that contains nothing but the subledger records belonging to the customer who is currently highlighted in the parent window. If we move the pointer in the parent window to another record, then his subledger records magically appear in the join window.

We could go on with more joins. For example, while we were defining the table for the subledger, we could have set up another join to an invoice file that contains the details of each invoice contained in the subledger summary. Now, the user could pick invoices (which would be the key from the subledger to the invoice file) from the second window and watch their details appear in a third window.

The invoice details might contain a reference to an inventory code number. There is nothing stopping us from defining another join to the inventory file from the invoices file. Lots of possibilities, right?

When setting up a join sequence, it makes logical sense to start with the lowest file in the join totem. It won't have a join to another file. Open it, declare a table, and proceed to the next lowest file in the hierarchy. If you are only joining two files, you can set up as in the example below.

Note that if onscreen editing is enabled in the parent window, it only applies to items on the parent window. You cannot perform onscreen editing on joined windows.

Example

```
Sub LinkBuyToSell_Click ()
  ' Demonstration of setting up visual relationships
  ' with the vxJoin command. What we have is a file of buyers
  ' categorized by type of aircraft they are interested in.
  ' What we are going to do is display a browse table of
  ' these buyer records and link any buyer record to
  ' another browse table of aircraft that match the the
  ' buyer aircraft type field.

  ' Conversely, the LinkSellToBuy proc does the opposite.
  ' It links the aircraft with all prospective buyers.
  ' ------------------------------------------------------

  ' open file that will control the join
  ' ------------------------------------
  AirbuyerDbf = vxUseDbf("\vb\vxbtest\airbuyer.dbf")
  Airbuy2Ntx = vxUseNtx("\vb\vxbtest\airbuy2.ntx")
  ' this index is on aircraft type
  ' ------------------------------

  ' define table to show data we are interested in
  ' ----------------------------------------------
  Call vxTableDeclare(VX_BLUE, ByVal 0&, ByVal 0&, 0, 1, 5)
  Call vxTableField(1, "Type", "b_cat", VX_FIELD)
  Call vxTableField(2, "Description", "left(b_desc,20)",
                    VX_EXPR)
  Call vxTableField(3, "Low", "b_low", VX_FIELD)
```

```
Call vxTableField(4, "High", "b_high", VX_FIELD)
Call vxTableField(5, "Customer", "b_code", VX_FIELD)
```

```
  ' now open secondary file and define its table
  ' -------------------------------------------
  AircraftDbf = vxUseDbf("\vb\vxbtest\aircraft.dbf")
  If AircraftDbf = FALSE Then
     MsgBox "Error Opening aircraft.dbf. Aborting."
     j% = vxSelectDbf(AirbuyerDbf)
     j% = vxClose()
     Exit Sub
  End If
  Aircraf2Ntx = vxUseNtx("\vb\vxbtest\aircraf2.ntx")

  Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 5)
  Call vxTableField(1, "Type", "c_cat", VX_FIELD)
  Call vxTableField(2, "Code", "c_code", VX_FIELD)
  Call vxTableField(3, "Price", "c_price", VX_FIELD)
  Call vxTableField(4, "Year", "c_year", VX_FIELD)
  Call vxTableField(5, "TTSN", "c_ttsn", VX_FIELD)

  ' reselect the master file and set up the join
  ' -------------------------------------------
  j% = vxSelectDbf(AirbuyerDbf)
  Call vxJoin(AircraftDbf, Aircraf2Ntx, "b_cat", VX_FIELD,
              "Possible Sales")

  ' this joins the Aircraft file using the index selected
  ' for it to the buyer file. The "b_cat" param is the
  ' field we will use as a key into the aircraft file and
  ' the VX_FIELD item tells vxBase that it is a field and
  ' not an expression. The last item in the call is a
  '  title for the join window.
  ' -----------------------------------------------------

  ' now set up and execute the browse. The JOIN menu item
  ' is automatically enabled.
  ' -----------------------------------------------------
  Call vxBrowse(VXFORM1.hWnd, AirbuyerDbf, Airbuy2Ntx,
                FALSE, TRUE, FALSE, 0, "Buyer Details",
                BuyerReturn)

  ' when we return from the browse we can ignore anything
  ' vxBase sent back to us in the BuyerReturn param
  ' -----------------------------------------------------
  j% = vxClose()
  j% = vxSelectDbf(AircraftDbf)
  j% = vxClose()

  ' we could get fancy and get the customer record if the
  ' use hit enter and then display or edit it. Do
  ' whatever you like.
  ' -----------------------------------------------------
End Sub
```

See Also

vxBrowse, vxTableDeclare, vxTableField

vxJoinNoAuto

Declaration

Declare Sub vxJoinNoAuto Lib "vxbase.dll" ()

Purpose

Declared join windows are automatically displayed when a browse window is opened. This command inhibits the creation of the joined window and forces the user to select the Join menu item from the browse main menu to display joined records.

Parameters

None.

Returns

Nothing.

Usage

If the file you are joining to has little chance of getting a match when you start the browse, it is preferable to force the user to pick the join from the menu rather than immediately displaying a vxBase error message box informing him that there are no records to display.

Example

```
' Join is declared above...
' -------------------------
Call vxJoinNoAuto
Call vxBrowse(VXFORM1.hWnd, AirbuyerDbf, Airbuy2Ntx,
              FALSE, TRUE, FALSE, 0, "Buyer Details",
              BuyerReturn)
```

See Also

vxJoin, vxJoinReset

vxJoinReset

Declaration

```
Declare Sub vxJoinReset Lib "vxbase.dll" ()
```

Purpose

Remove a join definition from the current database descriptor block and recover the memory.

Parameters

None.

Returns

Nothing. Affects internal parameters only.

Usage

It is only necessary to use this command if you intend to retain the open status of the current file and perhaps issue another vxBrowse command at some other point in your program. vxClose and vxCloseAll automatically reset the join and recover allocated memory.

Example

```
If BuyerReturn = BROWSE_ADD Then
   vxJoinReset
   AddProcedure
End If
```

See Also

vxClose, vxCloseAll, vxJoin

vxLocate

Declaration

Declare Function vxLocate lib "vxbase.dll" (ByVal XBaseExpr As String, ByVal Direction As Integer) As Long

Purpose

Searches for a record from and including the current record position that satisfies a logical xBase expression.

Parameters

XBaseExpr is a literal string or variable containing a valid xBase expression. The expression must evaluate as .T. or .F..

Direction is an integer defined as a global constant in vxbase.txt. VX_FORWARD (Value 0) tells vxLocate to search in a forward direction. VX_BACKWARD (Value 1) skips backwards during the search.

Returns

A long integer that contains the record number of the found record OR zero (0) if the search expression was not satisfied.

If found, the record buffer contains the found record. If not found, the record pointer is repositioned to the record that was active before the search.

Usage

Useful for searching a database for strings or values that are not keyed. vxLocate initiates the search from and including the current record position. Use vxLocateAgain to restart the same search.

Example

```
' We have a FIND button on a customer display form.
' When the button is clicked, we set up to solicit
' search parameters from the user and then load
' a modal form to gather the user input.
' ------------------------------------------------
Sub FindButton_Click ()
  If RecChange = TRUE Then CustSave_Click

  VXFORM3.Enabled = FALSE
  CustReturn = 0
  SaveRec& = vxRecNo() ' save where we are

  j% = vxSelectDbf(vxClientDbf)
  j% = vxSelectNtx(vxCl1Ntx)  ' index on customer serial
  j% = vxTop() ' start search from top

  ' display locate form as modal
  VXFORM5.Show 1

  ' VXFORM5 will fill in a value in our standard
  ' Browse return var CustReturn
  VXFORM3.Enabled = TRUE

  ' if rec wasn't found or user chose not
  ' to select a found rec then CustReturn will be zero
  ' ------------------------------------------------
  If CustReturn = 0 Then
```

```
     j% = vxGo(SaveRec&)
     Exit Sub
  Else
     CustReturn = BROWSE_EDIT
     CustDataLoad
     VXFORM1.StatBar.text = "Edit record " + LTrim$(Str$(vxRecNo()))
     VXFORM3.CustCompany.SetFocus
  End If
End Sub

' ------------------------------------------------------------
' VXFORM5 is a modal form that solicits locate information
' from the user. It puts a value into global var CustReturn
' that we can interrogate when we return from vxform5.
'
'    VXFORM5 has the following elements:
'    1. a group box with radio buttons for each field
'       that we want the user to be able to search
'       a. Option1 designates company name
'       b. Option2 designates the client name
'
'    2. a text box (SearchBox) that accepts a string
'       that we wish to locate
'
'    3. five buttons:
'       a. ButtonStart  initiates the search
'       b. ButtonAgain  looks for another FORWARD
'       c. ButtonBack   looks for another BACKWARD
'       d. ButtonOK     accepts the results and returns
'       e. ButtonCancel cancels the operation and returns
'
'    4. two text boxes in which we will display the
'       found data (CompanyBox and NameBox)
' ------------------------------------------------------------
Dim SearchStr As String  ' global var to hold search string

' events below are in logical sequence
' (starting with Form_Load)
' ------------------------------------
Sub Form_Load ()
  j% = vxSelectDbf(vxClientDbf)
  Option1.Value = 1 ' set default to company

  ' disable buttons that shouldn't work
  ' until something has been found
  ButtonAgain.Enabled = FALSE
  ButtonBack.Enabled = FALSE
  ButtonOK.Enabled = FALSE
End Sub

Sub Form_Paint ()
  ' register the database
  j% = vxSelectDbf(vxClientDbf)

  ' make the form pretty
  Call vxFormFrame(VXFORM5.hWnd)
  Call vxCtlStyle(Frame1, VX_RAISE)
  Call vxCtlStyle(SearchBox, VX_RECESS)
  Call vxCtlStyle(CompanyBox, VX_RECESS)
  Call vxCtlStyle(NameBox, VX_RECESS)
End Sub

Sub SearchBox_GotFocus ()
  ' user input convert to uppercase
  j% = vxCtlFormat(40, VX_UPPER, 0)
End Sub
```

```
' when user clicks the button to initiate the search...
' ----------------------------------------------------
Sub ButtonStart_Click ()
  ' ensure we have something to search for
  If SearchBox.Text = "" Then
     MsgBox "Search string required"
     Exit Sub
  End If

  ' ensure a button has been selected
  OpTotal% = Option1.Value + Option2.Value
  If Not OpTotal% Then
     MsgBox "Select a field"
     Exit Sub
  End If

  ' put single quotes around search string
  SearchStr = "'" + RTrim$((SearchBox.Text)) + "'"

  ' construct xbase expression
  If Option1.Value Then SearchStr = SearchStr + " $ upper(vxcompany)"
  If Option2.Value Then SearchStr = SearchStr + " $ upper(vxname)"
  j% = vxTop() ' always start search at top

  ' perform the search
  CustReturn = vxLocate(SearchStr, VX_FORWARD)

  ' vxLocate return will be zero if nothing found
  ' ---------------------------------------------
  If CustReturn = 0 Then
     MsgBox "Search string not found"
     CompanyBox.Text = ""
     NameBox.Text = ""
     ButtonAgain.Enabled = FALSE
     ButtonBack.Enabled = FALSE
     ButtonOK.Enabled = FALSE
  Else
     ' if found show results in text box
     CompanyBox.Text = vxField("vxcompany")
     NameBox.Text = vxField("vxname")

     ' and enable buttons
     ButtonAgain.Enabled = TRUE
     ButtonBack.Enabled = TRUE
     ButtonOK.Enabled = TRUE
  End If
End Sub
' user can look for next occurence (FORWARD) by
' pressing ButtonAgain
' ---------------------------------------------
Sub ButtonAgain_Click ()
  CustReturn = vxLocateAgain(VX_FORWARD)
  If CustReturn = 0 Then
     MsgBox "Search string not found"
     CompanyBox.Text = ""
     NameBox.Text = ""
     ButtonAgain.Enabled = FALSE
     ButtonBack.Enabled = FALSE
     ButtonOK.Enabled = FALSE
  Else
     CompanyBox.Text = vxField("vxcompany")
     NameBox.Text = vxField("vxname")
     ButtonAgain.Enabled = TRUE
     ButtonBack.Enabled = TRUE
     ButtonOK.Enabled = TRUE
  End If
End Sub
```

```
' user can look for previous occurence (BACKWARD)
' by pressing ButtonBack
' ---------------------------------------------
Sub ButtonBack_Click ()
  CustReturn = vxLocateAgain(VX_BACKWARD)
  If CustReturn = 0 Then
     MsgBox "Search string not found"
     CompanyBox.Text = ""
     NameBox.Text = ""
     ButtonAgain.Enabled = FALSE
     ButtonBack.Enabled = FALSE
     ButtonOK.Enabled = FALSE
  Else
     CompanyBox.Text = vxField("vxcompany")
     NameBox.Text = vxField("vxname")
     ButtonAgain.Enabled = TRUE
     ButtonBack.Enabled = TRUE
     ButtonOK.Enabled = TRUE
  End If
End Sub

' user cancels search by pressing ButtonCancel
' --------------------------------------------
Sub ButtonCancel_Click ()
  CustReturn = 0
  Unload VXFORM5
End Sub

' user accepts result of search and sends
' new record number back to caller
' by pressing ButtonOK
' ------------------------------------
Sub ButtonOK_Click ()
  Unload VXFORM5
End Sub

' deregister window and ctlformat
' when unloading form
' -------------------------------
Sub Form_Unload (Cancel As Integer)
  vxWindowDereg (VXFORM5.hWnd)
End Sub
```

See Also

vxLocateAgain

vxLocateAgain

Declaration

```
Declare Function vxLocateAgain lib "vxbase.dll" (ByVal Direction As
Integer) As Long
```

Purpose

Searches for a record from and NOT including the current record position that satisfies a logical xBase expression previously defined with vxLocateAgain.

Parameters

Direction is an integer defined as a global constant in vxbase.txt. VX_FORWARD (Value 0) tells vxLocateAgain to search in a forward direction. VX_BACKWARD (Value 1) skips backwards during the search.

Returns

A long integer that contains the record number of the found record OR zero (0) if the search expression was not satisfied.

If found, the record buffer contains the found record. If not found, the record pointer is repositioned to the record that was active before the search.

Usage

Used to continue a search that was initiated by vxLocate.

Example

See example in vxLocate documentation.

See Also

vxLocate

vxLockDbf

Declaration

Declare Function vxLockDbf Lib "vxbase.dll" () As Integer

Purpose

Lock the currently selected database and all of its index files.

Parameters

None.

Returns

TRUE If the operation was successful and FALSE if not. The operation could return false if the file or any of its records is already locked and the end user chose to abort the operation. Always test the result before proceeding with the code that requires the exclusive use of the file.

Usage

vxBase functions and procedures that automatically require a locked file (such as vxPack, vxZap, etc.) are already locked. It is not necessary to lock before performing these functions. If you require exclusive use of a file for any reason (e.g., closing a general ledger at the end of the year), use vxLockDbf. To unlock it, either close the file or use vxUnlock.

Example

```
If vxLockDbf() Then
   CloseTheBooks
   j% = vxUnlock()
Else
   MsgBox "Aborting year end procedure"
   Exit Sub
End If
```

See Also

vxIsRecLocked, vxLocked, vxLockRecord, vxLockRetry, vxSetLocks, vxUnlock

vxLocked

Declaration

Declare Function vxLocked Lib "vxbase.dll" () As Integer

Purpose

Determine if the current file is locked or not.

Parameters

None.

Returns

TRUE if the file is locked and FALSE if not locked.

Usage

Test if a file is locked before executing a procedure which will require exclusive use.

Example

```
j% = vxSelectDbf(GlMaster)
If vxLocked() Then
   MsgBox "File is locked. Try again later."
   Exit Sub
Else
   If vxLockDbf() Then
        CloseBooks
      j% = vxUnlock()
   End If
End If
```

See Also

vxIsRecLocked, vxLockDbf, vxLockRecord, vxLockRetry, vxSetlocks, vxUnlock

vxLockRecord

Declaration

```
Declare Function vxLockRecord Lib "vxbase.dll" () As Integer
```

Purpose

Lock the current record.

Parameters

None.

Returns

TRUE if the lock was successful or FALSE if it was not.

Usage

This function could be used as a status check to ensure that the record is indeed locked by your workstation. It would not normally be required if vxSetLocks is TRUE (the default) because vxBase automatically locks records as soon as they are read. If vxSetLocks is FALSE, this function MUST be used before updating a record.

Example

```
j% = vxGo(SaveRec&)
DoABunchOfStuff
If vxLockRecord() Then
   UpdateProc
Else
   MsgBox "Sorry. Can't lock the record"
End If
```

See Also

vxIsRecLocked, vxLockDbf, vxLocked, vxLockRetry, vxSetLocks, vxUnlock

vxLockRetry

Declaration

Declare Sub vxLockRetry Lib "vxbase.dll" (ByVal ErrorMode As Integer, ByVal WaitSeconds As Integer)

Purpose

Set the method of reporting a locked file or record to the user and also specify a lock try timeout value.

Parameters

ErrorMode is passed as either TRUE or FALSE.

If TRUE (the default), and an operation is performed that requires a lock (either record or file) which is unable to be set because another user has control of the record or file, the user is presented with a message from vxBase that asks if he wishes to retry the operation or abort. If "Retry" is chosen, vxBase attempts to set the lock again.

If *ErrorMode* is passed as FALSE, vxBase issues error code 610 "File lock error" instead of the retry message IF vxSetErrorMethod is set to TRUE. If the alternate error method is TRUE, the programmer can then set up his own method of dealing with locks through the VB ON ERROR routine.

If vxSetErrorMethod is FALSE and *ErrorMode* is FALSE, the default user retry message is sent instead.

WaitSeconds is the number of seconds to continue to attempt setting a lock. If zero (0), and a lock fails, the lock function returns with the error immediately and either presents the "Retry" message to the user or triggers the vxBase 600 error depending on the value of *ErrorMode* and vxSetErrorMethod.

If *WaitSeconds* is greater than zero, vxBase will continue to attempt to set the lock until *WaitSeconds* has expired. The maximum number of *WaitSeconds* that can be specified is 32,767 (which equates to over 9 hours) and is essentially a "wait forever" state.

Returns

Nothing.

Usage

In a multiuser environment the programmer often wishes to handle the failed lock scenario himself rather than rely on the user to retry the lock or not. This is the function of the *errormode* parameter.

Setting *WaitSeconds* to about 20 is a good value for normal database operations. For example, if a record is to be written, vxBase will try over and over again for 20 seconds to set the lock. After the time has expired, the selected error method is executed.

If using vxBase in an unattended program, it makes good sense to set the timeout value very high. Other processes that take control of a file for 15 or 20 minutes then do not necessarily disrupt the unattended

program from performing its tasks after the file has been released.

NOTE: If a file is opened for exclusive use with vxUseDbfEX then ALL other processes requiring that file across the entire network will be denied access to the file. The vxUseDbfEX function sets a network SHARE flag rather than a Clipper style lock on the file and no one is granted access to the file no matter what the value of vxSetLocks.

NOTE: vxLockRetry settings are GLOBAL to all vxBase tasks on a workstation. Once you select a set of values, you should use the same values in all of your vxBase programs.

Example

```
' this function attempts to write a record and, if the required
' lock fails, it send the user its own message asking for a
' another attempt instead of using the vxBase default Retry? message
' ------------------------------------------------------------------
Sub RecWrite
   Dim vxError As vxErrorStruc
   Dim RetryVal As Integer

   vxSetLocks FALSE
   vxSetErrorMethod TRUE
   vxLockRetry 0, 20
   ' will use alternate error method after
   ' retrying a lock for 20 seconds

   On Error GoTo LockError

   ' we will attempt to lock the record
   ' as long as RetryVal is zero. If the
   ' lock required by vxWrite fails, we
   ' exit to the On Error routine - which
   ' will set RetryVal to 2 if the user
   ' does not wish to retry.
   ' ------------------------------------
   RetryVal = 0

   Do
      If vxLockRecord() Then RetryVal = 1
   Loop While RetryVal = 0

   If RetryVal = 1 Then
      If Not vxWrite() Then
         MsgBox "Record Write Error"
      End If
   End If

   vxSetErrorMethod FALSE
   Exit Sub
LockError:
   If vxErrorTest(vxError) Then
      If vxError.ErrorNum = 600 Then
         j% = MsgBox("Lock failed. Retry?", 52)
         If j% = 6 Then
            Resume Next
         Else
            RetryVal = 2
            Resume Next
         End If
      End If
   End If
End Sub
```

See Also

vxIsRecLocked, vxLockDbf, vxLocked, vxLockRecord, vxSetLocks, vxUnlock, vxUseDbfEX

vxLong

Declaration

Declare Function vxLong Lib "vxbase.dll" (ByVal FieldName As String) As Long

Purpose

Extract the defined field and convert the contents to a long integer.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

A long integer representing the contents of the field.

Usage

This function obviously works on numeric fields. If the field contains decimals, they are truncated. If the value of the field is greater than the long integer maximum, the result is anybody's guess. This function also works on character fields that contain numbers.

Long integer range is -2,147,483,648 to 2,147,483,647.

Example

```
j% = vxGo(RecNum&)
If OrigNum& <> vxLong("OrigRecNo") Then
   MsgBox "File has been packed"
   Call vxReplLong("OrigRecNo", vxRecNo())
End If
j% = vxUnlock()
```

See Also

vxDouble, vxField, vxInteger, vxReplLong, vxSetAlias

vxMemCompact

Declaration

Declare Function vxMemCompact Lib "vxbase.dll" () As Long

Purpose

Windows memory can become extremely fragmented when running a vxBase application. The vxBase program architecture is built of many small chunks which are all discardable and may be loaded on call. Other tasks running concurrently with vxBase can also contribute to memory fragmentation and consequent loss of perfromance or even out of memory conditions when a large request is made for some fixed memory (e.g., reading a memo). vxMemCompact uses Windows API routines to compact memory and leave the largest contiguous free areas possible.

Parameters

None.

Returns

The number of bytes in the largest free global memory object in the global heap.

Example

```
' compact memory on each return to the controlling form
' ------------------------------------------------------
Sub Form_Unload (Cancel As Integer)
   vxWindowDereg(VXFORM5.hWnd)
   k& = vxMemCompact()
End Sub
```

vxMemoClear

Declaration

Declare Function vxMemoClear Lib "vxbase.dll" (ByVal MemoFieldName As String) As Integer

Purpose

Remove a memo block reference from a memo field. The bitmap OR memo attached to the field is effectively deleted.

Parameters

MemoFieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. MemoFieldName may be qualified with a valid alias name that points to any open database.

Returns

TRUE if the operation was successful and FALSE if not. Always returns FALSE is always returned if the associated dbf has been opened as Read Only with vxUseDbfRO.

Usage

Delete a memo or bitmap.

Example

```
Sub ButtonDelete_Click ()
   If vxMemoClear("pic") Then
      PicBox.Picture = LoadPicture()
   Else
      MsgBox "Delete failed"
   End If
End Sub
```

See Also

vxIsMemo, vxIsPicture, vxReplMemo, vxPictureImport, vxSetAlias

vxMemoEdit

Declaration

Declare Sub vxMemoEdit Lib "vxbase.dll" (ByVal Hwnd As Integer, ByVal FieldName As String)

Purpose

Edit an existing memo or create a new memo referenced by the specified memo field.

Parameters

Hwnd is the hWnd property of an active Visual Basic form. This window acts as parent to the memo window. It must be enabled and should be big enough to accomodate a reasonable edit window (though you can of course resize the vxMemoEdit window to whatever your heart desires). The inital size and position (as well as the memo window caption) may also be set with vxMemoPos.

FieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. FieldName may be qualified with a valid alias name that points to any open database.

Returns

Nothing. The procedure creates a standard Windows text editing window and puts the memo text into it. You can also create a new memo from scratch, import standard ASCII text files into the memo window, export the memo to a text file, copy,cut, and/or paste from and to the clipboard. Everything you would expect (including print).

Usage

The activated memo window comes with its own menu bar. You have plenty of options.

File Save Memo: saves the current memo into the .dbt file. If the edited memo will not fit into the same space it formerly occupied, it is moved to the end of the .dbt file and rewritten there. The old space is not reclaimed. vxPack or vxCopy will compress memo files by only writing memo blocks that have active references in the .dbf file. Note that this menu option is disabled if the dbf file has been opened with vxUseDbfRO (Read Only). If vxSetAnsi(TRUE) (the default), the text is saved as ANSI characters; if FALSE, the text is converted to the OEM character set before saving.

File Import ASCII: you may import any ASCII text file available on your system into the memo at the current cursor postiion. A standard Windows file pick list is presented when you choose this option, including a full disk/directory list box. Note that this menu option is disabled if the dbf file has been opened with vxUseDbfRO (Read Only). If vxSetAnsi(TRUE) (the default), the file is read directly from disk into the memo buffer; if FALSE, the text is converted from OEM to ANSI before filling the memo buffer.

File Export ASCII: you may export the current memo to a standard

ASCII file. The file is written into the current directory. A standard Windows file pick list is displayed when you choose this option but it gives you no opportunity to change the directory. If vxSetAnsi(TRUE) (the default), the buffer is written to the file without translation; if FALSE, the buffer text is converted to the OEM character set before writing.

File Print: Prints the memo to the current Windows printer exactly as it is shown in the memo edit window.

Edit Functions: All standard Windows editing functions along with the standard accelerator keys are available. Items can be cut, copied, and pasted to and from the clipboard (which means you can import things into your memo from any application that can paste into the clipboard!). An Undo option is also available when it is possible to undo the last operation, as well as a Select All function and an Insert Date function, which inserts a date and time stamp directly into the memo at the current cursor position.

All in all this is a pretty snazzy memo editor. There are only a few rules you have to follow to successfully edit memos, and they are fully documented in the source code example below.

Memo File Intricacies

vxBase memos are compatible with those of Clipper and dBase III/III+. Maximum memo size is 64k. Clipper memos are always stored with soft carriage return/line feeds that fit the memo to the size of the text window it was edited in. vxBase strips these soft returns and linefeeds from a Clipper maintained memo and does not restore them. A vxBase memo always fits the size of the window it resides in with automatic wordwrap. Remember that a Windows window can be dynamically resized by the user so it would be foolhardy to attempt to maintain an artficial end of line within paragraphs.

If you edit a vxBase memo with a Clipper MEMOEDIT(), the soft returns will be restored by Clipper so there should be no compatibility problems in moving from one type of application to another using the same files.

Example

```
Sub CustMemo_Click ()
  ' Edit memo. Always have an ENABLED form showing to act
  ' as parent to the memo window. It also must have the
  ' focus. Copy the code below EXACTLY to ensure successful
  ' memoedits (changing the form and field names to fit
  ' your application of course)
  '-------------------------------------------------------
  RecNum& = vxRecNo()         ' save rec num to goto later
  VXFORM3.SetFocus            ' make sure form has focus
  Call vxMemoEdit(VXFORM3.hWnd, "a_memo")
  j% = vxGo(RecNum&)          ' reset rec buffer
  j% = vxUnlock()             ' unlock the record

  ' The vxUnlock() is only necessary if you are working in
  ' a multiuser environment. The saving of the record
  ' number and then going to same after the memoedit is
  ' ABSOLUTELY NECESSARY. After the memo edit completes,
```

```
  ' the contents of the record buffer are undefined if the
  ' user chose not to save the memo contents.
  ' ---------------------------------------------------
End Sub
```

See Also

vxAppendFrom, vxCopy, vxIsMemo, vxIsPicture, vxMemoClear, vxMemoPos, vxMemoRead, vxPictureImport, vxPictureRead, vxPack, vxSetAlias, vxSetAnsi

vxMemoPos

Declaration

Declare Sub vxMemoPos Lib "vxbase.dll" (ByVal StartX As Integer, ByVal StartY As Integer, ByVal xWidth As Integer, ByVal yHeight As nteger, ByVal MemoTitle As String)

Purpose

Set the start position and size of an upcoming memo edit window that will edit a memo attached to the current database with vxMemoEdit. Also used to set up the memo edit window title.

Parameters

Window coordinates passed to this function use familiar character units in the x dimension and line height units in the y dimension. The units are converted to the average character width and height of the standard Windows system font and are therefore device independent.

StartX is the start position of the memo window in characters from the left edge of the screen.

StartY is the start position of the memo window in lines from the top of the screen.

xWidth is the start width of the memo window in characters.

yHeight is the height of the memo window (including caption and menu bars) in lines.

Returns

Nothing.

Usage

Memo window position and size are defaulted according to the size of the parent window passed to the vxMemoEdit function if this command is not issued. If this Sub is called, the position and size are relative to the entire screen.

The default memo window caption is "Memo Edit: DBF name". If this is good enough, pass the MemoTitle as a null value (ByVal 0&).

If you wish to set the memo title only and leave the size and position as the default values, pass all x and y coordinates as 0. For example,

```
Call vxMemoPos(0,0,0,0,"My Memo Title")
```

If the user sizes the memo window according to his own tastes, its position and size will be retained on subsequent edits.

Example

```
Call vxMemoPos(10, 5, 60, 15, "Customer Complaints")
RecNum& = vxRecNo()
Call vxMemoEdit(VXFORM3.hWnd, "a_memo")
j% = vxGo(RecNum&)
j% = vxUnLock()
```

See Also

vxMemoEdit

vxMemoRead

Declaration

```
Declare Function vxMemoRead Lib "vxbase.dll" (Byval FieldName  As
String, ByVal LineWidth As Integer) As String
```

Purpose

Read a memo into a Visual Basic string.

Parameters

FieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

LineWidth is the width of a formatted line that vxBase will terminate with a carriage return-linefeed.

If *LineWidth* is zero (or less than 10), no formatting is performed. This would be your option if you were simply displaying the memo contents in a multiline text box. Visual Basic will automatically perform word wrap within the multiline control.

If *LineWidth* is greater than zero then vxBase will insert a carriage return-linefeed pair at this position (if a space happens to occupy that position) or back up to the first space that precedes this position and insert the CR-LF there. Hard carriage return-linefeed pairs are left intact.

Returns

A Visual Basic string that contains the contents of the memo.

Usage

Use *LineWidth* = 0 to display the memo in a multiline text box. If you wish to print the memo, use a *LineWidth* equal to the number of characters you wish to print on one line. The minimum line width is 10. If less than 10, the result will be the same as if you had passed a zero (i.e., no formatting).

Note: If the memo contains soft carriage returns and linefeeds, they are stripped before vxBase starts processing.

Note: Maximum memo length is 64k. You will require 128k (unformatted) or 192k (formatted) in text buffers to retrieve a string of this length. If you have monster memos, beware. The memory requirement for unformatted memos is 2x the memo length; for formatted memos it is 3x the memo length.

If you want the user to edit the contents of the memo in the text box (instead of using vxMemoEdit), use vxMemoRepl to write the memo string.

If vxSetAnsi(FALSE), the memo is converted from the OEM character set to ANSI before the string is returned.

Example

```
' Read memo into a multiline text box.
' Ensure that the multiline property is set
'  to TRUE at design time (this property is
' read only at run time). Visual Basic will
' take care of word wrap for us.
' ----------------------------------------
TextBox.Text = vxMemoRead("memofld", 0)

' to print the memo, we must format the
' lines with carriage returns and
' linefeeds.
' -------------------------------------
MemoString$ = vxMemoRead("memofld",80)
Printer.Print MemoString$
```

See Also

vxIsMemo, vxMemoClear, vxMemoEdit, vxReplMemo, vxSetAlias, vxSetAnsi

vxMemRealloc

Declaration

Declare Function vxMemRealloc Lib "vxbase.dll" (ByVal NumDbf As Integer, ByVal NumNtx As Integer, ByVal ReindexBuff As Integer) As Integer

Purpose

Decrease the initial vxBase memory requirements.

Parameters

NumDbf is the number of dbf files that will be opened simultaneously. The minimum number is 2.

NumNtx is the number of ntx files that will be opened simultaneously. The minimum number is 2.

ReindexBuff is passed as either TRUE or FALSE. If TRUE, a 64k buffer used to speed up creation of indexes and the reindex/pack routines is set up. If this parameter is passed as FALSE, the buffer is either removed (if is has already been created) or not set up at all.

Returns

TRUE if the memory reallocation was successful and FALSE if not.

Usage

The first call to vxBase allocates about 160,000 bytes to track open dbf files and open index files and also creates a 64k buffer used by vxReindex, vxPack, and vxCreateNtx. By calling vxMemRealloc(2, 2, FALSE) this huge block of memory may be reduced to about 35,000 bytes. If you are not going to be creating indexes, reindexing, or packing, the *ReindexBuff* may be passed as FALSE to remove an immediate 64k.

You should not use this function to INCREASE memory requirements. Let vxBase handle that automatically. DBF descriptor blocks and NTX buffers are each limited to 64k (FAR pointer arithmetic is used by vxBase) so the maximum number of open dbf and ntx files for ALL concurrent vxBase tasks is about 75. If more memory is required by vxBase, it will increase the size of each dbf or ntx object by the immediate amount required (to a maximum of 64k each).

This function should be called in your initialization routine. It should only be called ONCE. It is not designed as an all purpose vxBase memory manager; rather it is designed to let users with low memory situations customize specific vxBase applications.

WARNING: Use this function with care. This function CLOSES all open dbf and ntx files that belong to the current task before the memory is reallocated. If any files are open that belong to OTHER vxBase tasks, the function fails. ALWAYS test the result for a TRUE value to ensure the reallocation takes place.

Example

```
' The FIRST form load is used to initalize vxbase
' ------------------------------------------------
Sub Form_Load ()
   vxInit
   vxCtlGraySet
   vxSetLocks FALSE
   j% = vxCloseAll()
   If Not vxMemRealloc(2, 2, FALSE) Then
      MsgBox "Reallocation failed"
      End
   End If
End Sub
```

See Also

vxInit, vxMemCompact

vxMenuDeclare

Declaration

Declare Sub vxMenuDeclare Lib "vxbase.dll" (ByVal NumItems As Integer)

Purpose

Allocate memory for a menu structure that will be attached to an upcoming browse window for the currently selected database.

Parameters

NumItems is the number of vxMenuItem definitions that will immediately follow this function call.

Returns

Nothing.

Usage

Used only if you wish to define and attach your own menus to a browse window.

Allocated memory is automatically released when the file is closed with vxClose or vxCloseAll. vxTableReset also frees menu memory.

Example

```
' Declare and build a user menu
' ----------------------------
Call vxMenuDeclare(19)     ' 19 items in the menu
```

See Also

vxBrowse, vxMenuItem, vxTableReset

vxMenuItem

Declaration

Declare Sub vxMenuItem Lib "vxbase.dll" (ByVal MenuIndex As Integer, ByVal MenuLev As Integer, ByVal MenuString As String, ByVal MenuType As Integer)

Purpose

Define a menu item that belongs to the set of items declared by vxMenuDeclare. The menu so defined by vxMenuDeclare and vxMenuItem will be attached to an upcoming browse window for the currently selected database.

Parameters

MenuIndex is a number from 1 to the number of items declared for this menu by vxMenuDeclare. If the item is defined as a VX_RETURN type, this number plus 100 and negated is returned to you in the RetVal parameter as defined in vxBrowse if the user selects this item. For example, if the item is defined as vxMenuItem(6, 4, "&New", VX_RETURN), and the user selects it from the browse window, the browse return will be -106. The return value is negated so that it does not conflict with record numbers passed back if the user presses the ENTER key. 100 is added so that it does not conflict with the standard return values passed back by vxBrowse (e.g., BROWSE_CLOSED is -1 if the system menu is used to close the browse window). When the browse window is closed, the record pointer is always positioned at the record that was highlighted when the close occurred.

MenuLev is a number between zero and 1 less than the number of items declared for this menu by vxMenuDeclare. It signifies that this menu item is attached to the sub menu defined with a MenuIndex of this number. A *MenuLev* of zero refers to the top level menu attached to every window. For example, if you wished to have the word "File" appear on a browse menu with two items ("New" and "Open") beneath it, the menu would be defined as follows:

```
Call vxMenuDeclare(3)
Call vxMenuItem(1, 0, "&File", VX_MENUHEAD)
Call vxMenuItem(2, 1, "&New", VX_RETURN)
Call vxMenuItem(3, 1, "&Open", VX_RETURN)
```

Item 1 ("File") would be attached to menu level 0 and appear on the main browse window menu bar.

Item 2 ("New") is attached to the submenu defined with menu index 1.

Item 3 ("Open") is also attached to the submenu defined with menu index 1.

You can attach submenus within submenus by defining VX_MENUHEAD items inside of a submenu as shown in the example below (reproduced from VXFORM1 procedure UMenu_Click in the sample application).

MenuString is the text that is to appear on the menu line. If a separator bar is being defined (MenuType VX_SEPBAR), then a space " " should be passed. An ampersand placed in front of a character in the

string will make that character the mnemonic for the menu item (i.e., it will be underlined in the menu structure).

MenuType is the type of menu item being defined. It can be one of three different items:

VX_MENUHEAD is a header to a submenu. Its menu index is never returned from a browse. A complete menu structure may contain up to 64 VX_MENUHEADs. Windows doesn't set a limit to the number of nested menu levels; however, three levels of menus (the main menu bar and two levels of popup menus) is the deepest you will most likely want to go for sanity's sake.

VX_RETURN is a normal, selectable item. If the user selects a VX_RETURN item, the browse window is closed, the record pointer is positioned at the highlighted record, and the *RetVal* parameter passed to the vxBrowse function is filled with the *MenuIndex* of the selected item plus 100 negated.

VX_SEPBAR is a separator bar (a line) between menu items. Items defined as separator bars must have a *MenuString* passed as a single space. A separator bar menu index is never returned from a browse.

Types defined as VX_RETURN or VX_SEPBAR must have *MenuLev* parameters that point to a VX_MENUHEAD item or to *MenuLev* 0 (the browse window top level menu bar).

VX_MENUHEAD, VX_RETURN, and VX_SEPBAR are all defined as Global Constants in the vxbase.txt module.

Returns

Nothing.

Usage

Used to define your own menus on browse windows and then take action according to the values returned.

Example

```
Sub UMenu_Click ()
  ' this proc shows how to set up user defined
  ' menus on a browse window and also how to
  ' define the browse window initial position
  ' -----------------------------------------

  ' Open aircraft types file
  ' ------------------------
  AirtypesDbf = vxUseDbf("\vb\vxbtest\airtypes.dbf")
  If AirtypesDbf = FALSE Then
     MsgBox "Error Opening airtypes.dbf. Aborting."
     Exit Sub
  End If
  AirtypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")
  If AirtypesNtx = FALSE Then
     MsgBox "Error Opening airtypes.ntx. Aborting."
     j% = vxClose()
     Exit Sub
  End If

  ' Declare types table
  ' -------------------
  Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 2)
```

```
Call vxTableField(1, "Type", "category", VX_FIELD)
Call vxTableField(2, "Description", "catname", VX_FIELD)

' Declare and build a user menu
' -----------------------------
Call vxMenuDeclare(19)     ' 19 items in the menu

' the menu item params are:
'   (1) menu index number (from 1 to whatever was declared)
'   (2) attach this item to submenu number where 0 is the
'       top level browse menu and any other number must
'       refer to a menu index that was defined as VX_MENUHEAD
'   (3) the menu string. An ampersand in front of a character
'       will make that character the mnemonic. If a VX_SEPBAR
'       is being defined, pass a space " "
'   (4) the menu item type as defined in the global module
'       where VX_MENUHEAD is a submenu header,
'             VX_SEPBAR   is a separator bar, and
'             VX_RETURN   is a returnable item
'
' If any item is selected from the browse that is defined
' as VX_RETURN, the RetVal parameter passed to vxBrowse
' will contain the value of the menu index number plus 100
' and negated (e.g., menu item 6 below will return -106).
' The record pointer will be positioned at the record that
' was highlighted when the return was made. If the user
' presses the ENTER key in the browse, the RetVal will
' contain the record number that was highlighted when ENTER
' was pressed.

' the first menu item will set up a submenu on the
' browse table top level menu (Attach to item 0)
Call vxMenuItem(1, 0, "&File", VX_MENUHEAD)

Call vxMenuItem(2, 1, "&New", VX_RETURN)
' the item above is attached to the submenu defined as item 1

Call vxMenuItem(3, 1, "&Open", VX_RETURN)
Call vxMenuItem(4, 1, "&Save", VX_MENUHEAD)
' the item above creates another submenu within the File menu

Call vxMenuItem(5, 4, "&Old", VX_RETURN)
Call vxMenuItem(6, 4, "&New Name", VX_RETURN)
' the items above are under the sub menu defined as item 4)

Call vxMenuItem(7, 1, " ", VX_SEPBAR)
Call vxMenuItem(8, 1, "&Print", VX_RETURN)
Call vxMenuItem(9, 1, " ", VX_SEPBAR)
Call vxMenuItem(10, 1, "E&xit", VX_RETURN)

' now we'll set up another menu on the top level browse menu
Call vxMenuItem(11, 0, "&Edit", VX_MENUHEAD)

' and attach items to menu number 11 below it
Call vxMenuItem(12, 11, "Undo", VX_RETURN)
Call vxMenuItem(13, 11, " ", VX_SEPBAR)
Call vxMenuItem(14, 11, "Cu&t", VX_RETURN)
Call vxMenuItem(15, 11, "&Copy", VX_RETURN)
Call vxMenuItem(16, 11, "&Paste", VX_RETURN)
Call vxMenuItem(17, 11, " ", VX_SEPBAR)
Call vxMenuItem(18, 11, "Cl&ear", VX_RETURN)
Call vxMenuItem(19, 11, "&Delete", VX_RETURN)

' The proc below will set up an initial position
' for the browse window
' ---------------------------------------------
Call vxBrowsePos(10, 5, 50, 15)
```

```
  ' the coordinates are in familiar character and line
  ' units. The first param is x (characters in from left),
  ' the second param is y (lines down from top), the third
  ' param is the width of the window in characters, and the
  ' last param is the window height in lines

  ' if the user moves or sizes the window, and subsequent
  ' vxBrowse calls are made without an intervening close of the
  ' file, the window will retain its last position and size.

  ' now we set up the browse
  ' ------------------------
  TypeReturn = 0
  VXFORM1.UMenu.Enabled = FALSE

  j% = vxSelectDbf(AirtypesDbf)
  j% = vxSelectNtx(AirtypesNtx)

  Call vxBrowse(VXFORM1.hWnd, AirtypesDbf, AirtypesNtx, TRUE, TRUE, FALSE, 0, "Aircraft
Types", TypeReturn)
  ' Note that the EDIT menu parameter should be FALSE if you
  ' are defining your own menus.

  MsgBox "Value returned from browse was " + Str$(TypeReturn)

  j% = vxClose()
  VXFORM1.UMenu.Enabled = TRUE
End Sub
```

See Also

vxBrowse, vxMenuDeclare

vxNtxCurrent

Declaration

Declare Function vxNtxCurrent lib "vxbase.dll" () As Integer

Purpose

Get the current index select area.

Parameters

None.

Returns

An integer pointing to the active index for the currently selected dbf. This integer is the same one returned by vxUseNtx when the file was opened.

FALSE (zero) is returned if there is no active index or if any other error occurs (such as no current dbf selected).

Usage

The programmer can let the user pick an active index from a list of indexes. In this case, you never know exactly what index is the current selection. If you have to find out, this is the function to use.

Example

```
' put current index name in text box
' --------------------------------
NtxNameBox.text = RTrim$(vxNtxName(vxNtxCurrent()))
```

See Also

vxAreaNtx, vxNtxExpr, vxNtxName, vxNtxSubExpr, vxSelectNtx, vxUseNtx

vxNtxDeselect

Declaration

```
Declare Function vxNtxDeselect Lib "vxbase.dll" () As Integer
```

Purpose

Temporarily turn off index ordering on the currently selected file.

Parameters

None.

Returns

TRUE if the operation is successful and FALSE if not.

Usage

If or any reason you wish to revert to record number ordering use this command. Any open indexes attached to the file remain open and unlocked. As soon as one of the indexes is selected again, index ordering is resumed.

This function is handy if you are skipping through a file record by record and changing key values. If index ordering is on, once a field has been changed that affects the selected index, the next skip will probably take you to a place you don't want to go. With vxNtxDeselect you can change fields that affect keys at will, reselect an index, and then reindex the file without having to close and then reopen all of the index files.

Example

```
If vxNtxDeselect() Then
   ChangeKeyValues
   j% = vxSelectNtx(BuyerNtx)
   j% = vxReindex()
End If
```

See Also

vxSelectNtx, vxUseNtx

vxNtxExpr

Declaration

Declare Function vxNtxExpr Lib "vxbase.dll" (ByVal NtxArea As Integer) As String

Purpose

Extract the index expression for the specified, open index.

Parameters

NtxArea is the select area of an index file returned by vxUseNtx or vxAreaNtx.

Returns

A Visual Basic string that contains the expression used to create the specified index.

Usage

Especially useful in creating files at run time that are copies of existing files and that are to be indexed in the same way.

Example

```
If Not vxCopyStruc(BatchName$) Then
   MsgBox "Error in batch file creation"
   j% = vxClose()
   Exit Sub
Else
   ' now create index same as master file
   ' ------------------------------------
   IndexExpr$ = vxNtxExpr(TrMasterNtx%)
   If Not vxCreateNtx(BatchName$, IndexExpr$) Then
      MsgBox "Error in index creation"
      Kill FileSpec$
      j% = vxClose()
      Exit Sub
   End If
End If
```

See Also

vxCreateNtx, vxNtxName, vxNtxSubExpr, vxUseNtx

vxNtxName

Declaration

Declare Function vxNtxName Lib "vxbase.dll" (ByVal NtxArea As Integer) As String

Purpose

Extract the name of the specified index file as it was passed to the vxUseNtx function.

Parameters

NtxArea is the select area of an index file returned by vxUseNtx or vxAreaNtx.

Returns

A Visual Basic string that contains the name of the file.

Usage

Used to head forms or reports.

Example

```
' display index items
' -------------------
NtxName.text = vxNtxName(BuyerNtx)
NtxExpr.text = vxNtxExpr(BuyerNtx)
```

See Also

vxAreaNtx, vxNtxExpr, vxNtxSubExpr, vxUseNtx

vxNtxRecNo

Declaration

```
Declare Function vxNtxRecNo lib "vxbase.dll" () As Long
```

Purpose

Get the ordinal position of the current key in the active index for the current dbf.

Parameters

None.

Returns

A long integer that describes the ordinal position of the current dbf record in the active index. FALSE (zero) is returned if an error occurs. If FALSE, the index pointer MAY be invalid. It is the programmer's responsibility to trap the error and reposition if necessary.

Usage

Primarily used to position a vertical scroll thumb when building a scrollable list of dbf records. The physical position of the record in the database probably bears little relation to its logical position in the index.

NOTE: This number is calculated by moving through the btree in reverse sequence from the current position until the first index entry is reached (maintaining a count all the while). After the current position is ascertained, the index pointer is moved back to the original position.

If the dbf file contains more than 5,000 records, or if the keys are inordinately large, this function can consume a great deal of time.

Example

```
j% = vxSelectDbf(TestDbf)
j% = vxGo(vxNumRecs()/2)

' display dbf record number
' and then ntx record number
debug.print vxRecNo()
debug.print vxNtxRecNo()
```

See Also

vxRecNo

vxNtxSubExpr

Declaration

Declare Function vxNtxSubExpr Lib "vxbase.dll" (ByVal NtxArea As Integer) As String

Purpose

Extract the conditional expression that controls the insertion and deletion of keys in a subindex.

Parameters

NtxArea is the select area of an index file returned by vxUseNtx or vxAreaNtx.

Returns

A string that contains the conditional expression used to create the subindex. The string is either in Visual Basic format or C format depending upon the value of vxSetString.

Usage

Especially useful in creating files at run time that are copies of existing files and that are to be indexed in the same way. Or in reporting the conditions of index insertion to the user.

Example

```
If vxIsSubNtx(vxNtxCurrent()) Then
   NumRecs& = vxNumRecsSub()
   Form1.Caption = "Subindex on " + vxNtxExpr(vxNtxCurrent()) +
                   " For Condition " + vxNtxSubExpr(vxNtxCurrent())
Else
   NumRecs& = vxNumRecs()
   Form1.Caption = "Master Index on " + vxNtxExpr(vxNtxCurrent())
End If
```

See Also

vxCreateSubNtx, vxIsSubNtx, vxNtxCurrent, vxNtxExpr, vxNtxSubExpr, vxNumRecsSub

vxNumRecs

Declaration

```
Declare Function vxNumRecs Lib "vxbase.dll" () As Long
```

Purpose

Extract the number of records in the current database file.

Parameters

None.

Returns

A long integer containing the number of records in the file. This includes logically deleted records.

Usage

Generally used as a FOR loop counter when you wish to process every record in the file or as a statistic to determine the approximate size of the file.

Example

```
HeadSize& = (vxFieldCount() * 32) + 34
FilSize& = (vxNumRecs() * vxRecSize()) + HeadSize&
FileSize.text = Format$(FilSize&, "#,###,###,###")
```

See Also

vxFieldCount, vxNumRecsFilter, vxNumRecsSub, vxRecSize

vxNumRecsFilter

Declaration

Declare Function vxNumRecsFilter Lib "vxbase.dll" () As Long

Purpose

Return the number of records in the database that pass the defined filter.

Parameters

None. The number of records returned is for the currently selected database.

Usage

Useful for generating accurate scroll bar extents and as a FOR LOOP counter. Note that this function does exactly what you would do to determine the number of records in a database that satisfy some condition. It must read every record in the database, evaluate the filter, and increment a counter. It is done at a lower level but still can take a lot of time in a large database.

Example

```
j% = vxSelectDbf(TestDbf")
Debug.Print vxNumRecs()
Call vxFilter("trim(vxcountry)='CANADA')
Debug.Print vxNumrecsFilter()
```

See Also

vxFilter, vxFilterReset, vxNumRecs, vxNumrecsSub

vxNumRecsSub

Declaration

Declare Function vxNumRecsSub Lib "vxbase.dll" () As Long

Purpose

Return the number of records in a subindex.

Parameters

None. The subindex MUST be the currently selected index.

Usage

Generally used as a FOR LOOP counter or as a statistic. It may also be used to set an accurate vertical scroll bar extent.

Example

```
If vxIsSubNtx(vxNtxCurrent()) Then
   NumRecs& = vxNumRecsSub()
   Form1.Caption = "Subindex on " + vxNtxExpr(vxNtxCurrent()) +
                   " For Condition " + vxNtxSubExpr(vxNtxCurrent())
Else
   NumRecs& = vxNumRecs()
   Form1.Caption = "Master Index on " + vxNtxExpr(vxNtxCurrent())
End If
```

See Also

vxCreateSubNtx, vxIsSubNtx, vxNtxCurrent, vxNtxExpr, vxNtxSubExpr

vxPack

Declaration

Declare Function vxPack Lib "vxbase.dll" (ByVal Hwnd As Integer) As Integer

Purpose

Remove all logically deleted records from the file and reindex.

Parameters

Hwnd is the hWnd property of an active Visual Basic Form. This window acts as parent to a window that displays a meter bar signifying the progress of the pack visually and in percentage complete.

Returns

TRUE if the operation was successful and FALSE if not. Always returns FALSE if the file has been opened with vxUseDbfRO. After the pack is complete, A meter bar will be displayed that charts the progress of the rebuilding of the indexes. If memos are attached to the dbf, a third meter bar is displayed that shows you how the memo compression is coming along.

Meters bars are only displayed if vxSetMeters is TRUE (the default). The programmer may alternatively specify his own gauge control with vxSetGauge.

Usage

A file maintenance item that packs all files in your application should be a standard feature of any xBase application.

Please ensure that ALL index files that belong to the dbf being packed are open.

Once a file has been packed, deleted records are no longer available for recall.

If a memo file is attached to the file being packed, it is also packed after the deleted records are removed. A temporary file with the same name as the .dbf but with an extension of .$$$ is created. Every record is analyzed for the presence of valid memo block references and, if any are found, the old memo or bitmap is copied to the new .$$$ memo file. Invalid memo blocks (which usually abound in xBase memo files) are not copied to the new file. At end of file, the old memo file is erased and the .$$$ file is renamed with the standard memo file .dbt extension.

If there is not enough space on the drive to hold a file of the same size as the old memo file, the memo file is not packed.

Always use vxAreaDbf to ensure that the file is not open in any active task before commencing the pack operation.

Multiuser Considerations

The dbf and its indexes are locked for the duration of the operation. Consider opening the file with vxuseDbfEX.

Example

```
' removes logically deleted records
' and reindexes
' ------------------------------------------------

' make sure file isn't open
' -------------------------
j% = vxAreaDbf("\vb\vxbtest\airtypes.dbf")
If j% = FALSE Then
   AirtypesDbf = vxUseDbf("\vb\vxbtest\airtypes.dbf")
   AirTypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")
   j% = vxPack(VXFORM1.hWnd)
   j% = vxClose()
End If
```

See Also

vxAreaDbf, vxCopy, vxDeleteRec, vxSetGauge, vxSetMeters, vxUseDbfEX

vxPictureImport

Declaration

Declare Function vxPictureImport Lib "vxbase.dll" (ByVal BmpFileName As String, ByVal MemoFieldName As String) As Integer

Purpose

Import a bitmap from a system .BMP file into a memo field. The image may be displayed with vxPictureRead. The maximum size of the bitmap is 16 megabytes.

Parameters

BmpFileName is the complete name of the bitmap file including path and extension. Files other than bitmaps may be stored into the memo file but they may not be read with vxPictureRead (unless they are run length encoded compressed variants of BMPS - i.e., RLE files in either 4 or 8 bit per pixel format). The file name may be represented by a literal string or a string variable.

MemoFieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *MemoFieldName* may be qualified with a valid alias name that points to any open database.

Returns

FALSE if the function fails and TRUE if successful. FALSE is always returned if the associated dbf has been opened as Read Only with vxUseDbfRO.

Usage

It is much more efficient both from a retrieval standpoint and from a disk management standpoint to store bitmaps you wish to have associated with database records in a single source file.

Store pictures of people in personnel files, parts images in inventory files, homes in real estate files, etc.

Files other than BMPs may also be stored in a memo file. Note, however, that only BMP or RLE format files are converted by vxPictureRead for display in a Visual basic picture box. The memo link parameters included in the vxCtlBrowse function will also result in bitmap display WITHOUT any effort required by the programmer other that defining the memo window and memo field name to the vxCtlBrowse function.

Collecting Bitmaps

vxBase takes advantage of the rich body of functions included in Visual Basic to handle bitmap files. Bitmaps are the Windows norm; all paint programs, viewers, etc. can handle bitmaps - and most programs that deal with pictures can convert foreign formats (e.g., GIF) to BMPs. As a last resort, cut a picture into the clipboard and paste it into a Windows PAINT window. It can then be stored as a .BMP file.

Once an image is stored in a BMP file it can be transferred to the

memo file with the vxPictureImport function. To retrieve the bitmap, vxBase uses the standard Windows Clipboard. It puts the bitmap out to the clipboard as a DIB (device independent bitmap). The Visual Basic Clipboard.GetData(8) function then is used to retrieve the image from the clipboard and display it in a VB Picture Box. The box can have the AUTORESIZE property set to TRUE as in the sample application if the images are all different sizes.

Bitmap files can be created from existing images you may be using in a Visual Basic program by calling the SavePicture function. You can then call vxPictureImport to store the image in a memo file by using the name of the bitmap file you created with SavePicture. You can also use SavePicture to export pictures from a memo file by first reading them into a picture box with vxPictureRead.

Example

```
' the name of the bmp picture file is the
' same as the string in field "title" so
' we can import the bmps into the memo file
' by cocatenating ".bmp" to the trimmed field
' contents
' ----------------------------------------
j% = vxTop()
If Not vxIsPicture("pic") Then
   For i& = 1 To 13 ' there are 13 recs in the file
      j% = vxGo(i&)
      ftitle$ = vxFieldTrim("type")
      fname$ = "c:\magic\bmp\" + ftitle$ + ".bmp"
      If Not vxPictureImport(fname$, "pic") Then
         MsgBox "Import Failed"
      End If
   Next i&
   j% = vxClose() ' close ensures buffers flushed
   AirPicsDbf = vxUseDbf("\vb\vxbtest\airpics.dbf")
   j% = vxSelectDbf(AirPicsDbf)
End If
```

See Also

vxIsPicture, vxMemoClear, vxPicturePrint, vxPictureRead

vxPicturePrint

Declaration

Declare Function vxPicturePrint Lib "vxbase.dll" (ByVal MemoFieldName As String) As Integer

Purpose

Print a MONOCHROME bitmap that has been stored in a vxbase memo file.

Parameters

MemoFieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *MemoFieldName* may be qualified with a valid alias name that points to any open database.

Returns

FALSE if the function fails and TRUE if successful.

Usage

The bitmap image is expanded or compressed to best fit the current page size set for the selected printer. Color bitmaps print very poorly because the Windows StretchBlt function used by vxPicturePrint does not convert the color images to black and white very well. If you are going to print a lot of bitmaps, use a conversion program to convert color bitmaps to black and white before saving to the memo file.

Example

```
Sub ButtonPrint_Click ()
   j% = vxSelectDbf(AirPicsDbf)
   TopRec& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_GETTOPREC, 0)
   RecNum& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_GETCURRENTREC, 0)
   j% = vxGo(RecNum&)
   If Not vxPicturePrint("pic") Then
      MsgBox "Print Failed"
   End If
   r& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_REFRESH, ByVal TopRec&)
End Sub
```

See Also

vxIsPicture, vxPictureImport, vxPictureRead

vxPictureRead

Declaration

Declare Function vxPictureRead Lib "vxbase.dll" (ByVal PicHwnd As Integer, ByVal MemoFieldName As String) As Integer

Purpose

Display a bitmap that was stored in a memo file with vxPictureImport in a defined window.

Parameters

PicHwnd is the window handle of the window that will receive the image. In Visual basic, use vxCtlHwnd to convert a control handle to a window handle.

MemoFieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *MemoFieldName* may be qualified with a valid alias name that points to any open database.

Returns

TRUE if the operation was successful and FALSE is not.

Usage

Only BITMAPS that have been stored with vxPictureImport may be extracted with vxPictureRead. vxPictureRead assumes the binary object contained in the memo is a formatted bitmap which contains a Windows BITMAPFILEHEADER followed by a BITMAPINFOHEADER followed by an array of RGBQUAD structures. All of this information is then followed by the bitmap data itself. vxPictureRead creates a DIB (device independent bitmap) out of the Windows data structures and passes the DIB to the clipboard, where it can easily be extracted and placed into a Visual Basic picture box (or onto a form) with the ClipBoard.GetData(8) function.

If the structure contained in the memo is not a formal bitmap, who knows what result?

If you wish to store Binary Large Objects (BLOBs) in the memo file other than bitmaps, you must use the vxBlobWrite and vxBlobRead functions to access the data. These functions use Windows Global memory handles as parameters and are not easily available to the Visual Basic programmer.

NOTE: The memo link parameters included in the vxCtlBrowse function will also result in bitmap display WITHOUT any effort required by the programmer other than defining the memo window and memo field name to the vxCtlBrowse function. When vxCtlBrowse displays a bitmap, it automatically sizes the memo link window to the size of the bitmap. The top left corner of your memo window is anchored. If there is not enough room on the form to contain the entire bitmap, it is clipped on the right and/or the bottom to the limits of the parent form.

Example

The following code is a complete reproduction of the code contained in VYFORM2 in the vxbtest project sample application:

```
' static switch set to TRUE in form
' load procedure so we know when this
' form is first loaded
Dim FirstTime As Integer

Sub BrowseBox_KeyDown (KeyCode As Integer, Shift As Integer)
  ' whenever a record is highlighted, this
  ' proc receives a middle button code
  ' from the ctlBrowse so we can dynamically
  ' display the picture in the memo field
  ' ----------------------------------------
  If KeyCode = 4 Then  ' middle button?
     j% = vxSelectDbf(AirPicsDbf)
     RecNum& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox),
                              VXB_GETCURRENTREC, 0)
     j% = vxGo(RecNum&)
     VYFORM2.Caption = vxFieldTrim("Title")
     If vxPictureRead(vxCtlHwnd(PicBox), "pic") Then
        ' the "8" param below is CF_DIB
        PicBox.Picture = Clipboard.GetData(8)
        ' If you want to leave the picture in
        ' the clipboard, comment out line below
        Clipboard.Clear
     Else
        PicBox.Picture = LoadPicture() ' clears the picture area
     End If
  End If

  ' ------------------------------------------------------------------
  ' NOTE: the code above is an example of using vxPictureRead.
  '       In this case (displaying records in a vxCtlBrowse table),
  '       it would be much more efficient to define the memo window
  '       and the memo field name to the vxCtlBrowse function instead
  ' ------------------------------------------------------------------
End Sub

Sub BrowseBox_KeyPress (KeyAscii As Integer)

  ' NOTE: YOU MUST ALWAYS TRAP THE ENTER KEY
  '       AND CHANGE THE KEYASCII CODE TO
  '       A ZERO WHEN USING VXCTLBROWSE
  '       EVEN IF YOU DON'T USE IT
  ' ---------------------------------------
  If KeyAscii = 13 Then
     KeyAscii = 0
     Exit Sub
  End If

  ' if ESC key is received, then emulate
  ' exit button press
  ' ------------------------------------
  If KeyAscii = 27 Then
     KeyAscii = 0
     ButtonExit_Click
     Exit Sub
  End If

End Sub

Sub ButtonExit_Click ()
  Unload VYFORM2
End Sub
```

```
Sub Form_Load ()
  ' set FirstTime switch on for Paint
  ' ---------------------------------
  FirstTime = True

  ' register the default database as the master
  ' -----------------------------------------
  AirPicsDbf = vxUseDbf("\vb\vxbtest\airpics.dbf")
  j% = vxSelectDbf(AirPicsDbf)

  ' first time load pictures
  j% = vxTop()

  ' the name of the bmp picture file is the
  ' same as the string in field "title" so
  ' we can import the bmps into the memo file
  ' by cocatenating ".bmp" to the trimmed field
  ' contents
  ' -----------------------------------------
  If Not vxIsPicture("pic") Then
     For i& = 1 To 13
        j% = vxGo(i&)
        ftitle$ = vxFieldTrim("type")
        fname$ = "c:\magic\bmp\" + ftitle$ + ".bmp"
        If Not vxPictureImport(fname$, "pic") Then
           MsgBox "Import Failed"
        End If
     Next i&
     j% = vxClose()
     AirPicsDbf = vxUseDbf("\vb\vxbtest\airpics.dbf")
     j% = vxSelectDbf(AirPicsDbf)
  End If

  ' set up the browse
  ' -----------------
  Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 1)
  Call vxTableField(1, "Type", "type", VX_FIELD)

  Call vxBrowseCase(VX_UPPER)
  Call vxBrowseSetup(0, 0, 0, 1, "Helv", 15, VX_SEMIBOLD, 0, 0, 0, 0)
  ' If the typeface is too large on your display,
  ' CHANGE the parameter following "Helv" above to
  ' a smaller number
  ' ------------------------------------------------

  ' change the mouse pointer in the browse box
  ' from an I-Beam to an arrow to stop any flicker
  ' ------------------------------------------------
  BrowseBox.MousePointer = 1

End Sub

Sub Form_Paint ()
 ' register the database with this window
  ' --------------------------------------
  j% = vxSelectDbf(AirPicsDbf)

  ' make the form 3-d
  ' -----------------
  Call vxFormFrame(VYFORM2.hWnd)
  Call vxCtlStyle(BrowseBox, VX_RECESS)

  ' initiate the browse the first time only
  ' ---------------------------------------
  If FirstTime = True Then
     j% = vxCtlBrowse(vxCtlHwnd(BrowseBox), AirPicsDbf,
```

```
                    0, 0, 0, 0, " ")
     FirstTime = False
  End If

  ' on initial paint of the browse, middle button
  ' keydown is not sent to browse box so we want
  ' to do our dynamic display here as well as
  ' from a keydown in the browse box code
  ' -------------------------------------------
  Call BrowseBox_KeyDown(4, 0)
End Sub

Sub Form_Resize ()
  VYFORM2.Refresh
End Sub

Sub Form_Unload (Cancel As Integer)
' close the browse
  ' ----------------
  k& = vxCtlBrowseMsg(vxCtlHwnd(BrowseBox), VXB_CLOSE, 0)

  ' close all the files
  ' -------------------
  j% = vxCloseAll()

  ' deregister the window and release memory
  ' ----------------------------------------
  vxWindowDereg (VYFORM2.hWnd)
End Sub
```

See Also

vxIsPicture, vxPictureImport, vxPicturePrint

vxPrinterDefault

Declaration

Declare Function vxPrinterDefault lib "vxbase.dll" () As String

Purpose

Retrieve the Windows default printer string in a format suitable for setting the default printer with vxPrinterSelect.

Parameters

None.

Returns

A Visual Basic String (or ASCIIZ string if vxSetString is 1) that contains a string that may be used by vxPrinterSelect to set the default Windows printer.

The format of the string returned is
PRINTER NAME,DRIVER,PORT:

Usage

Used to display the current default printer, and to re-set the default printer if it is changed with vxPrinterSelect.

Example

```
' display the default printer
' --------------------------
PrinterBox.Text = vxPrinterDefault()
```

See Also

vxPrinterEnum, vxPrinterSelect, vxSetupPrinter

vxPrinterEnum

Declaration

Declare Function vxPrinterEnum lib "vxbase.dll" (ByVal PIndex As Integer) As String

Purpose

Enumerate printers as defined in the WIN.INI file and retrieve a string suitable for setting the default printer with vxPrinterSelect.

Parameters

PIndex is an index number of the printer you wish to enumerate.

Returns

A Visual Basic String (or ASCIIZ string if vxSetString is 1) that contains a string that may be used by vxPrinterSelect to set the default Windows printer.

The format of the string returned is
PRINTER NAME,DRIVER,PORT:

If a single space is returned, there are no more printers to be found.

Usage

Would normally be used in a loop to enumerate the printers into a list box so the user could select the printer he wished to make current.

Example

```
' ------------------------------------------------------
' the following is taken from the sample app VYFORM1
'
' VYFORM1 contains:
'    1. a listbox named PrinterList
'    2. a button to set the default named SelectButton
'    3. an Exit button named ExitButton
'    4. a text box named PrinterBox to display the
'       selected printer
' all of the code for VYFORM1 is shown below
' ------------------------------------------------------

' -------------------------------------
' unload form when exit button clicked
' -------------------------------------
Sub ExitButton_Click ()
  Unload VYFORM1
End Sub

' ----------------------------------------
' when form is loaded, enumerate printers
' and put in list box
' ----------------------------------------
Sub Form_Load ()

  ' display the default printer
  ' ---------------------------
  PrinterBox.Text = vxPrinterDefault()

  j% = 1   ' the printer index
  PrinterOk% = TRUE
```

```
Do
   PrinterName$ = vxPrinterEnum(j%)
```

```
     ' all printers enumerated when vxPrinterEnum
     ' returns a single space
     ' ---------------------------------------
     If PrinterName$ = " " Then
        PrinterOk% = FALSE
     Else
        PrinterList.AddItem PrinterName$
        j% = j% + 1
     End If
  Loop Until Not PrinterOk%
End Sub

' --------------------
' make the form pretty
' --------------------
Sub Form_Paint ()
  Call vxFormFrame(VYFORM1.hWnd)
  Call vxCtlStyle(PrinterBox, VX_RECESS)
  Call vxCtlStyle(PrinterList, VX_RAISE)
End Sub

' ---------------------------------------
' if user resizes form, get rid of old frame
' ---------------------------------------
Sub Form_Resize ()
  VYFORM1.Refresh
End Sub

' ----------------------------------
' if user double clicks a selection,
' emulate select button press
' ----------------------------------
Sub PrinterList_DblClick ()
  SelectButton_Click
End Sub

' ----------------------------------
' if user selects a printer, display
' it in PrinterBox and also set the
' default printer
' ----------------------------------
Sub SelectButton_Click ()
  PrinterBox.Text = PrinterList.Text

  ' change the default printer
  ' --------------------------
  If vxPrinterSelect((PrinterBox.Text)) Then
     MsgBox "Default printer changed!"
  Else
     MsgBox "Error in Printer Name"
  End If
End Sub
```

See Also

vxPrinterDefault, vxPrinterSelect, vxSetupPrinter

vxPrinterSelect

Declaration

Declare Function vxPrinterSelect lib "vxbase.dll" (ByVal PrinterName As String) As Integer

Purpose

Select a new Windows default printer.

Parameters

PrinterName is a structured string used to set the default printer.

It is of the form

PRINTERNAME,DRIVERNAME,PORT:

For example,

EPSON LQ-500,EPSON24,LPT1:

A structured string that may be used to select a printer may be obtained with function vxPrinterEnum.

Returns

TRUE if a new printer has been correctly selected (or if the select string matches the current default printer already). FALSE is returned if the string does not match any printer that vxBase enumerates internally from the [devices] section of the WIN.INI file.

Usage

Allow the user to select a new printer without the need to bring up the Windows Control Panel.

The printer select string would normally be obtained from the user through a list box built with the vxPrinterEnum function.

Example

See the example in vxPrinterEnum for a complete routine that lets the user select a new printer.

See Also

vxPrinterDefault, vxPrinterEnum, vxSetupPrinter

vxRecall

Declaration

Declare Function vxRecall Lib "vxbase.dll" () As Integer

Purpose

Remove the deleted flag from the current record.

Parameters

None.

Returns

TRUE if the operation was successful and FALSE if not. Always returns FALSE if the file has been opened with vxUseDbfRO.

Usage

Undelete a record that was perhaps mistakenly deleted.

Example

```
If vxDeleted() Then
   j% = MsgBox("Record deleted. Recall?", 52)
      If j% = 6 Then
      If vxRecall() Then
         UpdateRec
      End If
   End If
End If
```

See Also

vxCopy, vxDeleted, vxDeleteRec, vxPack

vxRecNo

Declaration

```
Declare Function vxRecNo Lib "vxbase.dll" () As Long
```

Purpose

Extract the physical record number of the current record.

Parameters

None.

Returns

A long integer that contains the current record number.

Usage

Normally used to save a record number, unlock the record, perform some operation on the data from that record that has perhaps been stored in form controls, and then go back to that record and update it.

vxRecNo MUST be used in this fashion when editing a memo.

NOTE: When a vxCtlBrowse is active, vxRecNo will not necessarily return the record number of the highlighted record. Use VXB_GETCURRENTREC passed as a parameter to vxCtlBrowseMsg to retrieve the number of the highlighted record.

Example

```
 If vxSeek("ABC") Then          ' find the record to update
     RecNum& = vxRecNo()         ' save the record number
     Sig% = vxInteger("CustSig") ' and the signature
     Name.text = vxField("Name")  ' store the form vars
     Status.text = vxfield("Stat")

     ' now unlock the record
     ' ---------------------
     j% = vxUnlock()

     ' now perform the update on the vis basic form
     ' --------------------------------------------
     CustRecordUpdate

     ' now retrieve the record and test if anyone else
     ' has changed it
     ' ----------------------------------------------
     j% = vxGo(RecNum&)
     If Sig% <> vxInteger("CustSig") Then
        MsgBox "Another user beat you to it. Redo!"
     Else
        Call vxReplString("Name", (Name.text))
        Call vxReplString("Stat", (Status.text))
        Call vxReplInteger("CustSig", (Sig% + 1))
     End If
     j% = vxUnlock()
  End If
```

See Also

vxGo, vxMemoEdit, vxSkip

vxRecord

Declaration

Declare Function vxRecord Lib "vxbase.dll" (RecStruct As Any) As Integer

Purpose

Copy the contents of the record buffer to a data structure or fixed length string.

Parameters

RecStruct is a defined record structure or a predimensioned fixed string.

Returns

TRUE if the copy was successful. Otherwise, it is FALSE. A FALSE condition can occur if there is no selected database or if the current record number is invalid (e.g., skip past end of file).

Usage

Use to fill a record structure defined in the global module or to fill a fixed string variable with the complete contents of the record buffer. If you are defining a fixed string to hold the result of vxRecord, ensure that it is long enough to hold the entire record (including the deletion flag field).

vxRecord MUST be used to extract the contents of a character field that has a length exceeding 255.

All xBase data is saved on disk in character format. Numeric fields are saved as right justified numbers. Date fields are stored as CCYYMMDD. Memo fields are ten digit numbers that refer to the relative block number of the memo in the .DBT file. The first character in the record is a delete flag ('*' if deleted, blank if not).

If you use the vxRecord function, you are responsible for using the native language data conversion functions to convert numbers and dates to formats that the language can understand. Alternatively, you could define a record structure, fill it with the vxRecord function, and use only those elements that are defined as Character fields. vxBase functions such as vxDouble and vxDateFormat could still be used to convert the xBase ASCII data to numbers and dates. For a complete example of vxRecord usage, see the VXFORM8 code in the sample application.

Records extracted with vxRecord may be replaced with vxReplRecord.

vxRecord may also be used to extract data for languages other than Visual Basic. For example, Realizer users could define a string and pass the address of that string to the vxRecord function.

```
EXTERNAL "vxbase.dll" FUNC vxRecord(POINTER) As INTEGER

RString = String$(50, 0)
```

```
j = vxRecord(RString)
```

Field elements can then be extracted with the MID$ function.

If languages other than Visual Basic are used, remember to use the vxSetString(1) function as the first call to vxBase in your program. This will ensure that a pointer to a standard ASCIIZ string is passed from all vxBase functions that return strings instead of creating Visual Basic variable length strings.

If you wish to use a string instead of a typedef in Visual Basic, call the function as follows:

```
Buff$ = String$(512,0) ' string long enough to hold record
j% = vxRecord(ByVal Buff$)
```

Example

```
' Record structure is defined in the global module
' ------------------------------------------------

' ----------------------------------------
' define types file record structure for
' use in vxform8 and the vxRecord function
' ----------------------------------------
Type CatRec
   cDelFlag As String * 1
   Category As String * 3
   CatName As String * 35
End Type
' note that every xbase record structure MUST begin
' with a single character deletion flag
' ------------------------------------------------

' CODE in VXFORM8 module
' use vxRecord instead of vxField to display
' character fields
' ----------------------------------------
Sub Form8Display ()
   Dim Crec As CatRec

   If Not vxEof Then
      If vxRecord(Crec) Then
         CatBox.text = Crec.Category
         CatNameBox.text = Crec.CatName
      Else
         CatBox.text = ""
         CatNameBox.text = ""
      End If
   End If
End Sub
```

See Also

vxField, vxFieldTrim, vxReplRecord, vxSetString

vxRecSize

Declaration

Declare Function vxRecSize Lib "vxbase.dll" () As Integer

Purpose

Extract the size of the record in the currently selected database.

Parameters

None.

Returns

An integer containing the record size.

Usage

Generally used as a statistic to determine the approximate size of the file.

Example

```
HeadSize& = (vxFieldCount() * 32) + 34
FilSize& = (vxNumRecs() * vxRecSize()) + HeadSize&
FileSize.text = Format$(FilSize&, "#,###,###,###")
```

See Also

vxFieldCount, vxNumRecs

vxReindex

Declaration

```
Declare Function vxReindex Lib "vxbase.dll" () As Integer
```

Purpose

Recreate existing open index files.

Parameters

None.

Returns

TRUE if the operation was successful and FALSE if not. Always returns FALSE if the file has been opened with vxUseDbfRO.

Usage

Index files are among the most volatile files in an xBase application. They are constantly being reorganized and parts of them are being rewritten every time we get significant changes or record movement in large files. For this reason they are also easily corrupted, especially by forces beyond our control (such as power failures, static discharges, etc.).

If records don't appear in a skip procedure or a vxBrowse table that you KNOW are there, the index is probably corrupted. You can use the vxTestNtx function to test the integrity of an index. Always set up a file maintenance utility that either packs the files (which automatically reindexes them as well) or simply reindexes.

Ensure that all index files belonging to the current database are open.

Always use vxAreaDbf to ensure that the file is not open in any active task.

If vxSetMeters is TRUE (the default), a meter bar window is presented that charts the progress of the reindex routine for each index file being recreated. The programmer may alternatively specify his own gauge control with vxSetGauge.

Always close the file after a reindex to ensure that all buffers are flushed to disk.

Multiuser Considerations

The dbf and its indexes are locked for the duration of the operation.

Example

```
j% = vxAreaDbf("\vb\vxbtest\airtypes.dbf")
If j% = FALSE Then
   AirtypesDbf = vxUseDbf("\vb\vxbtest\airtypes.dbf")
   AirTypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")
   If Not vxReindex() Then
      MsgBox "Reindex unsuccesssful. Dbf corrupted."
   End If
   j% = vxClose()
End If
```

See Also

vxAreaDbf, vxPack, vxSetGauge, vxSetMeters, vxTestNtx

vxReplDate

Declaration

Declare Sub vxReplDate Lib "vxbase.dll" (ByVal FieldName As String, ByVal DateString As String)

Purpose

Replace an xBase date field with a Visual Basic string formatted as per specifications below.

Parameters

FieldName is either a string variable or a literal string that contains a valid date field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

DateString is a string representation of a date in the format dd-mmm-yyyy.

Returns

Nothing.

Usage

Change a date field in the database. A Visual Basic serial date must be formatted with the command Format$(SerialDate, "dd-mmm-yyyy") before it is passed to vxBase.

All xBase data is stored in string format within the record. The date could also be formatted with Format$(SerialDate, "yyyymmdd") and replaced within the record with the vxReplString command. xBase dates are stored as "yyyymmdd" internally.

The record buffer is not written to disk until an explicit vxWrite is issued or a command is issued that changes the status of the record pointer (such as vxGo, vxSkip, vxSeek, etc.). In a multiuser environment, always use an explicit vxWrite to ensure the record is available in its changed form as soon as possible.

This function is ignored if the file has been opened as Read Only with vxUseDbfRO.

Example

```
' set up date strings in preparation for replace
' -----------------------------------------------
RDate$ = Format$(Now, "dd-mmm-yyyy")
If CustReturn = BROWSE_ADD Then
   CDate$ = Format$(Now, "dd-mmm-yyyy")
Else
   CDate$ = vxDateFormat("a_cdate")
End If

' Data passed. Put it away
' ------------------------
CursorWait
If CustReturn = BROWSE_ADD Then
   j% = vxAppendBlank()
End If
```

```
Call vxReplString("a_code", (CustCode.text))
Call vxReplString("a_name", (CustName.text))
Call vxReplDate("a_cdate", CDate$)
Call vxReplDate("a_rdate", RDate$)
j% = vxWrite()
j% = vxUnlock()
```

Example 2

```
' using vxReplString to replace date fields
' with ambiguous vxDateString formatted dates
' (see also vxReplDateString)
' ----------------------------------------
EstDueDate.Text = vxDateString("est_due", VX_AMERICAN)
...
...
' Replace due date that is formatted as mm/dd/yy
' ----------------------------------------------
DateStr$ = String$(9,0)
EvStr$ = "DTOS(CTOD(" + (EstDueDate.Text) + "))"
vxEvalString(EvStr$, DateStr$)
vxReplString("est_due", DateStr$)
```

See Also

vxDateFormat, vxDateString, vxReplDateString, vxReplString, vxSetAlias, vxWrite

vxReplDateString

Declaration

Declare Sub vxReplDateString lib "vxbase.dll" (ByVal FieldName As String, ByVal DateString As String)

Purpose

Replace an xBase date field with a string formatted according to country specific conventions. The default format is "mm/dd/yy" (VX_AMERICAN).

Parameters

FieldName is either a string variable or a literal string that contains a valid date field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

DateString is a string representation of a date in the format defined by vxSetDate (default VX_AMERICAN mm/dd/yy). This is the same date style used to control data entry with vxCtlFortmat and used by vxDateString as a return value.

Returns

Nothing. If the database, date, or field name is invalid, no replacement occurs.

Usage

Use to replace date fields that have been entered and verified under the control of vxCtlFormat.

This function is ignored if the file has been opened as Read Only with vxUseDbfRO.

Example

```
Sub Form_Load ()
   MasterDbf = vxUseDbf("myfile.dbf")
   vxSetAlias("master", MasterDbf)
   vxSetDate(VX_AMERICAN)
   DateBox.Text = vxDateString("datefld", VX_AMERICAN)
End Sub

Sub DateBox_GotFocus ()
   j% = vxCtlFormat(8, VX_DATE, 0)
End Sub

Sub SaveButton_Click ()
   Call vxReplDateString("datefld", (DateBox.Text))
   j% = vxWrite()
   j% = vxWriteHdr()
End Sub
```

See Also

vxCtlFormat, vxDateFormat, vxDateString, vxDbfDate, vxReplDate, vxSetAlias, vxSetDate

vxReplDouble

Declaration

Declare Sub vxReplDouble Lib "vxbase.dll" (ByVal FieldName As String, DblAmount As Double)

Purpose

Replace an xBase numeric field with a Visual Basic double value.

Parameters

FieldName is either a string variable or a literal string that contains a valid numeric field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

DblAmount is a Visual Basic double value.

Returns

Nothing.

Usage

Any numeric field that contains decimal positions should be replaced with this command.

All xBase data is stored in string format within the record. The number could also be formatted with Format$(DoubleAmt, "000000.00") (or whatever data picture applies) and replaced within the record with the vxReplString command.

The record buffer is not written to disk until an explicit vxWrite is issued or a command is issued that changes the status of the record pointer (such as vxGo, vxSkip, vxSeek, etc.). In a multiuser environment, always use an explicit vxWrite to ensure the record is available in its changed form as soon as possible.

This function is ignored if the file has been opened as Read Only with vxUseDbfRO.

Example

```
' replace numeric values
' ----------------------
Call vxReplDouble("c_price", Val((AirPrice.text)))

' Vis Basic Val() function always returns a double
' value but is forced into the type of the assigned
' variable if is is other than a double
' ------------------------------------------------
NumVal% = Val((AirTTSN.text))
Call vxReplInteger("c_ttsn", NumVal%)

NumVal& = Val((AirSMOH.text))
Call vxReplLong("c_smoh", NumVal&)

j% = vxWrite()     ' locks and writes
j% = vxUnlock()    ' unlocks
```

See Also

vxDouble, vxReplString, vxSetAlias, vxWrite

vxReplInteger

Declaration

Declare Sub vxReplInteger Lib "vxbase.dll" (ByVal FieldName As String, IntAmount As Integer)

Purpose

Replace an xBase numeric field with a Visual Basic integer value.

Parameters

FieldName is either a string variable or a literal string that contains a valid numeric field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

IntAmount is a Visual Basic integer value.

Returns

Nothing.

Usage

Any numeric field that contains decimal positions should not be replaced with this command. A Visual Basic integer is a whole number with a range of -32,768 to 32,767. If the possible value of your field will exceed this, use vxReplLong or vxReplDouble.

All xBase data is stored in string format within the record. The number could also be formatted with Format$(IntegerAmt, "00000") (or whatever data picture applies) and replaced within the record with the vxReplString command.

The record buffer is not written to disk until an explicit vxWrite is issued or a command is issued that changes the status of the record pointer (such as vxGo, vxSkip, vxSeek, etc.). In a multiuser environment, always use an explicit vxWrite to ensure the record is available in its changed form as soon as possible.

This function is ignored if the file has been opened as Read Only with vxUseDbfRO.

Example

```
' replace numeric values
' ----------------------
Call vxReplDouble("c_price", Val((AirPrice.text)))

' Vis Basic Val() function always returns a double
' value but is forced into the type of the assigned
' variable if is is other than a double
' ------------------------------------------------
NumVal% = Val((AirTTSN.text))
Call vxReplInteger("c_ttsn", NumVal%)

NumVal& = Val((AirSMOH.text))
Call vxReplLong("c_smoh", NumVal&)

j% = vxWrite()     ' locks and writes
j% = vxUnlock()    ' unlocks
```

See Also

vxInteger, vxReplString, vxSetAlias, vxWrite

vxReplLogical

Declaration

Declare Sub vxReplLogical Lib "vxbase.dll" (ByVal FieldName As String, ByVal BoolVal As Integer)

Purpose

Replace an xBase logical field with "T" or "F" depending on a Boolean value.

Parameters

FieldName is either a string variable or a literal string that contains the name of a valid logical type field in the current database. *FieldName* may be qualified with a valid alias name that points to any open database.

BoolVal is either FALSE (zero) or NOT FALSE (not zero). If FALSE, the field will be replaced with "F". If NOT FALSE, the field will be replaced with "T". Any non-zero value will result in a replacement of "T".

Returns

Nothing.

Usage

Primarily used to replace logical fields depending on the value in Visual Basic check boxes or radio buttons (checked = 1, unchecked = 0).

The record buffer is not written to disk until an explicit vxWrite is issued or a command is used that changes the status of the record pointer (such as vxGo, vxSkip, vxSeek, etc.). In a multiuser environment, always use an explicit vxWrite to ensure the record is available in its changed form as soon as possible.

This function is ignored if the file has been opened as Read Only with vxUseDbfRO.

Example

```
' Replace logical fields
' ----------------------
Call vxReplLogical("LogField1", (CheckBox1.Value))
Call vxReplLogical("LogField2", (CheckBox2.Value))
' Note check box value is placed inside parentheses
' to extract the value
' -------------------------------------------------
```

See Also

vxSetAlias, vxTrue

vxReplLong

Declaration

Declare Sub vxReplLong Lib "vxbase.dll" (ByVal FieldName As String, LongInt As Long)

Purpose

Replace an xBase numeric field with a Visual Basic long integer value.

Parameters

FieldName is either a string variable or a literal string that contains a valid numeric field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

LongInt is a Visual Basic long integer value.

Returns

Nothing.

Usage

An xbase numeric field that contains decimal positions should not be replaced with this command.

A Visual Basic long integer is a whole number that has a range of -2,147,483,648 to 2,147,483,647. If the possible value of your field will exceed this, use vxReplDouble.

All xBase data is stored in string format within the record. The number could also be formatted with Format$(LongInt, "0000000") (or whatever data picture applies) and replaced within the record with the vxReplString command.

The record buffer is not written to disk until an explicit vxWrite is issued or a command is issued that changes the status of the record pointer (such as vxGo, vxSkip, vxSeek, etc.). In a multiuser environment, always use an explicit vxWrite to ensure the record is available in its changed form as soon as possible.

This function is ignored if the file has been opened as Read Only with vxUseDbfRO.

Example

```
' replace numeric values
' ----------------------
Call vxReplDouble("c_price", Val((AirPrice.text)))

' Vis Basic Val() function always returns a double
' value but is forced into the type of the assigned
' variable if is is other than a double
' ------------------------------------------------
NumVal% = Val((AirTTSN.text))
Call vxReplInteger("c_ttsn", NumVal%)

NumVal& = Val((AirSMOH.text))
Call vxReplLong("c_smoh", NumVal&)

j% = vxWrite()     ' locks and writes
j% = vxUnlock()    ' unlocks
```

See Also

vxLong, vxReplString, vxSetAlias, vxWrite

vxReplMemo

Declaration

Declare Function vxReplMemo Lib "vxbase.dll" (ByVal FieldName As String, MemoString As String) As Integer

Purpose

Replace a memo with a Visual Basic String.

Parameters

FieldName is either a string variable or a literal string that contains a valid memo field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

MemoString is a Visual Basic string. The memo string is usually read into a text box with vxMemoRead. The user can then edit the string and it can be replaced with vxReplMemo.

Returns

TRUE if the operation was successful; otherwise, FALSE. This is the only vxRepl command that is declared as a function and that returns a value. The memo string replaces a memo in an associated .dbt file rather than a simple record buffer replacement. Always returns FALSE if the associated dbf has been opened as Read Only with vxUseDbfRO.

Usage

Only use if you are gathering memo data in a Visual Basic text box (instead of using vxMemoEdit - which is much more powerful).

If vxSetAnsi(FALSE), the string is converted to the OEM character set before it is written to the memo file.

Example

```
Dim MemoString As String
MemoString = MemoBox.text
j% = vxGo(RecNum&)
If Not vxReplMemo("vxmemo", MemoString) Then
   MsgBox "Error writing memo"
End If
j% = vxUnlock
j% = vxClose()
```

See Also

vxIsMemo, vxMemoClear, vxMemoEdit, vxMemoRead, vxSetAlias

vxReplRecord

Declaration

Declare Sub vxReplRecord lib "vxbase.dll" (RecStruct As Any)

Purpose

Replace the contents of the internal vxBase record buffer for the currently selected dbf with a Visual Basic record structure or string.

Parameters

RecStruct is a defined record structure or a string containing a complete database record.

Returns

Nothing. If the current dbf is invalid, no replacement occurs.

Usage

This function is ignored if the file has been opened as Read Only with vxUseDbfRO.

If using type definitions to describe records and to retrieve their contents (with vxRecord), this function may be used to replace the contents of the record buffer.

All xBase records are kept on disk as fixed length strings containing ASCII data.

WARNING: No data validation of any kind is performed by vxReplRecord. If using this function, all validation must be performed by the programmer before the buffer is passed to vxReplRecord. Always remember that all data is in ASCII format and that the first byte in an xBase record is the deletion flag byte.

Example

```
  ' Record defined in Global Module
  ' ---------------------------------------
  ' define types file record structure for
  ' use in vxform8 and the vxRecord  and
  ' vxReplRecord functions
  ' ---------------------------------------
  Type CatRec
     cDelFlag As String * 1
     Category As String * 3
     CatName As String * 35
  End Type
  ' note that every xbase record structure MUST begin
  ' with a single character deletion flag
  ' ------------------------------------------------

Sub ChangeDeleted ()
  Dim Crec As CatRec

  If vxDeleted() Then
     If vxRecord(Crec) Then
        CatBox.Text = Crec.Category
        CatNameBox.Text = Crec.CatName

        ' we may also replace an entire
        ' record by pointing at a defined
```

```
        ' record or string containing the record
        ' -------------------------------------
        Crec.Category = "zzz" ' if deleted, change key to hi values
        Call vxReplRecord(Crec)
        j% = vxWrite()
        j% = vxWriteHdr()
        StatBox.Text = "Key changed"
     Else
        CatBox.Text = ""
        CatNameBox.Text = ""
        EvalBox.Text = ""
     End If
  End If
End Sub
```

NOTE: If replacing from a string buffer instead of a defined record type, always use BYVAL!

```
  Call vxReplRecord(ByVal Buffer$)
```

See Also

vxField, vxFieldTrim, vxRecord

vxReplString

Declaration

Declare Sub vxChar Lib "vxbase.dll" (ByVal FieldName As String, ByVal FieldString As String)

Purpose

Replace any xBase field with a Visual Basic string.

Parameters

FieldName is either a string variable or a literal string that contains a valid field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

FieldString is a string representation of the data.

Returns

Nothing.

Usage

Normally used to replace the contents of character fields.

All xBase data is stored in string format within the record. You may use any Visual Basic data conversion functions that result in a string to convert data before passing it to vxBase for replacement with the vxReplString command.

The record buffer is not written to disk until an explicit vxWrite is issued or a command is issued that changes the status of the record pointer (such as vxGo, vxSkip, vxSeek, etc.). In a multiuser environment, always use an explicit vxWrite to ensure the record is available in its changed form as soon as possible.

This function is ignored if the file has been opened as Read Only with vxUseDbfRO.

Example

```
' set up date strings in preparation for replace
' -----------------------------------------------
RDate$ = Format$(Now, "dd-mmm-yyyy")
If CustReturn = BROWSE_ADD Then
   CDate$ = Format$(Now, "dd-mmm-yyyy")
Else
   CDate$ = vxDateFormat("a_cdate")
End If

' Data passed. Put it away
' ------------------------
CursorWait
If CustReturn = BROWSE_ADD Then
   j% = vxAppendBlank()
End If

Call vxReplString("a_code", (CustCode.text))
Call vxReplString("a_name", (CustName.text))
Call vxReplDate("a_cdate", CDate$)
Call vxReplDate("a_rdate", RDate$)
```

```
j% = vxWrite()
j% = vxUnlock
```

See Also

vxField, vxSetAlias, vxWrite

vxSeek

Declaration

Declare Function vxSeek Lib "vxbase.dll" (ByVal SearchKey As String) As Integer

Purpose

Find and read the record whose index key matches the defined value.

Parameters

SearchKey is a literal string or string variable that contains the key value you are searching for.

Returns

TRUE if the record was found and FALSE if not.

Usage

This function is a real vxBase workhorse. Most file maintenance functions revolve around whether a particular record has a matching key or not.

If the vxExact flag is set off (the default value), vxSeek will find records with partial key matches. For example, to position the file to the first record whose key field begins with the letter "A", use vxSeek("A"). If there are no records that start with the letter "A", we will get a FALSE return. If the search key value is not as long as the actual key field or expression, TRUE will be returned on a partial key match only if vxExactOff is true (either by explicitly issuing a vxExactOff command or by never issuing a vxExactOn).

If vxExactOn has been issued, the search key must exactly match the key field in length and content before a TRUE is returned.

If the key was found, vxFound will return true any time after the seek (and before the next seek).

If the return is FALSE, the record pointer is undefined, the record buffer contents are also undefined, and vxEof will return TRUE.

If a filter has been set with vxFilter, and the only record that satisifes the seek does not satisfy the filter, the return will be FALSE. If vxExact is OFF, and a partial key is found that satisfies both the seek and the filter, the result will be TRUE.

Multiuser considerations

If vxSetLocks(TRUE), and if vxSeek finds a record, and that record is locked, it will wait (forever) for the record to be released before returning. This is as it should be because if we allow the user to abort a seek with the standard vxBase Retry? query when a locked record is required, the function would have to return a FALSE value. The programmer then couldn't be sure whether the record really wasn't found or if the user aborted because of a lock.

If a record is successfully found, that record is locked if vxSetLocks is TRUE (the default).

vxSetLocks(FALSE) will allow a locked record to be read by another workstation.

Example

```
Sub TypeSave_Click ()

  ' verify something in the field
  ' -----------------------------
  SeekKey$ = TypeCode.text
  If EmptyString(SeekKey$) Then
     MsgBox "Field cannot be empty"
     TypeCode.SetFocus
     j% = vxUnlock()
     Exit Sub
  End If

  ' verify unique key if adding
  ' --------------------------
  If TypeReturn = BROWSE_ADD Then

     If vxSeek(SeekKey$) Then
        MsgBox "Duplicate Key on Add"
        TypeCode.SetFocus
        j% = vxUnlock()
        Exit Sub
     End If
  End If

  ' Data passed. Put it away
  ' ------------------------
  CursorWait
  If TypeReturn = BROWSE_ADD Then
     j% = vxAppendBlank()
  End If

  ' notice the brackets around the control property
  ' below which gets at the data contained therein
  ' ----------------------------------------------
  Call vxReplString("category", (TypeCode.text))
  Call vxReplString("catname", (TypeDesc.text))
  j% = vxWrite()

  ' Update status box
  ' -----------------
  If TypeReturn = BROWSE_ADD Then
     TypeStatus.text = "Record " + LTrim$(Str$(vxRecNo())) + " added"
  Else
     TypeStatus.text = "Record " + LTrim$(Str$(vxRecNo())) + " saved"
  End If

  ' Update Button Status
  ' --------------------
  TypeSave.Enabled = TRUE
  TypeCancel.Enabled = TRUE
  TypeAdd.Enabled = TRUE
  TypeDelete.Enabled = TRUE
  TypeReturn = BROWSE_EDIT
  j% = vxUnlock()            ' ensure database unlocked
  CursorArrow
End Sub
```

See Also

vxDescend, vxExactOff, vxExactOn, vxFound, vxLocate, vxLocateAgain, vxSeekSoft, vxSetLocks

vxSeekFast

Declaration

Declare Function vxSeekFast Lib "vxbase.dll" (ByVal SearchKey As String) As Integer

Purpose

Perform significantly faster seeks in a known read-only environment.

Parameters

SearchKey is a literal string or string variable that contains the key value you are looking for.

Returns

TRUE if the key was found and FALSE if not.

Usage

Use to fill grids, arrays, etc. from a file that you are only going to be reading. vxUseDbfRO MUST be used to open the database. If the file is not opened with vxUseDbfRO, vxSeekFast always returns FALSE. Should only be used in loops where no user interaction is involved.

WARNING!

vxSeekFast performs little error trapping. It is the programmer's responsibility to know the database environment when using vxSeekFast.

The following internal vxBase checks are disabled when using vxSeekFast:

(1) current database selection is assumed to be correct.
(2) correct index is assumed to be selected and current.
(3) there is no checking for changed dbf or ntx buffers (therefore if any changed buffers exist, they are not written and will be LOST. Open the file cleanly as read only and perhaps Lock it to stop any other users from updating it while it is in use).
(4) No locking on the index is performed while reading.
(5) filters are NOT respected.
(6) vxExact status is NOT respected.
(7) vxFound is NOT set.
(8) vxEof is NOT necessarily TRUE if a seek is unsuccessful (unlike vxSeek).
(9) If a seek is NOT successful, the contents of the record buffer are undefined.
(10) Relations set with vxSetRelations are NOT respected by vxSeekFast.
(11) File and record locks placed by other users are NOT respected.

Example

```
' seek speed test
' ---------------
j% = vxUseDbfRO("\ab2\abacus\sam\cluser.dbf")
j% = vxUseNtx("\ab2\abacus\sam\cluser1.ntx")
j% = vxTop()
Key$ = "ABCDEFGHIJKLMNOPRSTUVWXYZ"
Debug.Print Time$

' following loop does 3000 seeks
For k% = 1 To 120
    For m% = 1 To 25
       ky$ = Mid$(Key$, m%, 1)
       If Not vxSeekFast(ky$) Then
          MsgBox "Seek failed"
       End If
    Next
Next
Debug.Print Time$
j% = vxCloseAll()
```

See Also

vxDescend, vxSeek, vxSeekSoft, vxSetLocks

vxSeekSoft

Declaration

Declare Function vxSeekSoft Lib "vxbase.dll" (ByVal SearchKey As String) As Integer

Purpose

Find a record whose key field matches or partially matches the defined search string. If the key is not found, position the record pointer to the next highest key value.

Parameters

SearchKey is a literal string or string variable that contains the key value you are searching for.

Returns

TRUE if a record is read into the buffer. The search key may or may not match the key field depending on the type of find. If no record is found, either partially matched, matched, or the record after, then FALSE is returned.

Usage

vxSeekSoft differs from vxSeek in that a TRUE condition is returned even if the key is not matched and there is a record with a key greater than the search key in the file.

The following conditions apply:

(1) if partial or exact match, vxSeekSoft returns TRUE, vxFound returns TRUE and vxEof returns FALSE.

(2) if not matched, and the record pointer is positioned to the record with a key higher than the search key, vxSeekSoft returns TRUE, vxFound returns FALSE, and vxEof returns FALSE.

(3) if there is no record with a higher key value, vxSeekSoft returns FALSE, vxFound returns FALSE, and vxEof returns TRUE.

This command is especially useful for delimiting a subset of records within a large database. Filters are inherently slow, and an internal routine such as that shown in the example could speed up processing enormously, given a file with a large number of records. There are other ways to accomplish the same result, of course, but this is one of them.

vxExactOn has no effect on vxSeekSoft.

Multiuser Considerations

If a record is found, it is locked if vxSetLocks is TRUE (the default).

Example

```
' finds the range of records in this
' file that all have "ABC" as the first
' part of the key
' ------------------------------------
SrchKey$ = "ABC"

' find the first record
' --------------------
If Not vxSeek(SrchKey$) Then
   Exit Sub
Else
   StartRec& = vxRecNo()

   ' make the last character in the key 1 binary number
   ' greater than the actual key and do a soft seek
   ' ----------------------------------------------
   SoftKey$ = Mid$(SrchKey$,1,2) +
              Chr$(Asc(Mid$(SrchKey$,3,1)) + 1)
   j% = vxSeekSoft(SoftKey$)

   ' As long as vxEof is false, we hit something
   ' ----------------------------------------
   If Not vxEof() Then
      vxSkip(-1)        ' back up one rec to last ABC
      EndRec& = vxRecNo()
   Else
      EndRec& = StartRec&
   End If
   ' now process the range
   ' ---------------------
   RangeProc
End If
```

See Also

vxDescend, vxLocate, vxLocateAgain, vxSeek, vxSetLocks

vxSelectDbf

Declaration

Declare Function vxSelectDbf Lib "vxbase.dll" (ByVal DbfArea As Integer) As Integer

Purpose

Make the open database identified by the passed area handle the current database.

Parameters

DbfArea is a valid area handle returned from vxUseDbf or one of its variants when the file was opened.

Returns

The select area of the previously selected database or zero (0) if there was no previously selected database. If the DbfArea parameter is invalid, subsequent operations will be undefined (like in CRASH).

Usage

Almost every vxBase function works on the selected database only. There is only ONE selected database at any given time, even though many dbf files may be open. Whenever you want to work on a different database, you must select it first.

Each database opened (with vxUseDbf) or selected (with vxSelectDbf) while a Visual Basic form is active is automatically attached to that window. If the user has a number of windows open, and switches between them at will, any vxBase commands that reference a database will automatically select the correct database. To use this automation effectively, you MUST:

(1) select the database as the first command in the FORM_LOAD procedure.

(2) select the database as the first command in the FORM_PAINT procedure.

(3) use vxWindowDereg in the FORM_UNLOAD procedure.

Each of these requirements is discussed in detail in the MultiTasking and MultiUser Considerations section.

If calling sub functions that access databases with vxBase calls, it is always a good idea to reselect the database that should be active immediately after returning from the call. This in effect re-registers the database with the window.

C programmers writing DLLs or VBXs using vxBase calls may turn the auto database selection feature off with vxSetSelect(FALSE).

Example

```
OldDbf% = vxSelectDbf(AirtypesDbf)
CurrRec& = vxRecNo()
If OldDbf% > 0 Then
   j% = vxSelectDbf(OldDbf%)
End If
```

See Also

vxAreaDbf, vxAreaNtx, vxDbfCurrent, vxSelectNtx, vxUseDbf, vxUseDbfAgain, vxUseDbfEX, vxUseDbfRO, vxUseNtx, vxWindowDereg

vxSelectNtx

Declaration

Declare Function vxSelectNtx Lib "vxbase.dll" (ByVal NtxArea As Integer) As Integer

Purpose

Make the open index file identified by the passed area handle the current index for use with the current database.

Parameters

NtxArea is a valid area handle returned by vxUseNtx when the file was opened or by vxAreaNtx.

Returns

The select area of the previously selected index for the current database, or zero (0) if there was no previously selected index. If the *NtxArea* parameter is invalid, subsequent operations will be use record number ordering.

Usage

Whenever an index is opened, it is automatically attached to the current database and selected. The last index opened is therefore the one selected for use. If there is more than one index open, the sequencing may be changed by selecting the new index with this command.

If another database has been selected, and then the dbf that this index belongs to is reselected, it is not necessary to also reselect the index. The index in use will remain the same until another is selected.

Example

```
AirbuyerDbf = vxUseDbf("airbuyer.dbf")
Airbuy1Ntx = vxUseNtx("airbuy1.ntx")
Airbuy2Ntx = vxUseNtx("airbuy2.ntx")

' the current sequence is in airbuy2 order
' ----------------------------------------
DisplayBuyer

' change the sequence
' -------------------
j% = vxSelectNtx(Airbuy1Ntx)
DisplayBuyer

' now select record number order
' ------------------------------
j% = vxNtxDeselect()
DisplayBuyer

' and then put it back the way it was
' -----------------------------------
j% = vxSelectNtx(Airbuy2Ntx)
```

See Also

vxAreaNtx, vxNtxCurrent, vxNtxDeselect, vxSelectDbf, vxUseNtx

vxSetAlias

Declaration

Declare Function vxSetAlias lib "vxbase.dll" (ByVal AliasName As String, ByVal DbfArea As Integer) As Integer

Purpose

Create an alias name for a dbf area in order to qualify field names used in vxBase database functions. If a fieldname is qualified with an alias, the database the alias refers to does not have to be the currently selected dbf.

Parameters

AliasName is a string up to 8 characters long that is used as a field qualifier.

DbfArea is the database select area returned by vxUseDbf or one of its variants when the file is opened. This select area is automatically selected whenever a field name that is qualified with an alias is passed to a vxBase function.

Returns

TRUE if the alias name was set up in the array of alias names.

FALSE if the operation was not successful. FALSE is returned for the following reasons:

(1) DbfArea does not refer to an open database.
(2) AliasName length is zero or greater than 8.
(3) AliasName has already been defined with a different DbfArea.
(4) Maximum number of alias names already defined (96 for all concurrent vxBase tasks).

Usage

It is recommended that all vxBase functions that take a field name as a parameter use alias names to qualify the field. In a multitasking (or multiwindow) environment, reference to a qualified field name will ALWAYS ensure that the correct database is selected no matter what the task or window.

Qualified field names allow the use of multiple databases in the program without requiring a vxSelectDbf prior to accessing the fields from a dbf that is already open.

Field qualifiers are essential if you use vxSetRelations to combine two or more databases into one comprehensive table.

Alias names may be used with the following vxBase functions:

vxChar	vxFieldSize	vxMemoRead	vxReplLong
vxCtlLength	vxFieldTrim	vxPictureImport	vxReplMemo
vxDateFormat	vxFieldType	vxPictureRead	vxReplString
vxDateString	vxIinteger	vxReplDate	vxSum
vxDecimals	vxIsMemo	vxReplDateString	vxTrue
vxDouble	vxLong	vxReplDouble	
vxEmpty	vxMemoClear	vxReplInteger	
vxField	vxMemoEdit	vxReplLogical	

Alias names are NOT used to define join expressions (in vxJoin) or Browse columns with vxTableField. Browse columns that display data joined with vxSetRelation are defined with vxTableFieldExt (for Extended functionality) rather than vxTableField. In vxTableFieldExt the columnar fields or expressions are qualified by passing the actual dbf area to the function rather than by using an alias name. This allows the use of xBase expressions when defining columns whose data resides in the child files of a vxSetRelation.

xBase style alias names and alias names set with the vxSetAlias function are supported within a vxBase xBase expression. The alias names used must be set with vxSetAlias. File alias names are separated from the field reference by "->" (classical xBase syntax) within an xBase expression string. When alias names are used within vxBase functions that refer to field names, a period delimiter is used instead (to conform to Visual Basic syntax).

For example,

```
If Not vxEval("master->country = 'Canada'") Then
   MsgBox "Country does not exist"
Else
   Country$ = vxField("master.country")
End If
```

An alias name construct within a vxBase function call is of the form "Alias.FieldName". A period delimiter is used between the alias and the field name.

Example

```
Sub Form_Load ()
  ' vxSetRelations and vxSetAlias Example
  ' -------------------------------------
  ' We will skip through the Airbuyer file
  ' which has a many to one relationship with
  ' both the aircust.dbf file and the airtypes.dbf
  ' file.

  ' open child files first
  AircustDbf = vxUseDbf("\vb\vxbtest\aircust.dbf")
  Aircust1Ntx = vxUseNtx("\vb\vxbtest\aircust1.ntx")

  AirtypesDbf = vxUseDbf("\vb\vxbtest\airtypes.dbf")
  AirtypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")

  ' open parent file (has many records)
  AirbuyerDbf = vxUseDbf("\vb\vxbtest\airbuyer.dbf")
  Airbuy1Ntx = vxUseNtx("\vb\vxbtest\airbuy1.ntx")

  ' define alias names so we can use field
  ' qualifiers when extracting data
  ' ------------------------------------
  j% = vxSetAlias("buyer", AirbuyerDbf)
  j% = vxSetAlias("customer", AircustDbf)
  j% = vxSetAlias("type", AirtypesDbf)

  ' define relationship to current selection
  ' the 1st param defines the file we are setting
  ' up the relationship to (the child file)
  ' and the second param tells vxbase how to
```

```
  ' construct a key to be used on the current
  ' index in use by the child file
  ' ---------------------------------------
  ErrCode% = FALSE

  ' parent file must be the current selection
  ' when defining the relationship
  If Not vxSetRelation(AircustDbf, "b_code") Then
     MsgBox "1st relation failed"
     ErrCode% = TRUE
  Else
     If Not vxSetRelation(AirtypesDbf, "b_cat") Then
        MsgBox "2nd relation failed"
        ErrCode% = TRUE
     End If
  End If

  If ErrCode% Then
     Unload VYFORM0
  Else
     ' now when we issue vxTop(), the two related
     ' file pointers will move to match the values
     ' in the parent file key fields
     j% = vxTop()
  End If
  VYForm0Display
End Sub

' display procedure uses alias names to extract
' data rather than selecting each database
' ----------------------------------------------
Sub VYForm0Display ()
  If Not vxEof() Then
     Bcode.text = vxField("buyer.b_code")
     Bcat.text = vxField("buyer.b_cat")
     Aname.text = vxField("customer.a_name")
     Catname.text = vxField("type.catname")
   Else
     Bcode.text = ""
     Bcat.text = ""
     Aname.text = ""
     Catname.text = ""
   End If
End Sub
```

See Also

vxSetRelation
and all the field functions listed above

vxSetAnsi

Declaration

Declare Sub vxSetAnsi lib "vxbase.dll" (ByVal OnOrOff As Integer)

Purpose

vxSetAnsi(FALSE) properly handles databases that were created with a DOS based application (such as Clipper). These databases are OEM databases. Characters with diacritical marks in the high end of the OEM character collating sequence are NOT the same as the ANSI characters. It is necessary for vxBase to translate the characters to ANSI (both Windows and vxBase native mode) before they can be used in a vxBase application. They also must be translated back again when they are written.

Parameters

OnOrOff is either TRUE or FALSE. TRUE is the default value.

If TRUE, all data records and index entries are assumed to be in the ANSI character set. No translation takes place.

If FALSE, data records are converted to ANSI from OEM after being read from the file. All internal vxBase operations then take place on the ANSI data. If a record is written, it is converted back to the OEM character set before being written to disk.

If FALSE, index key entries are NOT converted to ANSI. Instead, requests to seek result in the key being translated to OEM before the seek takes place. Similarly, as the index is updated, keys are translated back to OEM before insertion or updating.

All translation between character sets takes place in the background and are transparent to the user.

Returns

Nothing.

Usage

vxSetAnsi would be set to FALSE if you were working with a database that was created with a DOS based application and whose data contains characters from the high end of the character table (i.e., those characters with diacritical marks common to languages other than English).

It should also be set to FALSE if the database with the diacritical characters is going to be used by DOS based applications (e.g., running a Clipper program on a network concurrently with a vxBase program).

The default value of vxSetAnsi is TRUE (no translation takes place). If the database was created and is maintained by vxBase (or DataWorks) and is only going to be used by Windows applications, vxSetAnsi should be TRUE.

If using databases with different native character sets, vxSetAnsi

may be used to toggle translation on and off (as long as the current database has no relations set up to databases with a different native character set).

Example

```
Sub Form_Load ()
   Call vxInit
   Call vxCtlGraySet
   Call vxCtlGraySet
   Call vxSetLanguage(VX_GERMAN)
   Call vxSetLocks(FALSE)
   Call vxSetString(0)
   j% = vxCloseAll()

   ' using OEM databases
   ' -------------------
   Call vxSetAnsi(FALSE)

   ' create descending collating sequence table
   ' ------------------------------------------
   i% = 255
   For j% = 1 To 256
      CharMap(j%) = i%
      i% = i% - 1
   Next j%
   Call vxCollate(CharMap(1))

   ' turn off table usage until required
   ' -----------------------------------
   Call vxSetCollate(FALSE)
End Sub
```

Example ANSI-OEM Conversion

You may wish to convert a vxBase database created with vxSetAnsi(TRUE) into an OEM database for use with DOS based applications such as R&R Report Writer.

The following routine was written by Dr. Alain Spaite of France:

```
Sub AnsiOem_Click ()
Dim FileOrigin As String, FileDest As String
Dim Buffer As String, LenRec As Integer

  FileOrigin = NomPSI(0).Caption ' ANSI file name already opened
  FileDest = NimPSI(1).Caption ' OEM file not yet opened
  Ret% = MsgBox("Please confirm translation", 36)
  If Ret% = 6 Then
     j% = vxSelectDbf(DBFArea(0))
     DBFArea(1) = vxUseDbf(FileDest)
     If DBFArea(1) = FALSE Then
        MsgBox "ERROR: Destination file cannot be opened"
        Exit Sub
     End If
     j% = vxSelectDbf(DBFArea(0))
     j% = vxTop()
     LenRec = vxRecSize()
     Do While Not vxEof()
        Buffer = String$(LenRec, 0)
        j% = vxSelectDbf(DBFArea(0))
        j% = vxRecord(ByVal Buffer)
        Call vxSetAnsi(FALSE)
        j% = vxSelectDbf(DBFArea(1))
        If Not vxAppendBlank() Then
           MsgBox "Error in append"
           Exit Sub
        End If
        Call vxReplRecord(ByVal Buffer)
        j% = vxWrite()
        j% = vxWriteHdr()
        Call vxSetAnsi(TRUE)
        j% = vxSelectDbf(DBFArea(0))
        j% = vxSkip(1)
     Loop
  End If
  j% = vxCloseAll()
End Sub
```

See Also

vxCollate, vxSetCollate

vxSetCollate

Declaration

Declare Sub vxSetCollate lib "vxbase.dll" (ByVal OnOrOff As Integer)

Purpose

Toggle the use of a defined collating sequence table.

Parameters

OnOrOff is either TRUE or FALSE. When a collating sequence table is defined with vxCollate, the value is set to TRUE (its default value is FALSE).

Returns

Nothing.

Usage

Turn an alternate collating sequence table on or off. If no table has been defined with vxCollate, this function has no effect.

You can build a special collating sequence table that only applies to a given index and then turn that table on or off depending on whether the index is in use or not.

Example

```
Sub Form_Load ()
   Call vxInit
   Call vxCtlGraySet
   Call vxCtlGraySet
   Call vxSetLanguage(VX_GERMAN)
   Call vxSetLocks(FALSE)
   Call vxSetString(0)
   j% = vxCloseAll()

   ' using OEM databases
   ' -------------------
   Call vxSetAnsi(FALSE)

   ' create descending collating sequence table
   ' ------------------------------------------
   i% = 255
   For j% = 1 To 256
      CharMap(j%) = i%
      i% = i% - 1
   Next j%
   Call vxCollate(CharMap(1))

   ' turn off table usage until required
   ' -----------------------------------
   Call vxSetCollate(FALSE)
End Sub
```

See Also

vxCollate, vxSetAnsi

vxSetDate

Declaration

```
Declare Sub vxDateString Lib "vxbase.dll" (ByVal DateType As
Integer)
```

Purpose

Set the date display format to be used by xBase date functions (CTOD(), DATE(), and DTOC()) and also by vxBrowse columnar displays of dates and as an input edit mask when editing fields from a browse window.

Parameters

DateType is a country identifier as defined in vxbase.txt. It is one of the following:

```
VX_AMERICAN format mm/dd/yy
VX_ANSI     format yy.mm.dd
VX_BRITISH  format dd/mm/yy
VX_FRENCH   format dd/mm/yy
VX_GERMAN   format dd.mm.yy
VX_ITALIAN  format dd-mm-yy
VX_SPANISH  format dd-mm-yy
```

Returns

Nothing.

Usage

This function is provided to conform with international date conventions. The default value is VX_AMERICAN (MM/DD/YY).

The call should be issued in your initialization procedure.

Warning: If you have used the international section of the WIN.INI file to set Windows dates to a format other than American, beware that the Visual Basic Date$ Function always returns a string in the format "mm-dd-yyyy" and the Visual Basic DateValue Function expects a date in the format defined in the WIN.INI international section. The twain shall not meet. If they do, Visual Basic returns with an "Illegal Function Call" error. If you have set the date to, for example, British format (dd-mmm-yyyy), use code as in the sample below to handle today's date and forget about the Date$ Function:

```
XDate$ = Format$(Now, "dd-mmm-yyyy")
DaysOnFile% = DateValue(XDate$) - DateValue(DateCreate$)) + 1
```

Example

```
Call vxSetDate(VX_BRITISH)
```

See Also

vxCtlFormat, vxDateFormat, vxDateString, vxReplDateString, vxSetErrorCaption, vxSetLanguage

vxSetErrorCaption

Declaration

```
Declare Sub vxSetErrorCaption Lib "vxbase.dll" (ByVal CaptionString
As String)
```

Purpose

Change the caption presented on vxBase error message boxes to whatever the user desires. The default value is "vxBase Error".

Parameters

CaptionString is the new string that will be displayed as the caption in every vxBase error message box.

Returns

Nothing.

Usage

Should be issued in the FORM_LOAD procedure of your startup form.

Example

```
Call vxSetErrorCaption("Real Estate System Error")
```

See Also

vxSetDate, vxSetLanguage

vxSetErrorMethod

Declaration

Declare Sub vxSetErrorMethod lib "vxbase.dll" (VBorVX As Integer)

Purpose

Activate or deactivate the alternate vxBase error method.

Parameters

VBorVX set to TRUE turns on the alternate error method. FALSE sets the error reporting method to the vxBase default (run time errors are reported via message boxes).

If *VBorVX* (the alternate error method) is TRUE, vxBase internal errors MUST be trapped with vxErrorTest.

Returns

Nothing.

Usage

See vxErrorTest for details on using the alternate error method.

Example

```
' test alternate error method VB 1.0
' ---------------------------------
Call vxSetErrorMethod(TRUE)
jj% = vxUseNtx("\vb\vxbtest\testerr.ntx")
If vxErrorTest(vxError) Then
   ProcessError
End If
Call vxSetErrorMethod(FALSE)
```

For an example of use with VB 2.0, see vxErrorTest.

See Also

vxErrorTest

vxSetGauge

Declaration

```
Declare Sub vxSetGauge Lib "vxbase.dll" (ByVal Hwnd As Integer)
```

Purpose

Sets up an alternative method for generating analog or digital gauge information on time consuming indexing and pack procedures. If vxSetMeters is FALSE, and a window handle representing a gauge control (or any other valid window) is passed to vxBase via vxSetGauge, a KeyDown event is triggered for the defined control or window whenever the completion percentage of the active procedure changes.

The KeyCode parameter passed to the KeyDown event procedure contains the percentage complete instead of a key value. This percentage is passed as a NEGATIVE number so the programmer can distinguish between normal key codes and key codes sent by the vxBase procedure.

Parameters

Hwnd is a window handle to the gauge or textbox control. If the form element KeyDown event being triggered is for a Visual Basic control, use vxCtlHwnd to convert the control handle to a window handle.

Returns

Nothing.

Usage

The sample below shows how to use the vxSetGauge function to control the progress of a gauge control (the one included in the Visual Basic Professional kit).

The following functions will generate KeyDown events for the defined control: vxCreateNtx, vxCreateSubNtx, vxPack, and vxReindex (note the absence of vxTestNtx from this list).

With vxCreateNtx and vxCreateSubNtx we are working with a single file at a time so the absolute value of the percentage number generated may be used to directly set the gauge control value.

With vxPack and vxReindex, you must use a method like the one shown in the example below to generate meaningful gauge information. For example, if packing a dbf file with 3 indexes and an attached memo (dbt) file, vxBase will generate percentage progress from 0 to 100 for EACH of the 5 files involved (1 dbf, 3 ntxs, 1 dbt). If you use a method to factor the passed percentage and also track the total progress then the gauge information will be properly presented to the user.

Example

```
' This example uses a general form that contains nothing but
' a gauge control (in the form of a speedometer) to present
' analog progress information to the user. The gauge form is
' shown and set up from a PackFiles_Click menu item on the
' project's main form
```

```
' The gauge form is VXFORMG and the gauge control is generated
' by GAUGE.VBX included with the Visual Basic Professional
' toolkit. The same control is also available from MicroHelp.

' The Gauge control uses default properties EXCEPT for the
' following:
'    Name = Gauge
'    Picture = (Bitmap)  (\vb\bitmqaps\gauge\speedo.bmp)
' Style - 2-'Semi' Needle

' the following Globals are defined to enable proper multifile
' gauging during the pack operation:

Global GaugeFactor As Integer
Global GaugeTotal As Integer

' The user clicks the PackFiles menu item
' ---------------------------------------
Sub PackFiles_Click ()
  vxSetMeters False  ' turn off default meters
  VXFORMG.Show       ' show the form with the gauge on it
  VXFORMG.Caption = "Pack vxUser"  ' set the caption
  DoEvents           ' allows the bitmap to be drawn on the gauge
  vxSetGauge vxCtlHwnd(VXFORMG.Gauge)   ' tell vxBase about the gauge
  GaugeTotal = 0     ' global total progess variable

  ' pack user file
  ' --------------
  j% = vxAreaDbf(DirName + "vxuser.dbf")
  If j% > 0 Then
     MsgBox "vxuser in use!"
  Else
     vxClientDbf = vxUseDbf(DirName + "vxuser.dbf")
     vxWhere = vxUseNtx(DirName + "vxwhere.ntx")
     vxCl1Ntx = vxUseNtx(DirName + "vxuser.ntx")
     vxNorthNtx = vxUseNtx(DirName + "vxnorth.ntx")
     vxPhoneNtx = vxUseNtx(DirName + "vxphone.ntx")
     vxNameNtx = vxUseNtx(DirName + "vxname.ntx")
     vxCompNtx = vxUseNtx(DirName + "vxcomp.ntx")

     ' GaugeFactor is 8 (one for each file above PLUS a dbt
     ' which this file has attached)
     GaugeFactor = 8
     j% = vxSelectDbf(vxClientDbf)
     j% = vxPack(VXFORM1.hWnd)
     j% = vxClose()
  End If

  ' pack fax file
  ' -------------
  VXFORMG.Caption = "Pack vxFax"  ' new caption
  VXFORMG.Gauge.Value = 0         ' reset gauge to 0
  j% = vxAreaDbf(DirName + "vxfax.dbf")
  If j% > 0 Then
     MsgBox "vxfax in use!"
  Else
     GaugeTotal = 0    ' reset total progress var
     GaugeFactor = 3   ' this file has 3 elements (dbf, ntx, dbt)
     vxFaxDbf = vxUseDbf(DirName + "vxfax.dbf")
     vxFaxNtx = vxUseNtx(DirName + "vxfax.ntx")
     j% = vxSelectDbf(vxFaxDbf)
     j% = vxPack(VXFORM1.hWnd)
     j% = vxClose()
  End If
  Unload VXFORMG   ' unload the gauge form
  vxSetGauge 0     ' turn off the gauge setting
  vxSetMeters True ' set default meters back on
```

```
End Sub

' This is ALL of the code associated with the generic gauge form
' ---------------------------------------------------------------
Sub Form_Paint ()
  Call vxFormFrame(VXFORMG.hWnd)
End Sub
```

```
Sub Gauge_KeyDown (KeyCode As Integer, Shift As Integer)
  Dim pValue As Integer

  ' NOTE: ---------------------------------
  ' vxBase sends the KeyCode as a negative value
  ' so the user can distinguish between normal
  ' keydown events and those triggered by the
  ' vxSetGauge function. This is especially
  ' important if the Form KeyPreview property
  ' is set to TRUE. You can then ignore all
  ' key values that are less than 0
  ' ---------------------------------------
  If KeyCode < 0 Then
     If GaugeFactor > 0 Then
        If Abs(KeyCode) = 100 Then
           GaugeTotal = (100 / GaugeFactor) + GaugeTotal
           pValue = 0
        Else
           pValue = Abs(KeyCode) / GaugeFactor
        End If
     Else
        pValue = Abs(KeyCode)
     End If
     Gauge.Value = pValue + GaugeTotal

     ' -----------------------------------------------
     ' you could also set a digital value into a text
     ' box or label here
     ' -----------------------------------------------
  End If
End Sub
```

See Also

vxCreateNtx, vxCreateSubNtx, vxPack, vxReindex, vxSetMeters

vxSetHandles

Declaration

Declare Function vxSetHandles Lib "vxbase.dll" (ByVal NumHandles As Integer) As Integer

Purpose

Change the number of file handles available to a task. By default, the Windows maximum number of file handles available to a task is 20 (15 useable).

Parameters

NumHandles is the number of file handles you wish to allocate to the task.

Returns

An integer that specifies the number of handles actually available to the application (usually 255).

Usage

If you are going to have more than 15 files open simultaneously (DBF, NTX, DBT) in a vxBase application, then you must increase the number of handles using this function.

The call to vxSetHandles should be in your initialization sequence.

SHARE.EXE must be loaded at the workstation.

The maximum number of dbf-ntx-dbt files that can be opened in all concurrent vxBase tasks is about 75. This is a memory constraint.

Example

```
If vxSetHandles(32) < 32 Then
   MsgBox "Not enough handles available"
   End
End If
```

See Also

vxUseDbf, vxUseNtx

vxSetLanguage

Declaration

Declare Sub vxSetLanguage Lib "vxbase.dll" (ByVal LangType As Integer)

Purpose

Change the language in which vxBase displays Browse menus, memo menus, dialog boxes, and error messages. The default is VX_ENGLISH.

Parameters

LangType is one of the following:

VX_ENGLISH defined as Global Const VX_ENGLISH = 0 (default).
VX_FRENCH defined as Global Const VX_FRENCH = 3.
VX_GERMAN defined as Global Const VX_GERMAN = 4.
VX_ITALIAN defined as Global Const VX_ITALIAN = 5.
VX_DUTCH defined as Global Const VX_DUTCH = 6.
VX_SPANISH defined as Global Const VX_SPANISH = 7.

Returns

Nothing.

Usage

Any call to vxSetLanguage sets an initialization file variable. Future tasks will display menus, dialog boxes, and error messages in the language last selected by vxSetLanguage unless vxSetLanguage is called again.

Example

```
Sub Form_Load()
  vxInit
  vxCtlGraySet
  Call vxSetLanguage(VX_FRENCH)
End Sub
```

See Also

vxSetDate, vxSetErrorCaption

vxSetLocks

Declaration

Declare Sub vxSetLocks Lib "vxbase.dll" (ByVal OnOrOff As Integer)

Purpose

Change the record locking mechanism used by all vxBase tasks.

Parameters

OnOrOff is either TRUE or FALSE. The default value is TRUE, which means that every operation that results in a record being read into the vxBase record buffer also locks that record (e.g., vxGo, vxSkip, vxSkip, etc). Traditionally, records in a multiuser system are not locked unless a specific record locking function is called. This is what happens if you set *OnOrOff* to FALSE.

If FALSE, vxBase locking is 100% compatible with Clipper style record locking protocols (Clipper version 5.1 and below).

Returns

Nothing.

Usage

Call vxSetLocks in your initialization routine.

You should also set *OnOrOff* to FALSE when you are testing your application in Visual Basic Design Mode (and remove the command if you really want the default locking mechanism when you create your .EXE). This will ensure that any locks that normally would be in place are not there if your program fails to run to completion and you wish to try it again in the same session. See Visual Basic and VXLOAD.EXE for more information.

Example

```
Sub Form_Load()
  vxInit
  vxCtlGraySet
  Call vxSetLanguage(VX_FRENCH)
  Call vxSetLocks(FALSE)
End Sub
```

See Also

vxBottom, vxBrowse, vxCopy, vxGo, vxIsRecLocked, vxLockDbf, vxLocked, vxLockRecord, vxSeek, vxSeekSoft, vxSkip, vxTop, vxUnlock

vxSetMeters

Declaration

Declare Sub vxSetMeters lib "vxbase.dll" (ByVal OnOrOff As Integer)

Purpose

Set analog meter bars on or off for vxCreateNtx, vxCreateNtx, vxReindex, vxPack, and vxTestNtx.

Parameters

OnOrOff passed as TRUE will turn meter bars on, which allows the user to gauge the progress of the functions listed above. TRUE is the default value.

If *OnOrOff* is passed as FALSE, meter bars will NOT be displayed. It is the programmer's responsibilty to display an hourglass or otherwise inform the user that something is going on when one of the functions that use meter bars is called with vxSetMeters(FALSE).

Returns

Nothing.

Usage

Use to stop meter bar windows from appearing - especially in very small files. They come and go so quickly in small files that they may disconcert the user (because they can't even be read).

If setting FALSE, it is good practice to set the value back to TRUE when the operation is complete.

An alternative gauge setting may be used via vxSetGauge.

Example

```
Call vxSetMeters(FALSE) ' set meter bar off
j% = vxAreaDbf("\vb\vxbtest\airtypes.dbf")
If j% > 0 Then
   MsgBox "airtypes in use!"
Else
   AirtypesDbf = vxUseDbf("\vb\vxbtest\airtypes.dbf")
   AirtypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")
   j% = vxPack(VXFORM1.hWnd)
   j% = vxClose()
End If
Call vxSetMeters(TRUE) ' set meter bars back on
```

See Also

vxCreateNtx, vxCreateSubNtx, vxPack, vxReindex, vxSetGauge, vxTestNtx

vxSetRelation

Declaration

Declare Function vxSetRelation Lib "vxbase.dll" (ByVal ToDbfArea As Integer, ByVal KeyConstruct As String) As Integer

Purpose

Define a relationship between a parent file and a child file based on a key that may be constructed from parent data into an index controlled by the child file.

Parameters

ToDbfArea is the database select area of the child file returned from vxUseDbf or one of its variants when the file was opened.

Note: to clear relations set up to child files without closing the parent, *ToDbfArea* may be passed as a zero.

KeyConstruct is an xBase expression (which may be as simple as a field name) that tells vxBase how to construct a key into the child file index. The maximum length of the *KeyConstruct* string is 511 characters. *KeyConstruct* MUST evaluate as a character string.

Whenever a record pointer is moved in the parent file, a key into the child file index is constructed and a seek is performed into the child file. An exact match or a partial match moves the record pointer in the child file. If the key is not found, the child file record buffer is cleared, the record pointer is positioned to the last record + 1, and vxEof() on the child returns TRUE.

If the child has any relations defined and there was no match, all record buffers in the relational cascade will be cleared and pointers moved as above.

If clearing a set of relationships (i.e., *ToDbfArea* is passed as zero), pass this parameter as 0& (NULL long integer).

Returns

TRUE if the relationship was properly set up. FALSE for any of the following reasons:

(1) no database selected
(2) *ToDbfArea* is the current selection
(3) there are already 8 relations set up for this file
(4) *ToDbfArea* is not open
(5) not enough memory
(6) unable to evaluate *KeyConstruct*
(7) *KeyConstruct* does not evaluate as a character string.

Numeric and date indexes are not supported by vxBase. Indexes on numeric and date fields may be created by using the xBase STR() and DTOS() functions to convert numbers and dates to character strings. The same functions may be used in KeyConstruct to build keys into the child file.

Usage

The parent file must have a many to one or one to one relationship to the child file.

The parent file must be the current selection when vxSetRelation is invoked.

The child file must be open and have an index selected.

Up to eight relations per select area may be defined. Cascading relationships are supported; cyclical relations are not. You may not relate a database either directly or indirectly to itself. vxBase traps direct relationships. Indirect relationships are not trapped and if defined will result in a system hang (an infinite loop will be entered because the record reading routine is recursive).

Example

```
Sub BuyBrowse_Click ()
  ' example of using vxSetRelation
  ' and vxTableFieldExt to produce
  ' a browse table with fields from
  ' multiple databases included on each row
  ' ---------------------------------------

  ' open child files first
  AircustDbf = vxUseDbf("\vb\vxbtest\aircust.dbf")
  Aircust1Ntx = vxUseNtx("\vb\vxbtest\aircust1.ntx")

  AirtypesDbf = vxUseDbf("\vb\vxbtest\airtypes.dbf")
  AirtypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")

  ' open parent file (has many records)
  AirbuyerDbf = vxUseDbf("\vb\vxbtest\airbuyer.dbf")
  Airbuy1Ntx = vxUseNtx("\vb\vxbtest\airbuy1.ntx")

  ' define alias names so we can use field
  ' qualifiers when extracting data
  ' -------------------------------------
  j% = vxSetAlias("buyer", AirbuyerDbf)
  j% = vxSetAlias("customer", AircustDbf)
  j% = vxSetAlias("type", AirtypesDbf)

  ' define relationship to current selection
  ' the 1st param defines the file we setting
  ' up the relationship to (the child file)
  ' and the second param tells vxbase how to
  ' construct a key to be used on the current
  ' index in use on the child file
  ' ---------------------------------------
  ErrCode% = FALSE
  If Not vxSetRelation(AircustDbf, "b_code") Then
     MsgBox "1st relation failed"
     ErrCode% = TRUE
  Else
     If Not vxSetRelation(AirtypesDbf, "b_cat") Then
        MsgBox "2nd relation failed"
        ErrCode% = TRUE
     End If
  End If

  If ErrCode% Then
     j% = vxCloseAll()
```

```
    Exit Sub
  Else
    ' now when we issue vxTop(), the two related
    ' file pointers will move to match the values
    ' in the parent file key fields
    j% = vxTop()
  End If

  ' define the browse table with the extended
  ' vxTableFieldExt function
  ' ----------------------------------------
  Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 6)
  Call vxTableFieldExt(1, "Cust", "b_code", VX_FIELD, 0, AirbuyerDbf)
  Call vxTableFieldExt(2, "Name", "a_name", VX_FIELD, 0, AircustDbf)
  Call vxTableFieldExt(3, "Cat", "b_cat", VX_FIELD, 0, AirbuyerDbf)
  Call vxTableFieldExt(4, "Description", "catname", VX_FIELD, 0,
                       AirtypesDbf)
  Call vxTableFieldExt(5, "Low", "b_low", VX_FIELD, 0, AirbuyerDbf)
  Call vxTableFieldExt(6, "High", "b_high", VX_FIELD, 0, AirbuyerDbf)

  BuyerReturn = 0
  BuyerRec = vxRecNo()

  ' Execute the browse routine (onscreen editor ON)
  ' ----------------------------------------------
  Call vxBrowse(VXFORM1.hWnd, AirbuyerDbf, Airbuy1Ntx, TRUE, FALSE,
                FALSE, BuyerRec, "Buyer Records", BuyerReturn)

  j% = vxCloseAll()
End Sub
```

See Also

vxSetAlias, vxTableFieldExt

vxSetSelect

Declaration

Declare Sub vxSetSelect Lib "vxbase.dll" (ByVal OnOrOff As Integer)

Purpose

Turn off vxBase automatic database selection. Whenever a vxBase database function is called, the last database selected for the current window is used to perform the database function. If there was no database active for the current window, then the last selected database for the current task is automatically selected instead.

Parameters

If *OnOrOff* is TRUE (the default), automatic database selection according to the active window takes place whenever a vxBase function that accesses a database is called. If FALSE, the last selection is used without regard to window or task.

Returns

Nothing.

Usage

Used within Visual Basic sub functions where code attached to a particular form does not come into play. Turning the auto select off ensures there will be no incorrect selection going on that the programmer is not aware of. Care must be taken when using this function because, if the programmer allows the user to open a number of windows each accessing a different database, the selection process may become totally unhinged.

vxSetSelect(FALSE) is normally used by C programmers writing DLLs or VBXs which use vxBase calls. In a DLL, there is commonly no window that can act as controller and even if there were, the programmer knows exactly what database he is using and can reselect at every opportunity to ensure the correct data comes into play.

Example

```
' sub function that does not interfere
' with (or get interfered with by)
' auto selection of database in
' main line
' ------------------------------------
Sub GetMasterNum()
   Call vxSetSelect(FALSE)
   PrevDbf% = vxSelectDbf(MasterDbf)
   MasterNum = vxInteger("masternum")
   Call vxSetSelect(TRUE)
   j% = vxSelectDbf(PrevDbf%)
End Sub
```

See Also

vxSelectDbf

vxSetString

Declaration

"C"
```
void FAR PASCAL vxSetString(int);
```

"Realizer"
```
EXTERNAL "vxbase.dll" PROC vxSetString(INTEGER)
```

"Visual Basic"
```
Declare Sub vxSetString Lib "vxbase.dll" (ByVal StrType As Integer)
```

Purpose

Set the string type returned by all vxBase string functions to ASCIIZ (NULL terminated) or to Visual Basic variable string types.

Parameters

If *StrType* is 0 (zero), the strings returned will be Visual Basic Strings. This is the default value and need not be called if you are writing your vxBase application in Visual Basic.

If *StrType* is 1, then all vxBase functions that return strings will return a pointer to an ASCIIZ string instead of a Visual Basic string.

Returns

Nothing.

Usage

Used to set the string type so that vxBase may be used with languages other than Visual Basic. A call to this function should always be the first call to vxBase from within your application.

If the string type is changed to ASCIIZ with vxSetString(1), all vxBase functions return a pointer to a global string variable contained within vxBase. The returned pointer will always be the same. If a vxBase string function is called immediately after another string function, the contents of the string buffer from the previous function will be overwritten. Always COPY the result of a string function to a variable local to your program.

See Also

vxRecord, vxReplRecord

vxSetupPrinter

Declaration

Declare Sub vxSetupPrinter Lib "vxbase.dll" (ByVal Hwnd As Integer)

Purpose

Access standard Windows printer setup dialog.

Parameters

Hwnd is the hWnd property of an active Visual Basic form. This window acts as parent to the printer select dialog box. It must be enabled.

Returns

Nothing.

Usage

Especially useful for setting form lengths or changing printers (if you have more than one printer port) from within your vxBase application. The user doesn't have to go to the Windows control panel to change printer configuration.

It is not possible to activate another printer with this function if you have more than one printer defined for the same port. See vxPrinterEnum and vxPrinterSelect to change the default printer.

Note: The vxSetupPrinter list box always highlights the current default printer.

Example

```
' PrSetup is a menu item or a button
' ---------------------------------
Sub PrSetup_Click ()
   Call vxSetupPrinter(VXFORM1.hWnd)
End Sub
```

See Also

vxPrinterDefault, vxPrinterEnum, vxPrinterSelect

vxSkip

Declaration

Declare Function vxSkip Lib "vxbase.dll" (ByVal NumRecs As Long) As Integer

Purpose

Skip forwards or backwards the specified number of records.

Parameters

NumRecs is the number of records to skip. If negative, the skip is backwards. If positive, the skip is forwards.

Returns

TRUE if successful and FALSE if not.

Usage

Always used to control record by record processing. If an index is selected, the skip follows the index sequence, otherwise record number sequence is employed.

If a filter is active, vxSkip skips by records that don't pass the filter.

Always use vxEof and vxBof to test whether the end of file has been reached (when skipping forwards) or the beginning of file has been reached (when skipping backwards). Note that if vxEof is true, it will be necessary to position the record to the last record in the file with vxBottom if you wish to have a valid record in the buffer. If vxBof is TRUE, then the record buffer will contain the first record in the file.

Multiuser Considerations

If the skip was successful, the record is locked if vxSetLocks is TRUE.

Example

```
  ' skip forward one record
  ' -----------------------
  Do
     If Not vxSkip(1) Then

        ' if skip error, exit
        ' -------------------
        MsgBox "Error on Skip Next. Try Reindex."
        Exit Sub
     End If

     If vxEof() Then Exit Do
  Loop Until Not vxDeleted()

  ' test for end of file
  ' --------------------
  If vxEof() Then
     Beep
     TypeStatus.text = "End of File!"
     j% = vxBottom()
  Else
     TypeStatus.text = "Skipped to record " +
```

```
                    LTrim$(Str$(vxRecNo()))
  End If
```

See Also

vxBof, vxEof, vxGo, vxSeek, vxSeekSoft, vxSetLocks

vxSum

Declaration

Declare Sub vxSum Lib "vxbase.dll" (ByVal FieldName As String, DblAmount As Double)

Purpose

Sum the contents of a numeric field for all records that satisfy the filter condition (if any).

Parameters

FieldName is either a string variable or a literal string that contains a valid numeric field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

DblAmount is a pre-dimensioned Visual Basic double variable that will hold the result of the procedure.

Returns

No explicit return. The sum is stored in the variable sent in the call to the procedure.

Usage

Extract the sum of the defined field. May be used with a filter to limit the sum to a subset of records in the database.

After the operation has completed, the record pointer is restored to its condition prior to the call.

Multiuser Considerations

The database is locked for the duration of the operation.

Example

```
Dim CalifTotal As Double

' this routine adds up the amounts owing by customers
' in California
' ------------------------------------------------
Call vxFilter("(.NOT. deleted()) .AND. (state = 'CA')")
CalifTotal = 0
j% = vxTop()
Call vxSum("amtowing", CalifTotal)
TotalBox.text = Format$(CalifTotal, "#######0.00")
vxFilterReset
```

See Also

vxFilter, vxSetAlias

vxTableDeclare

Declaration

Declare Sub vxTableDeclare Lib "vxbase.dll" (ByVal ColorRef As Long, BofExpr As Any, EofExpr As Any, ByVal Scope As Integer, ByVal Quick As Integer, ByVal Columns As Integer)

Purpose

Set up a custom table for use by the vxBrowse or vxCtlBrowse functions. The vxTableDeclare command must be followed by vxTableField commands (as many as specified in the Columns parameter) to define the browse table columns.

Parameters

ColorRef is the color to be used for the browse table column heads. There are three Global constants defined in vxbase.txt which may be used with 3d style browse tables: VX_RED, VX_BLUE, and VX_GRAY. VX_WHITE may also be used if the browse table will be displayed in flat style (see vxBrowseSetup).

BofExpr is an xBase expression controlling beginning of file logic (in addition to vxBof() - which is automatic). *BofExpr* is defined "As Any" because in most cases it will be passed as NULL (i.e., ByVal 0&). This parameter is especially useful in limiting the browse table to a subset of the records contained in the file being browsed. For example, suppose you had an accounts receivable subledger with a file key that was composed of two fields, CustCode + InvoiceNo. Now suppose you wish to limit the display to only those subledger records that belonged to customer "ABCDEF". You could either set a filter (which is not very efficient - especially if its a big file -in that a user pressing a page up key when he is at the first record in the file may have to wait a few minutes before vxBase satisfied itself that there were no records above that met the filter) or you can define a *BofExpr* as "CustCode < 'ABCDEF'". If a *BofExpr* is defined, every record must pass the *BofExpr* test. Now when our user is at the first record in the subset and presses the page up key, vxBrowse will skip back one record and test the *BofExpr*. If it fails, vxBrowse goes back to where it was and beeps. The artificial beginning of file set in this manner is evaluated and acted upon virtually instantaneously. A filter would skip backwards until it reached the real beginning of file before determining that there was nothing left to display. Notice that it is not necessary to add the phrase " .OR. BOF()" to the xBase expression because vxBrowse always evaluates the actual BOF() in addition to *BofExpr*.

EofExpr is an xBase expression controlling end of file logic (in addition to vxEof() - which is automatic). It is normally used in conjunction with *BofExpr* to limit the vxBrowse display to a subset of records in the file. In the example shown above, *EofExpr* would be "CustCode > 'ABCDEF'". Now when the user hits the page down key, the first record that has a CustCode greater than "ABCDEF" would effectively stop the display, just as the *BofExpr* does when moving in the opposite direction. Notice that it is not necessary to add the phrase " .OR. EOF()" to the xBase expression because vxBrowse always evaluates the actual EOF() in addition to *EofExpr*. If the scope of the display is

every record in the file, you would pass a NULL value (i.e., ByVal 0&).

Scope is an integer that effectively controls the action vxBrowse takes when the user presses the Home or End keys (or uses the vertical scroll bar thumb to position the file to the top or bottom).

Always use 0 (zero) when the scope you are interested in is every record in the file, or when every record in the file has a unique single element key. If you wish to limit the scope to a subset of records as in the discussion of *BofExpr* and *EofExpr* above, then set *Scope* to the length of the key prefix that is common to the subset. In the example above, the subledger key is composed of two elements - CustCode + InvoiceNo. There are probably many records in the file with the same CustCode but different InvoiceNos and we only want to look at the ones with CustCode = "ABCDEF". This is the common prefix in every key we are interested in; therefore, the *Scope* parameter is set to 6 (the length of the common part of the key).

When a *Scope* other than zero is passed to vxBrowse via the vxTableDeclare command, vxBrowse reacts to a Home request by issuing a vxSeek to the file with a value in the searchkey that is equal to the current key for the length specified by Scope. This will position the record pointer to the first record in our subset (because we get a partial match). When the user requests a positioning to End, the partial key is extracted from the current record ("ABCDEF") and a binary 1 is added to the last character (which makes it a "G"). A vxSeekSoft is then issued which positions the record pointer to the record immediately following our defined subset and vxBrowse then skips back one record and, voila , we are at the end of our subset. Slick! Sure beats filters.

Quick is an integer that specifies the character position of the key vxBrowse uses to construct Quick keys. A zero will turn quick key off (and we don't want to do that on indexed files). If the key to be used for the quick search starts at the first character position of the current index expression, use 1 (which will be most of the time). If we are interested in only a subset of records (as in the example above), then the unique part of the key - InvoiceNo - is what the user should enter to find the record he is looking for. If we defined the quick key as 1 in this case, and the user wanted to find InvoiceNo "1001", then he would have to enter "ABCDEF1" just to position the file to the first invoice that started with a "1". When all of the records in our subset have a common prefix, we use the length of that prefix plus one (in this case 7) to tell vxBrowse that the first 6 positions are always the same so it automatically prepends them to the entered quick key. We don't even have to display the CustCode field in our table and we can find any invoice we want that belongs to this customer by actually entering the invoice number.

Columns is an integer that specifies how many columns our table will have. This number determines the amount of memory to allocate to hold our table definition and it also indicates that this many vxTableField commands will immediately follow. We need 1 vxTableField command for every number passed in this parameter.

Returns

Nothing.

Usage

Some of the concepts discussed above in relation to limiting your displays to a subset of records without having to set a filter may seem confusing at first, but a little study of the example shown below and its effect in the sample program will add clarity to the situation.

When scoping a browse display, the only thing you MUST do is position the record pointer to the first record in the group and then pass that record number to the vxBrowse proc (the StartRec& parameter). See the Scoped Complex Example below.

Declared tables attached to a database are also used by vxBrowse if this file happens to be the object of a relational Join.

vxTableDeclare, vxTableField, vxJoin, and vxBrowse provide you with a browse object that is unparalleled in the xBase world.

NOTE: if your table will contain expressions dependent on the value in fields residing in the current database, you MUST position the record pointer to a valid record with vxTop, vxGo, etc. BEFORE declaring the table and its fields and expressions. vxTableField validates expressions by parsing and executing them to see if they return a valid result.

Simple Example

```
' Open aircraft types file
' ------------------------
  AirtypesDbf = vxUseDbf("\vb\vxbtest\airtypes.dbf")
  If AirtypesDbf = FALSE Then
     MsgBox "Error Opening airtypes.dbf. Aborting."
     Exit Sub
  End If
  AirtypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")
  If AirtypesNtx = FALSE Then
     MsgBox "Error Opening airtypes.ntx. Aborting."
     j% = vxClose()
     Exit Sub
  End If

' Declare types table to get nice headings
' (TableDeclare works on currently selected DBF)
' ---------------------------------------------
  Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 2)
  Call vxTableField(1, "Type", "category", VX_FIELD)
  Call vxTableField(2, "Description", "catname", VX_FIELD)

  ' Open a browse table with full editing capabilities
  ' -------------------------------------------------
  TypeReturn = 0  ' declared as GLOBAL so VXFORM2 can
                  ' interrogate

  ' The menu Form VXFORM1 must be visible because we need a
  ' parent for our browse
  ' -------------------------------------------------------
  If Not VXFORM1.Visible Then VXFORM1.Show

  ' Execute the browse routine (using table declared above)
  ' -------------------------------------------------------
  Call vxBrowse(VXFORM1.hWnd, AirtypesDbf, AirtypesNtx,
                TRUE, TRUE, TRUE, 0, "Aircraft Types",
                TypeReturn)
```

Scoped Complex Example

```
Sub BuyRecs_Click ()

  ' Close states file to free some handles
  ' --------------------------------------
  j% = vxSelectDbf(AirstateDbf)
  j% = vxClose()            ' also does vxTableReset

  ' open airtypes file and buyer file
  ' --------------------------------
  TypesOpen
  BuyerOpen

  j% = vxSelectDbf(AirbuyerDbf)
  j% = vxSelectNtx(Airbuy1Ntx)
  CustKey = CustCode.text

  ' Set up browse table limited to buyer records
  ' that match the CustKey. We do this by sending
  ' the vxTableDeclare proc a beginning of
  ' file expression and an end of file expression.
  ' --------------------------------------------
  BofExpr$ = "b_code < '" + CustKey + "'"
  EofExpr$ = "b_code > '" + CustKey + "'"

  Call vxTableDeclare(VX_RED, ByVal BofExpr$, ByVal
                      EofExpr$, 6, 7, 4)

  ' The vxBrowse object now knows to limit the
  ' records in the table to those that have b_code
  ' values equal to CustKey. We also scope the records
  ' with the "6" following the EofExpr and set the quick
  ' key index to "7". An explanation follows:
  '
  ' The key we are going to use to browse this file is
  ' b_code + b_cat, whose elements are 6 long and 3 long
  ' respectively.  Every record we are interested in has
  ' the same b_code (i.e., they all belong to the same
  ' customer). Setting the scope index to 6 determines
  ' the action to be taken when the HOME or END keys
  ' are depressed. The normal value is 0, which takes
  ' you to the first and last logical records in the
  ' file when HOME or END is hit. If other than
  ' zero, then the HOME key will result in a softseek
  ' on the file to the current key for the length
  ' specified by the scope index. The END key will
  ' softseek to the current key plus 1 and then skip
  ' back one record to position the record pointer to
  ' the last record in the group.
  '
  ' The quick index is set to 7, which is the first
  ' position of the aircraft type code in the key. We
  ' aren't even going to display the b_code for the
  ' buyer records. Setting the quick index to 7 means
  ' that the common part of the key for the group of
  ' records we are interested in (the first 6 which form
  ' the customer code), will be prepended to the
  ' quick keys entered at the keyboard before a seek
  ' is done on the file. Makes sense, huh?

  ' When scoping a file in this fashion, the only thing you
  ' MUST do is position the record pointer to the first
  ' record in the group and then pass that record number
  ' to the vxBrowse proc (the StartRec& parameter).

  Call vxTableField(1, "Type", "b_cat", VX_FIELD)
  Call vxTableField(2, "Description", "b_desc", VX_FIELD)
```

```
Call vxTableField(3, "Low", "b_low", VX_FIELD)
Call vxTableField(4, "High", "b_high", VX_FIELD)
```

```
  ' Because we are interested in only a subset of the
  ' possible records in the buyer file, we have to
  ' determine ourselves whether there are any records in
  ' the file that match the group. If not, we ask the user
  ' if he wants to add a record. vxBrowse normally does
  ' this, but the file must be empty before it asks the
  ' question and sets the return value accordingly.
  ' ------------------------------------------------------
  BuyerRec = 0                      ' global var
  If vxSeek(CustKey) Then
     BuyerRec = vxRecNo()       ' set for browse start rec
     VXFORM3.Hide
     BrowseBuyers
  Else
     j% = MsgBox("No buyer records. Add?", 52)
     If j% = 6 Then
        VXFORM3.Hide
        BuyerReturn = BROWSE_ADD
        VXFORM4.Show
     Else
        j% = vxClose()
        StatesOpen
        j% = vxSelectDbf(AircustDbf)
     End If
  End If
End Sub
```

See Also

vxBrowse, vxBrowseSetup, vxCtlBrowse, vxCtlBrowseMsg, vxJoin, vxTableField, vxTableFieldExt, vxTableReset

vxTableField

Declaration

Declare Sub vxTableField Lib "vxbase.dll" (ByVal ColIndex As Integer, ByVal ColHead As String, ByVal ColExpr As String, ByVal ColType As Integer)

Purpose

Define the contents of table columns declared by vxTableDeclare for use with vxBrowse and vxCtlBrowse.

Parameters

ColIndex is the sequence number of the column from left to right. The first index number is 1 (NOT ZERO).

ColHead is a string representing the column header. The width of the column is calculated by using the greater of the width of the column head and the data represented by the field or expression.

ColExpr is a string defining the data to be displayed. It may be as simple as a field name (not a memo) or a complex xBase expression. Alias field qualifiers are NOT allowed in *ColExpr*. If defining a browse window with relations set up (via vxSetRelation), use vxTableFieldExt to define column expressions that refer to a child dbf.

Arithmetic operations may be performed on groups of fields with the appropriate expression (e.g., "Current + PastDue"). Conditional IIF expressions are also allowed. For example, the expression "IIF(DTOC(RecdDate) = ' / / ', 'No Date ', DTOC(RecdDate))" would display "No Date " if the field was empty or the actual date if it was not empty. Notice in this example that both the true and false results of the IIF expression are character strings and that they both would result in displays that are 8 characters long. Any xBase expression resulting in a character, numeric, or date data type is allowed. Expressions that return logical results or that reference memo fields are not allowed.

NOTE: If the result of an xBase expression is numeric, it must be passed enclosed in the STR() function. This enables vxBrowse to set the column width properly (e.g., STR(CurrAmt+PastDue,11,2)).

ColType defines the type of ColExpr to vxBrowse. Use one of the Global constants VX_FIELD or VX_EXPR defined in vxbase.txt to tell vxBrowse that the data being defined is simply a field or an xBase expression. This speeds processing somewhat because simple fields do not have to go through an evaluation and pseudo compilation.

Returns

Nothing.

Usage

The number of field definitions following the vxTableDeclare statement must conform to the number sent to vxBase in the vxTableDeclare Columns parameter.

If soliciting an XBase expression from the user for use in vxTableField, always test the expression with vxEval before passing it as a parameter in this function.

If onscreen editing is allowed in your vxBrowse table that will use these field definitions, remember that data resulting from an expression (ColType = VX_EXPR) may not be edited in this fashion. You can use this to your advantage by defining columns you do not want the user to edit as VX_EXPR.

Tables declared and then used as a resultant Join window may not have any fields edited onscreen. This is an obvious point because joined relational windows are not explicitly called by a vxBrowse statement anyway.

NOTE: The record buffer must be filled with a valid record from the database that the expression applies to BEFORE vxTableField is called. The database must be open and selected.

The TRIM() Function

If you wish to use the xBase TRIM() function in your table display, the resultant expression length MUST be fixed. For example, suppose your database has fields for LastName and FirstName. Instead of displaying SMITH and JOHN in two separate columns. you wish to display them in one column as "SMITH, JOHN".

The following is ILLEGAL:

```
vxTableField(6,"Name","TRIM(LastName)+', '+TRIM(FirstName)",VX_EXPR)
```

This would result in variable length column widths because every first and last name would result in a different number of characters. Use the SUBSTR() function to properly define a fixed column width while still using TRIM() to concatenate two variable length items:

```
vxTableField(6,"Name","SUBSTR((TRIM(LastName)+', ' + TRIM(FirstName)
+ SPACE(25)),1,25)",VX_EXPR)
```

Ensure that there are enough spaces included after the last TRIMmed element to always result in a length at least as long as the length parameter of the SUBSTR() function (in this case, 25).

Example

SEE THE EXAMPLES IN vxTableDeclare
ON THE PREVIOUS PAGE.

See Also

vxBrowse, vxBrowseSetup, vxCtlBrowse, vxCtlBrowseMsg, vxEval, vxJoin, vxTableDeclare, vxTableFieldExt, vxTableReset

vxTableFieldExt

Declaration

Declare Sub vxTableFieldExt Lib "vxbase.dll" (ByVal ColIndex As Integer, ByVal ColHead As String, ByVal ColExpr As String, ByVal ColType as Integer, ByVal ColWidth As Integer, ByVal DbfArea As Integer)

Purpose

Define the contents of table columns declared by vxTableDeclare for use with vxBrowse and vxCtlBrowse. vxTableFieldExt provides the same functionality as vxTableField and extended functionality with the addition of the ColWidth and DbfArea parameters. vxTableFieldExt MUST be used if defining a browse table that includes data from child files whose relationship to the parent file has been defined by vxSetRelation.

Parameters

Parameters *ColIndex* through *ColType* are as defined in the documentation for vxTableField.

ColWidth allows the programmer to explicitly specify the width of a vxBrowse column in number of characters. Passing a zero width results in vxBrowse using the default calculated width. Numeric fields displayed with the STR() xBase function use the length as defined in the STR() function (i.e., specifying a width via the *ColWidth* parameter is the same as passing a zero width - the default is used instead).

If the width passed through *ColWidth* is insufficient to display the column header, *ColWidth* is again ignored.

Note that the data displayed in the column will NOT be truncated if the column is not wide enough. This parameter is provided mainly for the purpose of fine tuning your browse displays to match the type of data being displayed.

DbfArea is the database select area returned from vxUseDbf or one of its variants when the file is opened. This parameter may be the select area of the parent file or of any child files defined as possessing a relationship to the parent through vxSetRelation.

Returns

Nothing.

Usage

Must always be used if defining a browse table that contains relational data.

Alias field qualifiers are NOT allowed in ColExpr. The expression or field is instead qualified through the DbfArea parameter.

See the vxTableField documentation for a thorough discussion of usage.

Example

```
  ' define the browse table with the extended
  ' vxTableFieldExt function
  ' ----------------------------------------
  Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 6)
  Call vxTableFieldExt(1, "Cust", "b_code", VX_FIELD, 0, AirbuyerDbf)
  Call vxTableFieldExt(2, "Name", "a_name", VX_FIELD, 0, AircustDbf)
  Call vxTableFieldExt(3, "Cat", "b_cat", VX_FIELD, 0, AirbuyerDbf)
  Call vxTableFieldExt(4, "Description", "catname", VX_FIELD, 0,
                       AirtypesDbf)
  Call vxTableFieldExt(5, "Low", "b_low", VX_FIELD, 0, AirbuyerDbf)
  Call vxTableFieldExt(6, "High", "b_high", VX_FIELD, 0, AirbuyerDbf)

  BuyerReturn = 0
  BuyerRec = vxRecNo()

  ' Execute the browse routine (onscreen editor ON)
  ' -----------------------------------------------
  Call vxBrowse(VXFORM1.hWnd, AirbuyerDbf, Airbuy1Ntx, TRUE, FALSE,
                FALSE, BuyerRec, "Buyer Records", BuyerReturn)

  j% = vxCloseAll()
```

See Also

vxBrowse, vxBrowseSetup, vxCtlBrowse, vxCtlBrowseMsg, vxEval, vxJoin, vxTableDeclare, vxTableField, vxTableReset

vxTableReset

Declaration

Declare Sub vxTableReset Lib "vxbase.dll" ()

Purpose

Remove a browse table definition attached to the current vxBase descriptor block and free the associated memory.

Parameters

None.

Returns

Nothing.

Usage

This statement is only necessary if you wish to leave the file open and perhaps define a different table somewhere else in your program. If the file is closed with vxClose or vxCloseAll, the allocated memory is freed automatically.

Example

```
Call vxTableDeclare(VX_RED, ByVal 0&, ByVal 0&, 0, 1, 2)
Call vxTableField(1, "Type", "category", VX_FIELD)
Call vxTableField(2, "Description", "catname", VX_FIELD)
TypeReturn = 0  ' declared as GLOBAL so VXFORM2 can
                ' interrogate
If Not VXFORM1.Visible Then VXFORM1.Show

' Execute the browse routine (using table declared above)
' ------------------------------------------------------
Call vxBrowse(VXFORM1.hWnd, AirtypesDbf, AirtypesNtx,
              TRUE, TRUE, TRUE, 0, "Aircraft Types",
              TypeReturn)
vxTableReset
```

See Also

vxClose, vxCloseAll, vxJoinReset, vxMenuDeclare, vxMenuItem, vxTableDeclare

vxTestNtx

Declaration

Declare Function vxTestNtx "vxbase.dll" (ByVal NtxArea As Integer) As Integer

Purpose

Test the integrity of the defined index.

Parameters

NtxArea is a valid area handle returned by vxUseNtx when the file was opened.

Returns

TRUE if the index passes all tests. Index integrity is most often compromised by the programmer failing to open the index when an update to the dbf file is made that should affect the index in question. FALSE is returned for any of the following reasons:

(1) no index key for an existing dbf record (most common cause).
(2) index key collating sequence is incorrect.
(3) an index key was built from an index expression that no longer matches the expression contained in the index header.
(4) more than one index entry for the same record.
(5) no dbf record for an existing index key.
(6) memory allocation error due to too many records in the database.
(7) index or dbf could not be locked.

Usage

Usually used in a file maintenance function. If the index does not pass, it should be reindexed (or the file should be packed).

Always reselect the dbf following a call to this function.

Note that if a record is appended and not yet written after its fields are filled when this function is called, vxTestNtx will return FALSE. It is good practice to only call this function on a database that has just been opened.

A meter bar window is presented to the user during the operation so the user can gauge the testing progress if vxSetMeters is TRUE (the default).

NOTE: A sub index (created with vxCreateSubNtx) should NOT be tested with this function. It will always fail test (1) abaove.

Multiuser Considerations

The index file and its corresponding dbf are locked for the duration of the operation.

Example

```
If NOT vxTestNtx(NtxArea1) Then
   If Not vxReindex() Then
      MsgBox "Reindex unsuccessful!"
   End If
End If
j% = vxSelectDbf(DbfArea)
```

See Also

vxPack, vxReindex, vxSetMeters

vxTop

Declaration

```
Declare Function vxTop Lib "vxbase.dll" () As Integer
```

Purpose

Position the record pointer to the first record in the current database. If an index is active, this is the first logical record. If there is no index active, the first physical record is retrieved.

Parameters

None.

Returns

TRUE if the operation was successful and FALSE if not. If the file is empty, FALSE will be returned. FALSE will also be returned if the record is locked and the user chose not to retry the operation.

Usage

After opening a file (both dbf and ntx), vxTop is called internally by vxBase to position the record pointer to the first record in the file. When a dbf is opened, this is the first physical record. When an ntx file is opened, this is the first logical record.

If a filter is active, vxTop will attempt to find the first record in the file that satisfies the filter.

Multiuser Considerations

A successful vxTop locks the record if vxSetLocks is TRUE.

Example

```
' test for beginning of file
' --------------------------
If vxBof() Then
   Beep
   TypeStatus.text = "Beginning of File!"
   j% = vxTop()
Else
   TypeStatus.text = "Skipped to record " +
                     LTrim$(Str$(vxRecNo()))
End If
```

See Also

vxBottom, vxSetLocks

vxTrue

Declaration

Declare Function vxTrue Lib "vxbase.dll" (ByVal FieldName As String) As Integer

Purpose

Determine whether a logical field in the current database contains a true or false value.

Parameters

FieldName is either a string variable or a literal string that contains a valid logical field name from the currently selected database. *FieldName* may be qualified with a valid alias name that points to any open database.

Returns

TRUE if the field contains an xBase logical true value (t, T, y, Y) or FALSE if not (either f, F, n, N, or blank).

Usage

Logical fields can easily be used to set form check boxes or radio buttons.

Example

```
' Return from logical field interrogation
' vxTrue() is -1 (TRUE) or 0 (FALSE).
' By using the unary negation operator
' we will transform any -1 values to the
' checkbox value 1, which means "selected"
' ---------------------------------------
CustBuyer.Value = -vxTrue("a_buyer")
CustSeller.Value = -vxTrue("a_seller")
```

See Also

vxField, vxReplLogical, vxSetAlias

vxUnlock

Declaration

Declare Function vxUnlock Lib "vxbase.dll" () As Integer

Purpose

Remove all locks on the currently selected database, including file, record, and index locks.

Parameters

None.

Returns

TRUE if the operation was successful and FALSE if not.

Usage

If vxSetLocks is TRUE, all vxBase record positioning functions automatically lock the record after it has been read into the record buffer. In a multiuser situation, you should get the record, transfer the fields you wish to use to form controls, and then unlock the record to make it and the file available to other users. See the Multiuser Considerations section in this manual for methods that ensure proper record maintenance in a multiuser environment.

If vxSetLocks is FALSE, you should explicitly lock a record immediately prior to updating it and writing it. You should then use vxUnlock to remove the record lock after the write.

Example

```
' vxSetLocks is TRUE
' ------------------
If vxSeek("ABC") Then     ' find the record to update
   RecNum& = vxRecNo()          ' save the record number
   Sig% = vxInteger("CustSig") ' and the signature
   Name.text = vxField("Name")  ' store the form vars
   Status.text = vxfield("Stat")

   ' now unlock the record
   ' ---------------------
   j% = vxUnlock()

   ' now perform the update on the vis basic form
   ' --------------------------------------------
   CustRecordUpdate

   ' now retrieve the record and test if anyone else
   ' has changed it
   ' -----------------------------------------------
   j% = vxGo(RecNum&)
   If Sig% <> vxInteger("CustSig") Then
      MsgBox "Another user beat you to it. Redo!"
   Else
      Call vxReplString("Name", (Name.text))
      Call vxReplString("Stat", (Status.text))
      Call vxReplInteger("CustSig", (Sig% + 1))
   End If
   j% = vxUnlock()
End If
```

See Also

vxIsRecLocked, vxLockDbf, vxLocked, vxLockRecord, vxSetLocks

vxUseDbf

Declaration

```
Declare Function vxUseDbf Lib "vxbase.dll" (ByVal DbfName As String)
As Integer
```

Purpose

Open a database file for reading and writing.

Parameters

DbfName is a either a string variable that contains the name of the file (including an optional path specification) or a literal string. If no file extension is supplied, vxUseDbf defaults to ".dbf".

Returns

FALSE if the open attempt was not successful. Otherwise, an integer greater than zero is returned that defines the select area handle to the file to be used in all subsequent vxBase operations. If the same file has a vxUseDbf command issued more than once without closing, the same integer is returned. If you wish to open another instance of the same file, use vxUseDbfAgain instead.

If an attempt is made to open a vxUseDbf file with vxUseDbfRO without closing the the first instance, the file will not be read only. Only one instance of a vxUseDbf, vxUseDbfEX, or vxUseDbfRO file can be active at a given time. vxUseDbf opens a file for Read/Write access. If the current user has read only access rights, use vxUseDbfRO to open the file. No updating may be performed on a read only file of course.

Usage

The file is opened, selected, and registered with the vxBase Task-Window manager. The select area handle should be retained in a GLOBAL integer for use with that file throughout your application. Use variable names that describe the file.

The first time the file is opened, the result should be tested to ensure that a valid file exists where you think it should be.

After a file is opened, the record pointer is positioned to the first record in the file.

See the discussion under "Multitasking and Multiuser Considerations" for more information on how vxBase controls databases attached to multiple windows.

Example

```
  ' open aircraft file
  ' ------------------
  AircraftDbf = vxUseDbf("\vb\vxbtest\aircraft.dbf")
  If AircraftDbf = FALSE Then
     MsgBox "Error Opening aircraft.dbf. Aborting."
    End
  End If
  Aircraf1Ntx = vxUseNtx("\vb\vxbtest\aircraf1.ntx")
  Aircraf2Ntx = vxUseNtx("\vb\vxbtest\aircraf2.ntx")
```

See Also

vxAreaDbf, vxAreaNtx, vxSelectDbf, vxSetHandles, vxUseDbfAgain, vxUseDbfEX, vxUseDbfRO, vxUseNtx

vxUseDbfAgain

Declaration

Declare Function vxUseDbfAgain lib "vxbase.dll" (ByVal DbfName As String) As Integer

Purpose

Opens a database that has already been opened IN ANOTHER AREA. Any indexes attached to this database with vxUseNtx are also opened in areas separate from any other instances of the same files. The file is opened for reading and writing.

Parameters

DbfName is a either a string variable that contains the name of the file (including an optional path specification) or a literal string. If no file extension is supplied, vxUseDbfAgain defaults to ".dbf".

Returns

FALSE if the open attempt was not successful. Otherwise, an integer greater than zero is returned that defines the select area handle to the file to be used in all subsequent vxBase operations.

Usage

The file is opened, selected, and registered with the vxBase Task-Window manager. The select area handle should be retained in a GLOBAL integer for use with that file throughout your application. Use variable names that describe the file.

After a file is opened, the record pointer is positioned to the first record in the file.

See the discussion under "Multitasking and Multiuser Considerations" for more information on how vxBase controls databases attached to multiple windows.

Updates performed on one instance of an open file will not appear in the second and subsequent instances until the updated or added record is re-read.

Example

```
' open aircraft file
' ------------------
AircraftDbf = vxUseDbf("\vb\vxbtest\aircraft.dbf")
If AircraftDbf = FALSE Then
   MsgBox "Error Opening aircraft.dbf. Aborting."
  End
End If
Aircraf1Ntx = vxUseNtx("\vb\vxbtest\aircraf1.ntx")

' now open another instance with a different
' controlling index
' ----------------------------------------
AirCraftDbf2 = vxUseDbfAgain("\vb\vxbtest\aircraft.dbf")
Aircraf2Ntx = vxUseNtx("\vb\vxbtest\aircraf2.ntx")
```

See Also

vxUseDbf

vxUseDbfEX

Declaration

Declare Function vxUseDbfEX lib "vxbase.dll" (ByVal DbfName As String) As Integer

Purpose

Opens a database for EXCLUSIVE use. If any other user or task is currently using the database, this function will fail (zero is returned as the select area). This function should be used to open a database that will be undergoing critical operations (e.g., vxPack, vxReindex).

Parameters

DbfName is a either a string variable that contains the name of the file (including an optional path specification) or a literal string. If no file extension is supplied, vxUseDbfEX defaults to ".dbf".

Returns

FALSE if the open attempt was not successful. Otherwise, an integer greater than zero is returned that defines the select area handle to the file to be used in all subsequent vxBase operations.

Usage

The file is opened, selected, and registered with the vxBase Task-Window manager. The select area handle should be retained in a GLOBAL integer for use with that file throughout your application. Use variable names that describe the file.

The first time the file is opened, the result should be tested to ensure that a valid file exists where you think it should be.

After a file is opened, the record pointer is positioned to the first record in the file.

See the discussion under "Multitasking and Multiuser Considerations" for more information on how vxBase controls databases attached to multiple windows.

Example

```
' pack file
' ---------
j% = vxAreaDbf("\vb\vxbtest\airtypes.dbf")
If j% = FALSE Then
   AirtypesDbf = vxUseDbfEX("\vb\vxbtest\airtypes.dbf")
   AirTypesNtx = vxUseNtx("\vb\vxbtest\airtypes.ntx")
   j% = vxPack(VXFORM1.hWnd)
   j% = vxClose()
End If
```

See Also

vxUseDbf

vxUseDbfRO

Declaration

Declare Function vxUseDbfRO Lib "vxbase.dll" (ByVal DbfName As String) As Integer

Purpose

Open a database file in Read Only mode.

Parameters

DbfName is a either a string variable that contains the name of the file (including an optional path specification) or a literal string. If no file extension is supplied, vxUseDbfRO defaults to ".dbf".

Returns

FALSE if the open attempt was not successful. Otherwise, an integer greater than zero is returned that defines the select area handle to the file to be used in all subsequent vxBase operations. If the same file has a vxUseDbf or vxUseDbfRO command issued more than once without closing, the same integer is returned. The attributes in effect are those of the first successful open. Only one instance of an open file can be active at a given time in a given task. vxUseDbf opens a file for Read/Write access. If the current user has read only access rights, use vxUseDbfRO to open the file. No updating may be performed on a read only file of course.

Usage

The file is opened, selected, and registered with the vxBase Task-Window manager. The select area handle should be retained in a GLOBAL integer for use with that file throughout your application. Use variable names that describe the file.

The first time the file is opened, the result should be tested to ensure that a valid file exists where you think it should be.

After a file is opened, the record pointer is positioned to the first record in the file.

See the discussion under "Multitasking and Multiuser Considerations" for more information on how vxBase controls databases attached to multiple windows.

Any auxiliary files opened that are attached to a database opened Read Only are also opened Read Only (i.e., index and memo files). No updates are allowed on the database, the indexes, or the memo files. Memos may be edited and exported to ASCII files.

The actual file attributes do not necessarily have to be read only to use this function. If you open a dbf with this function, all write functions are disabled whether the DOS file attributes (or Network rights or flags) are read only or not.

If the file flags are read only, then vxUseDbf will fail where this function will succeed.

Example

```
  ' open aircraft file
  ' ------------------
  AircraftDbf = vxUseDbf("\vb\vxbtest\aircraft.dbf")
  If AircraftDbf = FALSE Then
     AircraftDbf = vxUseDbfRO("\vb\vxbtest\aircraft.dbf")
     If AircraftDbf = FALSE then
           MsgBox "Error Opening aircraft.dbf. Aborting."
        End
     Else
        Aircraf1Ntx = vxUseNtx("\vb\vxbtest\aircraf1.ntx")
        Aircraf2Ntx = vxUseNtx("\vb\vxbtest\aircraf2.ntx")
        Call DisplayOnly
        Exit Sub
     End If
  End If
  Aircraf1Ntx = vxUseNtx("\vb\vxbtest\aircraf1.ntx")
  Aircraf2Ntx = vxUseNtx("\vb\vxbtest\aircraf2.ntx")
  Call UpdateRoutine
```

See Also

vxAreaDbf, vxAreaNtx, vxSelectDbf, vxSetHandles, vxUseDbf, vxUseNtx

vxUseNtx

Declaration

Declare Function vxUseNtx Lib "vxbase.dll" (ByVal NtxName As String) As Integer

Purpose

Open an index file and attach it to the currently selected database.

Parameters

NtxName is a either a string variable that contains the name of the file (including an optional path specification) or a literal string. If no file extension is supplied, vxUseNtx defaults to ".ntx".

Returns

FALSE if the file could not be opened. If the open is successful, an index area handle is returned that should be retained for all subsequent operations using this index file.

Usage

The defined index file must belong to the database that is currently selected. **The last opened index file becomes the selected index** until changed with vxSelectNtx or vxNtxDeselect.

The select area handle should be retained in a GLOBAL integer for use with that file throughout your application. Use variable names that describe the file.

A successful open positions the record pointer to the first record (pointed to by the first index entry in this file) in the database. Filters and relations are respected.

Example

```
' open aircraft file
' -------------------
  AircraftDbf = vxUseDbf("\vb\vxbtest\aircraft.dbf")
  If AircraftDbf = FALSE Then
     MsgBox "Error Opening aircraft.dbf. Aborting."
    End
  End If
  Aircraf1Ntx = vxUseNtx("\vb\vxbtest\aircraf1.ntx")
  Aircraf2Ntx = vxUseNtx("\vb\vxbtest\aircraf2.ntx")
```

See Also

vxAreaNtx, vxNtxDeselect, vxSelectNtx, vxSetHandles, vxTestNtx

vxWindowDereg

Declaration

Declare Sub vxWindowDereg Lib "vxbase.dll" (ByVal Hwnd As Integer)

Purpose

Deregister a database select area from the vxBase Task-Window manager and also release vxCtlFormat memory (if any).

Parameters

Hwnd is the hWnd property of the Visual Basic form that you are deregistering.

Returns

Nothing.

Usage

The vxBase Task-Window manager can keep track of up to 96 task-window-select area combinations. vxWindowDereg is used to ensure that all references to this database in this window are removed when the form is closed. Always issue this command in your FORM_UNLOAD procedure after closing any databases. It will ensure that the Task manager does not overflow.

See the discussion under "Multitasking and Multiuser Considerations" for more information.

vxBase can also keep format information for up to 256 active text boxes. If vxCtlFormat is used for this purpose, vxWindowDereg must always be called in the Form Unload procedure to release format memory and to clear references to active text boxes for re-use.

Example

```
If CustReturn <> BROWSE_USER Then
   j% = vxSelectDbf(vxClientDbf)
   j% = vxClose()
   j% = vxSelectDbf(vxStateDbf)
   j% = vxClose()
   vxWindowDereg (VXFORM3.hWnd)
   VXFORM1.OpenVx.Enabled = TRUE
   VXFORM1.PackFiles.Enabled = TRUE
   VXFORM1.TestMEmo.Enabled = TRUE
End If
```

See Also

vxCtlFormat, vxSelectDbf

vxWrite

Declaration

```
Declare Function vxWrite Lib "vxbase.dll" () As Integer
```

Purpose

Write the contents of the current record buffer to disk.

Parameters

None.

Returns

TRUE if the operation was successful or FALSE if not. Always returns FALSE if the associated dbf has been opened as Read Only with vxUseDbfRO.

Usage

Record fields are changed with the vxReplxxx functions. These changes occur internally in a record memory buffer. The contents of that buffer are written out whenever another record operation occurs (such as vxGo, vxSkip, vxTop, etc.) or when the file is closed.

vxWrite explicitly writes the record as soon as the replacements are complete. In a multiuser environment, always use vxWrite to write the record contents as soon as possible after changes have been made, and then unlock the file to make the record available to other users.

Warning: DO NOT use this function as an all purpose buffer clearing function. If the record pointer is in an undefined state, a blank record will be appended to your database.

Example

```
If CustReturn = BROWSE_ADD Then
   j% = vxAppendBlank()
Else
   vxGo(SaveRec&)
End If

Call vxReplString("a_code", (CustCode.text))
Call vxReplString("a_name", (CustName.text))
Call vxReplDate("a_cdate", CDate$)
Call vxReplDate("a_rdate", RDate$)
j% = vxWrite()
j% = vxUnlock()
```

See Also

vxAppendBlank, vxSetLocks, vxWriteHdr

vxWriteHdr

Declaration

Declare Function vxWriteHdr Lib "vxbase.dll" () As Integer

Purpose

Explicitly write XBase header information.

Parameters

None.

Returns

TRUE if the operation was successful and FALSE if not.

Usage

vxBase only updates the XBase file header information when the file is closed. This information includes a date and time stamp, and the number of records in the database.

If you are using a concurrent third party XBase file management program to monitor the results of your vxBase application, it will probably not recognize the addition of records to the database because it relies on the header record count to determine the database extent.

Use vxWriteHdr after every record update or addition to make vxBase 100% compatible with other XBase file programs.

Example

```
' vxSetLocks is TRUE
' ------------------
If vxSeek("ABC") Then     ' find the record to update
   RecNum& = vxRecNo()         ' save the record number
   Sig% = vxInteger("CustSig") ' and the signature
   Name.text = vxField("Name)  ' store the form vars
   Status.text = vxfield("Stat")

   ' now unlock the record
   ' ---------------------
   j% = vxUnlock()

   ' now perform the update on the vis basic form
   ' --------------------------------------------
   CustRecordUpdate

   ' now retrieve the record and test if anyone else
   ' has changed it
   ' ----------------------------------------------
   j% = vxGo(RecNum&)
   If Sig% <> vxInteger("CustSig") Then
      MsgBox "Another user beat you to it. Redo!"
   Else
      Call vxReplString("Name", (Name.text))
      Call vxReplString("Stat", (Status.text))
      Call vxReplInteger("CustSig", (Sig% + 1))
   End If
   j% = vxUnlock()
   j% = vxWriteHdr()
End If
```

See Also

vxZap

Declaration

```
Declare Function vxZap Lib "vxbase.dll" () As Integer
```

Purpose

Physically delete all of the records in the file.

Parameters

None.

Returns

TRUE if the operation was successful and FALSE if not. Always returns FALSE if the associated dbf has been opened as Read Only with vxUseDbfRO.

Usage

Would normally be used to delete the contents of a permanent batch file after the batch records have been appended to a master file.

Ensure that all index files associated with the file are open. The file is reindexed after the vxZap (i.e., the index files are cleaned out as well).

Multiuser Considerations

The file and all of its index files are locked for the duration of the operation.

Example

```
TrMasterDbf% = vxUseDbf("Transmas.dbf")
TrMasterNtx% = vxUseNtx("Transmas.ntx")
j% = vxSelectDbf(TrMasterDbf%)
vxAppendFrom("Transbat.dbf")
j% = vxClose()      ' close master file

' reopen transaction batch because the From
' file is closed by vxAppendFrom
' ----------------------------------------
TransDbf% = vxUseDbf("Transbat.dbf")
TransNtx% = vxUseNtx("Transbat.ntx")
j% = vxDbfSelect(TransDbf%)
j% = vxZap()        ' clear the batch
```

See Also

vxDeleteRange, vxPack

Error Messages

150 Arithmetic overflow
Numeric field not long enough to hold the result of an xBase arithmetic expression.

200 Unable to evaluate expression.
One or more errors found in xBase expression string. Unable to continue.

230 Logical values ynYNtfTF only allowed.
vxBrowse onscreen edit of logical field. Characters shown above are the only ones allowed.

290 Cannot access specified print driver.
Error occurred in vxSetupPrinter dialog. The print driver selected cannot be found.

302 Close active join links before closing this window.
Windows created with the JOIN browse menu item must be closed before the main window.

305 Active browse tables. vxCloseAll illegal.
All active browse tables for this task must be closed before the files may be closed.

340 Create database error
Either a DOS error (e.g., out of disk space) or an error in the field structure passed to the vxCreateDbf function.

420 Dialog box in use!
An attempt has been made to activate a dialog box that is currently in use (perhaps by another task). vxBase dialog boxes may be used by only one task at a time.

501 Cannot edit result of expression.
Attempt made to onscreen edit a vxBrowse column that is the result of an xBase expression rather than a field.

502 Cannot edit memo with onscreen editor
Attempt made to onscreen edit a memo field displayed with vxBrowse. Use vxMemoEdit or vxMemoRead/vxReplMemo instead.

504 Field Edit not allowed on joined windows.
vxBrowse onscreen edit of fields only allowed on the parent window originating the first join link.

505 Only one active field edit allowed.
Finish the first onscreen edit before proceeding to another.

530 Error in Printer setup.
An error occurred in accessing the selected Windows print driver during vxSetupPrinter.

550 Expression length error
vxBase could not evaluate an expression because the return length is zero.

555 Expression too long
xBase expression length is limited to 127 characters.

560 Expression type check error
Mismatched data type within xBase expression. Comparisons require same data type on either side of the relational operator. Functions require set data type (e.g., SUBSTR() takes a character value).

600 File creation error
DOS could not create the file. Either disk space problem or network security violation or not enough handles allocated with vxSetHandles.

605 File name or path invalid.
Import memo file function failed because the entered file could not be found.

610 File lock error
DOS could not lock the requested record bytes.

620 File open error
File may not exist or not enough handles allocated with vxSetHandles or a network security violation or SHARE /f parameter not large enough.

625 File positioning error
DOS could not position its read/write pointer to a valid location in the file. Record number may be larger than the number of records in the file.

640 File read error
DOS could not read the file. Either a disk error occurred or there was a network security violation or SHARE violation (SHARE is not loaded a the workstation).

670 File unlock error
DOS could not unlock the requested record bytes. DOS internal error.

680 File write error
DOS could not write to the file. Either a disk problem, out of space, or a network security violation.

690 Field replace type mismatch
The data type of the replacement data does not match the defined field type.

694 From file cannot be found
vxAppendFrom could not find the file it is supposed to append data from.

900 Incomplete expression
xBase expression is incomplete or unsupported.

904 Index close error
DOS could not close the index file. Could be due to an invalid index select area.

908 Index corrupted
vxBase detected a corrupted index. Use vxReindex to repair.

914 Out of memory in index sort
The file is too large to index with vxBase. Try cutting down the number of key elements.

918 Internal index invalid key pointer
Destroy the index and try vxReindex.

920 Internal index block size error
Destroy the index and try vxReindex.

922 Internal index node position error
Destroy the index and try vxReindex.

924 Internal index read error
Destroy the index and try vxReindex.

926 Internal index root seek error
Destroy the index and try vxReindex.

928 Internal index skip error
Destroy the index and try vxReindex.

930 Internal index leaf size error
Destroy the index and try vxReindex.

932 Invalid record number. Record not written!
The contents of the record buffer cannot be written to the specified location because that record does not exist. New records require vxAppendBlank to create an empty record.

934 File has zero length
DOS directory entry error. File was not closed properly.

935 Invalid column index
vxTableField column index is out of the range specified by vxTableDeclare.

936 Invalid date
Date passed back to vxBase cannot be translated into an xBase date, or a date entered into a vxBrowse onscreen edit of a date field or text box formatted with vxCtlFormat was invalid.

938 Invalid Dbf Area
Attempt was made to access a select area that does not contain a valid database.

940 Invalid number of delimiters
xBase expression evaluation error. Mismatched parentheses or quotation marks.

942 Invalid field number
A relative field access cannot be completed because the field number is greater than vxFieldCount.

944 Invalid field name
The referenced field could not be found in the current select area. If multiple windows are present on the screen, or multiple select areas are being used in one form's logic, vxBase may have changed the select area in response to a user transparent message passed to Visual Basic from Windows. If the field name is spelled correctly, try inserting an explicit vxSelectDbf in front of the offending field reference.

946 Invalid Index Area
The index select area passed to a vxBase function is invalid.

948 Invalid record length
Maximum record length is 32666.

952 Invalid memo file name
A .dbt file could not be found that matches the name of the .dbf.

953 Invalid menu index
vxMenuItem index parameter is out of the range specified by vxMenuDeclare.

954 Invalid menu level
vxMenuItem level param must be greater than or equal to zero.

955 Invalid menu type
vxMenuItem type parameter must be VX_RETURN (0), VX_MENUHEAD (1), or VX_SEPBAR (2).

956 Invalid number in expression
An xBase expression element contains an invalid number (e.g., negative number as index to SUBSTR()) OR there are illegal mixed data types on either side of an xBase operator (e.g., trying to add a character field to a date field).

960 Invalid operator
An xBase expression contains an unrecognized operator, or an operator that does not work on the data types involved (e.g., 5 $ NumField is invalid because the "is contained in" operator only works on character fields).

964 Incorrect number of parameters
An xBase function was passed the wrong number of parameters (e.g., LEFT(FieldName) is invalid because a number must follow the FieldName).

970 Invalid record number on vxGo
The record number is not within the file range (negative or greater than that returned by vxBottom).

975 Invalid registration number
The shareware license number entered is invalid. Try again or call to confirm the number issued.

980 Invalid seek. No index open.
vxSeek only allowed on indexed files.

984 Invalid select area
The select area sepcified does not contain a valid database descriptor block.

990 Invalid date format expression
An xBase expression evaluation could not decipher the date format contained within the expression.

1000 No records found that match join key.
User message. The record pointer in the vxBrowse master window was moved to a record that has no matching records in the joined file. Information only.

1100 Key does not match expression
The key in the index does not match the expression that the index was built with. If a file structure is modified, and the type of a field that is an element in a key expression changes, then the index becomes invalid. Rebuild the index with vxCreateNtx.

1110 Key max length exceeded (338 chars)
The maximum length of a key is 338 characters.

1120 Key must evaluate as a string
vxBase keys must evaluate as strings. Use the STR() function to convert numeric values to strings, and the DTOS() function to convert dates to strings.

1290 Maximum submenus exceeded
Only 64 VX_MENUHEAD types are allowed within a single defined menu structure.

1300 Windows memory allocation error
Windows could not allocate the requested memory. Buy more.

1305 Memory deallocation error
A memory handle has become invalid for some reason. A UAE will usually occur before we ever get this message (Windows 3.0).

1307 Memo max length (65534) exceeded
The maximum length of a memo is 65534 (unsigned integer max - 1).

1310 Memo type not supported
Only Clipper or dBase III type memo files are supported by vxBase.

1315 Memo write error
DOS error or network security violation.

1317 Menu structure error
A vxMenuItem level parameter refers to a menu index that is not defined as VX_MENUHEAD.

1320 String delimiter missing
xBase expression string delimiters are double or single quotes. They must be matched.

1347 Must declare menu before vxMenuItem
vxMenuDeclare must be used to allocate memory for the upcoming menu structure defined by a series of vxMenuItem commands.

1350 Must declare table before vxTableField
vxTableDeclare must be issued on the selected database before the fields in the table can be defined.

1400 Expression must evaluate as Character string.
Key expessions passed to vxCreateNtx must evaluate as character strings. See error code 1120 above.

1406 Expression must evaluate as logical TRUE or FALSE.
xBase expressions to be used as filters must evaluate as logical results.

1409 No database currently selected
Field references and file statistical references require an open, selected database. The file you think is selected may have been attached to another task (e.g., a print job) or another window in the same task. Issue another vxSelectDbf immediately following calls to subroutines that may deselect the database from the current window or task or immediately following a Print.EndDoc statement.

1412 No browse handles available
Up to eight vxBrowse windows may be open at one time (for all active tasks).

1415 No index active
Attempt made to perform an index function (e.g., vxReindex) while no index was active.

1418 No records found that pass filter.
Information only. vxBrowse reports that there are no records that qualify for display given the current filter.

1420 No matching fields
An attempt was made to vxAppendFrom a file that contains no matching field names in the currently selected database.

1422 Cannot allocate memory for memo edit
Not enough memory to edit the memo. At least 65535 bytes must be

free.

1424 Edit control out of space.
A memo was read into an edit control (vxMemoEdit) that is not large enough to hold the memo.

1430 Search string not found.
Information only. A search string entered in the Query Search vxBrowse menu item could not be found.

1436 Not a memo field!
An attempt was made to pass a field name to a memo function that is not a memo type.

1442 Not an NTX format index
Invalid index format. NDX and CDX files are not supported in this version of vxBase.

1448 Not an xBase database
The requested file open was not performed because the database header was not a dBase III or Clipper type file. dBase II and dBase IV file formats are not supported.

1450 Number of columns required
vxTableDeclare requires the number of columns that will be contained in the vxBrowse table.

1454 Numbers only allowed.
Onscreen edit of a numeric field error message if a non-numeric character was entered.

1500 Out of memory
Self explanatory.

1602 Internal Pack Error
We may have run out of disk space in the pack. The database could be corrupted.

1620 Parentheses mismatched in expression
Self expanatory.

1630 Sign must be in first position
If a sign is entered into a numeric field with the vxBrowse onscreen editor or in a vxCtlFormat numeric text box, it must precede the numeric portion of the field.

1650 Printer error!
Self explanatory.

1900 Record skip error
Should never happen. If it does, the database is probably corrupted.

1950 Too many params in expression
The xBase expression is too complex to evaluate. Simplify and try

again or call for help.

1990 Task closure sequence error.

An attempt has been made to close a vxBase task that controls the shared memory among all concurrently running vxBase tasks. This task was the first task among the set of vxBase tasks currently running and as such it must be the last one to be closed becuase it controls all of the memory allocted to database functions by vxBase. See vxInit and vxDeallocate for more information.

2002 Task list overflow!

The vxBase Task manager may contain up to 96 task-window-select area entries. Use vxWindowDereg in your FORM_UNLOAD procedure to deregister windows when they are closed.

2004 Too many decimals

vxBrowse onscreen edit of a numeric field or vxCtlFormat numeric text box found too many decimal points (e.g., 34.56.7).

2010 Too many signs

vxBrowse onscreen edit of a numeric field or vxCtlFormat numeric text box found too many signs (e.g., -34.56-)

2050 Type mismatch

Attempt to compare apples to oranges in an xBase expression or a wrong data type was used as a parameter to an xBase function.

2100 Unsupported function in expression

vxBase does support this function. Request its addition via Compuserve if you absolutely must have it.

2120 User aborted print

Information only. User cancelled print job (either memo print or vxBrowse print).

Software License Agreement

vxBase is not and never has been public domain software, nor is it free software.

The software product and user's manual are copyrighted and all rights are reserved by vxBase Systems.

Non-licensed users are granted a limited license to use vxBase on a thirty day trial basis for the purpose of determining whether vxBase is suitable for their needs. The use of vxBase beyond the thirty day trial period requires registration and the issuing of a license number. The use of unlicensed copies of vxBase beyond the thirty day evaluation period by any person, business, corporation, government agency, or any other entity is strictly prohibited.

A license permits a user to use vxBase on any single computer, or, in a LAN environment, one copy may be installed on one server and this copy may be shared among the workstations connected to the LAN that are under the same roof as the LAN server.

Licensed users may use the program on different computers, but may not use the program on more than one computer at the same time.

No one may modify or patch the vxBase files in any way, including but not limited to decompiling, disassembling, or otherwise reverse engineering the program.

A limited license is granted to copy and distribute vxBase for the trial use of others, subject to the above limitations, and to those below:

(1) vxBase must be copied in unmodified form, complete with the file containing this license information.

(2) vxBase may not be distributed in licensed form to any person using an application written in Visual Basic that makes use of the vxBase function calls. It MUST be distributed as an unlicensed copy except as noted under Developer Distribution License below.

(3) No fee, charge, or other compensation may be requested or accepted for distributing vxBase, except as follows:

(a) operators of electronic bulletin board systems may make vxBase available for downloading. A time-dependent charge for the use of the bulletin board is permitted so long as there is no specific charge for the download of any vxBase files.

(b) vendors of Shareware may distribute vxBase, subject to the above conditions, and may charge a disk duplication and handling fee, not to exceed ten dollars.

Developer Distribution License

A Developer Distribution License may be granted to developers in consideration of the payment of $195.00 U.S. (less the shareware registration fee if one has been paid). This license allows the developer to distribute a special run-time only version of vxbase.dll to end users for their use with the developer's application. The run-time version of vxbase.dll plus a printed copy of the vxBase manual will be forwarded to any developer who pays the Developer Distribution License fee. The run-time version of vxbase.dll may be distributed in unlimited quantities by the developer who has been granted such a license. The run-time version of vxbase.dll is free of all nagware and has been disabled for use in Visual Basic Design mode.

Limited Warranty

vxBase Systems guarantees your satisfaction with this product for a period of sixty days from the date of original purchase. If you are dissatisfied with vxBase within that time period, return the package in saleable condition to vxBase Systems for a full refund.

vxBase Systems warrants that all disks provided are free from defects in material and workmanship, assuming normal use, for a period of sixty days from the date of purchase.

vxBase Systems warrants that vxBase will perform in substantial compliance with the documentation supplied with the software product. If a significant defect in the product is found, the Purchaser may return the product for a refund. In no event will such a refund exceed the purchase price of the product.

The product and all updates are provided on an "as is" basis without warranty of any kind, express or implied, except as stated above including, but not limited to the implied warranties of merchantibility or fitness for a particular purpose. The entire risk as to the selection, quality, results, and performance of the product is with the Licensee. Should the product prove defective, then the Licensee (and not vxBase Systems or its dealer) assumes all liability and expense incurred as a result thereof. Some jurisdictions do not allow the exclusion of certain implied warranties so in such jurisdictions, the above exclusion of implied warranties may not apply to you. The limited warranty gives you specific legal rights. You may also have other rights which vary from jurisdiction to jurisdiction.

vxBase Systems shall have no liability or responsibility to you or to any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by the product or your use, misuse or inability to use the product, including but not limited to, any interruption of service, loss of business, anticipatory or actual profits or consequential damages resulting from the use, misuse or inability to use the product.

vxBase Systems does not warrant that the functions contained in the product or updates will meet your requirements.

Use of this product for any period of time constitutes your acceptance of this agreement and subjects you to its contents.

vxBase Ordering Information

You may order vxBase and DataWorks directly from vxBase Systems via check, money order, Visa, or Mastercard. You may also order vxBase and Dataworks from Public (software) Library with your Mastercard, Visa, AmEx, or Discover card by calling 1-800-242-4PsL (from overseas: 713-524-6394) or by FAX to 713-524-6398 or by CompuServe to 71355,470. These numbers are for ordering from PsL only. vxBase Systems can NOT be reached at those numbers. Shareware disks may be ordered from PsL with item number 7614/3946. vxBase and DataWorks may also be registered through PsL by quoting item numbers 10473 for vxBase or 10472 for DataWorks. Shipping will originate from vxBase.

For technical support, shipping status, direct licenses, and developer distribution licenses, contact vxBase Systems at the address, phone, or FAX listed below.

When ordering from vxBase Systems, please provide the following information:

(1) Company
(2) Programmer name
(3) Street address
(4) City and State/Province
(5) Country
(6) Zip/Postal Code
(7) Telephone (overseas include country code)
(8) FAX number
(9) if paying by credit card,
credit card type (Visa or Mastercard ONLY)
Credit card number
Expiration date
Signature
(10) Disk preference (3-1/2" or 5-1/4")

Pricing:

vxBase	$59.95 U.S.
vxBase Manual	$20.00 U.S.
DataWorks	$49.95 U.S.
Developer's Kit*	$195.00 U.S.

Air Mail Shipping (No Manual):

U.S./Canada Shipping	$10.00 U.S.
Overseas Shipping	$15.00 U.S.

Air Mail Shipping (Developer's Kit or vxBase with Manual):

U.S./Canada Shipping	$17.00 U.S.
Overseas Shipping	$28.00 U.S.

Canadian orders add 7% GST (GST# R133247296)

*The Developer's Kit includes vxBase, vxBase RunTime Unlimited Distribution, vxBase manual, and Dataworks. If you have already registered vxBase and/or DataWorks, deduct your registration fee(s) from

$195.00 and include your license number(s).

For Priority Courier Service, substitute the following for Shipping and Handling:

	W/Manual	No Manual
Canada and the U.S.	$ 37.00	$ 31.00
United Kingdom	70.00	55.00
France	70.00	55.00
Western Europe	77.00	60.00
Australia	88.00	70.00
Pacific Rim	88.00	70.00
Central America	88.00	70.00
Eastern Europe	117.00	95.00
Middle East	117.00	95.00
India & Japan	117.00	95.00
South America	124.00	100.00
Korea	124.00	100.00
Africa/Indian Ocean	134.00	110.00
Mainland Asia	134.00	110.00

Mail, phone, or fax your order to:

vxBase Systems
#488, 9707 - 110 Street
Edmonton, Alberta, Canada
T5K 2L9
Phone (403) 488-8100
Fax (403) 488-8150
BBS (403) 488-8365

Purchase orders accepted with prior approval.