vxBase

Visual Basic
xBase Functions for Windows 3.x

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 $49.95 plus $5.00 Shipping and Handling 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 $295.00 (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.

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

Manuals

A printed copy of this manual is available for $15.00 (including shipping and handling) to users who do not purchase the developer distribution license.

Release Notes

Release 1.00 November 10, 1991
Release 1.01 November 19, 1991
Release 1.02 December 1, 1991
Release 1.03 January 22, 1992
Release 1.04 February 20, 1992
Release 1.05 March 20, 1992

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

Terry Orletsky
Compuserve I.D. 70524, 3723

vxBase (512523 Alberta Ltd.)
#200, 10310 - 176 Street
Edmonton, Alberta, Canada
T5S 1L3

Phone (403) 489-5994
Fax (403) 486-4335

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, and to Willy Koch of Geneva for his French translation.

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 5.11, and Windows 3.0 in Standard 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 UAE report sent by fax or to my Compuserve address. Please describe your operating environment in detail and include a listing of your config.sys file.

vxBase Table of Contents

Contents (continued)

Contents (continued)

Installation

vxBase is distributed on a single diskette or on bulletin boards 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 2 more ZIP files. These ZIP files are:

vxbdll.zip	the vxBase DLL
vxbtest.zip	sample source code, sample database, and vxbase.txt

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.

Unzip vxbdll.zip and place the resulting file (VXBASE.DLL) 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 the *Sample Application* section below.

Release History

vxBase 1.00
November 10, 1991 original release

vxBase 1.01
November 19, 1991

String routines handled by Jonathan Zuck's vbpoint.dll replaced by Microsoft VBAPI functions and installation procedure changed accordingly.

New functions:

vxCtlGrayReset	Resets disabled color to system standard.
vxCtlGraySet	Sets disabled color to dark gray.
vxCtlLength	Set data entry length for a control.
vxCtlStyle	Set recessed, raised, creased contol style.
vxFormFrame	Draw a frame around the form.

Most of the new functions have been added to enhance the appearance of your VB application. VGA/SVGA users can now give their forms a metallic, three dimensional look. The sample application forms have been redesigned using the new functions.

Anomalies Discovered in Version 1.00

Two problems surfaced in Release 1.00. The first resulted in UAEs when running Windows in 386-enhanced mode. This was a memory deallocation error. Apparently 386-enhanced protected mode is more protected than Standard protected mode. Go figure.

The second problem was the inadvertent deletion of a stock object in the browse function. Problems caused by this bug were intermittent.

vxBase 1.02
December 1, 1991

New Functions:

vxBrowseCase	Set browse case to upper or lower as default.
vxMemoRead	Creates Vis Basic string out of a memo either unformatted for multiline text boxes or formatted for printer output.
vxReplMemo	Replace memo with a Visual Basic string. You may now edit memos in your own text boxes.
vxSetErrorCaption	Set your own error message box caption if you want to replace the default "vxBase Error".
vxSetupPrinter	Allows direct access to Windows Print Manager setup routines. Especially useful for changing form sizes from your app instead of having to bring up the control panel.
vxWindowDereg	Deregister a select area attached to a window. This is a new function that helps implement the vxBase multitasking scheme.

Important changes implemented include:

(1) up to 8 browse windows may be active at one time. Reports reflecting your browse table layout may now be printed from the vxBrowse Utilities menu.

(2) restriction on multiple instances removed.

(3) select areas are now attached to windows so you can have multiple forms displaying data from several databases.

(4) indexing buffer space increased to handle very large files with complex key structures.

(5) number of open index areas increased to 32 from 20.

(6) numer of open dbf files increased to 24 from 20.

Please read the new section on Multitasking to get some idea of the way select areas are now attached to windows.

Anomalies Discovered in Version 1.01

UAEs browse windows with complex filter expressions. Stack overflow problem corrected.

vxBase 1.03

January 22, 1992 1.03 Maintenance Release

Indexing problem corrected on very large files (over 200,000 records).

Scrolling and quick key positioning in Browse windows corrected for files with more than 10,000 records.

Field structure added to "About File" dialog box in vxBrowse.

Enhanced mode UAE corrected in vxClose() (inconsistent memory deallocation).

C run time library bug on close index file corrected if more than 20 files open in a given task.

vxBase 1.04

February 20, 1992

New functions:

vxBrowsePos	Allows setting initial position and size of a browse window.
vxDateString	Returns a date string conforming to a selected international convention.
vxDeallocate	Releases memory supplied by vxBase to VB.EXE when in design mode. See explanantion below.
vxDescend	Creates a search key to seek records in indexes built with the new xBase DESCEND() function added to the xBase Function list that vxBase supports.
vxMenuDeclare	Allocate memory for a user defined menu to be placed on an upcoming browse window.
vxMenuItem	Define a menu structure for a user defined menu to be placed on a browse window.

vxRecord	Returns an entire record as a string or a defined data structure.
vxSetDate	Sets the international date format to be used in vxBrowse displays of date fields, on-screen editing of date fields, and the format of dates returned by xBase functions CTOD(), DTOC() and DATE().
vxSetHandles	Allows opening more than 20 files per task.
vxSetString	All vxBase functions that return VB variable length strings may be set to return standard ASCIIZ strings instead.

Corrections and changes made to Version 1.03

vxReplLong function corrected.

vxBrowse menu item Query Find Next corrected to retain previous search string.

vxBrowse quick key with dashes and ampersands corrected.

VB.EXE memory growth problem in VB Design Mode corrected. Repeated test runs of a vxBase application in VB Design Mode resulted in the non-discardable memory portion of VB.EXE growing arithmetically by at least 130k with each repeat until VB ran out of memory. The problem has been corrected with the new vxDeallocate function (which only works when in design mode).

Scrolling problem in vxBrowse corrected if start record was not the first in the file or the first in a defined subset.

Intermittent stack error in vxMemoRead and vxMemoRepl corrected.

If a join is defined in a browse window, the joined window is automatically displayed without the user having to select the Join menu item from the browse menu.

vxBase 1.05

March 20, 1992

New Functions:

vxDbfDate	Extract the date of last database access.
vxInit	**Required** first call to vxBase to register task for multitasking management.
vxIsRecLocked	Determine if current record is locked.
vxJoinNoAuto	Turn off automatic joining of linked windows.
vxReplLogical	Replace logical field by passing boolean values.
vxSetLanguage	French language support added.
vxTestNtx	Test the integrity of an open index.
vxUseDbfRO	Open a database file READ ONLY.

Corrections and changes made to Version 1.04

vxBrowsePos vertical size increased slightly to always display a full record at the bottom of the browse window.

QuickKey in vxBrowse corrected. Previous quick key entered was being cleared if window was redrawn.

About vxBase menu item removed from Browse menus.

vxFile intermittent UAE corrected. Call to DOS function now initializes segment registers.

vxReplLong corrected if number being saved was greater than short integer max.

vxDeallocate changed to report a true or false condition depending on whether or not the current task can be terminated without interfering with other vxBase tasks concurrently running.

vxTop() and vxBof() corrected when used with filters and the first record or records in the file do not pass the filter.

Cleaner redrawing of vxBrowse menus if modifications made.

F3 accelerator key attached to Find Again item in browse search

menu.

vxPack NOW PACKS attached memo files!

vxCopy now does NOT copy deleted records. It also uses the active index to copy records so the new database is in sorted order. vxCopy now copies (and compresses) memo files attached to the From database.

Meter bar added to index routines. vxPack meter bar changed. Also includes meter bar on memo file compression.

Creating a vxBase Application

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.

And that's it. Have fun, eh?

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.

Borland C++

vxBase functions may also be called from Borland C++. Contact the author for a sample application written in Borland C++ 2.0 Turbo mode. 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. 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. This means that 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.

Expressions

This section and those following on 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.

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, and in statements used to define the contents of a display column when defining a table.

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
< 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

Note the leading and trailing periods that delimit a logical operator.

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**
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
IIF(Logical, True Result, False Result)	Logical if
LEFT(Char_Value, Length)	Leftmost n characters
MONTH(Date_Value)	Numeric month
RECNO()	Record number
RIGHT(Char_Value, Length)	Rightmost n characters
SOUNDEX(Char_Value)	String to phonetic complement
STR(Number, Len, Dec)	Numeric value to string
SUBSTR(Char_Value, Start, Length)	Substring
TIME()	System time as string
UPPER(Char_Value)	Convert to uppercase
VAL(Char_Value)	Character to numeric value
YEAR(Date_Value)	Numeric Year

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

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)

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")

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 AA1111.

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 6 character string

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.

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 Application

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 application included with vxBase is intended to be used as a template 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 thier 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 b_desc field is redundant. It comes from the Airtypes file when the buy category is selected. This data redundancy is necessary to simplify and clarify the browse tables we will construct from this file.

To effectively use vxBrowse, you may find it necessary to make more data redundant than you have been used to in a Clipper or similar environment. At some point in the development of vxBase, we will make a SET RELATION clone command to eliminate this requirement.

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.

The Forms

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.

The Link Menu

These two 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.
Open the 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.

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. The sample application has many examples to guide you towards proper data validation. It takes a little more work in Windows. 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.

Parents for vxBase Windows

Windows created with vxBase functions (vxBrowse and vxMemoEdit) absolutely require an active Visual Basic form to act as parent. Their 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. In any single instance, any given database is opened only once, no matter how many vxUseDbf commands you issue for it. If the user wants two forms up editing or displaying records in the same file, he can run a second instance of your program. The second instance gets its own select area. Always remember that you don't know what the user is liable to do, so disable those functions that could compromise your current position.

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 the same price (and 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 mutiple 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.

To run more than one vxBase task at the same time, 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 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.

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.

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 field reference. 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.

vxBase allows up to 24 databases to be open simultaneously (with up to 32 indexes) in total (not per task). Windows only 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. 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.

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, as opposed to Clipper 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 vxBase, 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 and the vxBase procedure is that with vxBase 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. 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.

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 <u>not required</u> 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).

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. 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
        vxReplString("Field1","New Field")
  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 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.

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.

Warning: If the sending and receiving files have memo fields with the same name, the receiving file will get the memo reference but **no memo** will be transferred.

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.

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%)
     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 with a range from 1 through 24, which is the maximum number of open dbf files that can be assigned to all vxBase tasks on a single workstation.

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.

Usage

Use the returned value as input to any other vxBase function that may require a number instead of a file name to identify the requested dbf-ntx set or to check on whether the file is open or not. Use GLOBAL variable names that uniquely identify each dbf in your application. "CustomerFile" is a better name than "DbfFile".

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 at this workstation
' ----------------------------------------
NamesDbf% = vxAreaDbf("c:\database\names.dbf")
If NamesDbf% = FALSE Then
   vxUseDbf("c:\database\names.dbf")
```

```
    j% = vxPack()
    j% = vxClose()
  Else
    MsgBox "File is open. Function aborted."
  End If
```

See Also

vxPack
vxSelectDbf
vxUseDbf

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 with a range from 1 through the number of open files your system can support for a single task. vxBase allows up to 24 open dbf areas. The number of index files attached to these is limited by the operating environment (maximum 32).

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 as input to any other vxBase function that may require a number instead of a file name to identify the requested ntx or to check on whether the file is open or not. Use GLOBAL variable names that uniquely identify each ntx in your application. "CustIndex" is a better name than "NtxFile".

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
vxSelectNtx
vxUseNtx

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 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, 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

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

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.

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 upon which the table is drawn.

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. 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 is intended to be the primary tool 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).

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.

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.**

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).

Note: If a filter is set in a large file, a vxBrowse table will take some time to initialize. The number of records in the file that pass the filter must be ascertained to determine the vertical scroll bar extent. The entire file must be read.

Other Menu Items

The browse table always has a menu bar. 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 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.

Multiuser Considerations

No records are locked by vxBrowse unless and until the user makes a record selection. If other users lock records that will be displayed by the browse, the browse will wait until the file is free. 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.

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
vxJoin
vxMenuDeclare
vxMenuItem
vxSetDate
vxTableDeclare
vxTableField

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 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 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

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 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).

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 with an intervening close of the
' file, the window will retain its last position and size.
```

See Also

```
vxBrowse
vxBrowseCase
```

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 a valid field name from the currently selected 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 *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.

Example

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

See Also

vxCloseAll
vxCloseNtx
vxDbfDate
vxJoinReset
vxTableReset
vxUseDbf
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% = vxNtxClose(MastIndex%)
MastIndex2% = vxUseNtx("Transfi2.ntx")
```

See Also

vxClose
vxCloseAll
vxNtxDeselect

vxCopy

Declaration

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

Purpose

Make an exact 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 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.

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.

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
vxPack

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%)
      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 Fld*nn* 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. 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.

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

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

NewNtxNameis 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).

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
vxDescend
vxNtxExpr

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
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.

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
vxCtlStyle
vxFormFrame

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 a valid field name from the currently selected 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.

Restriction

This function is restricted to Visual Basic users only.

Example

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

vxCtlStyle

Declaration
Declare Sub vxCtlStyle Lib "vxbase.dll" (ControlName As Any, 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 gives the control a recessed look. VX_RAISE raises the control away from the form, and VX_CREASE gives the control a creased border.

Returns
Nothing.

Usage
Gives your application a metallic, three-dimensional look (on VGA/SVGA monitors only). 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. Its 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
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.

Parameters

DateField is a valid date field name from the currently selected 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
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 a valid date field name from the currently selected 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. 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

vxDateFormat
vxSetDate

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 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. 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.

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 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.

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

vxDeleted

Declaration

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

Purpose

Determine whether a record 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.

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

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
   vxClose()
   Kill "Transfil.dbf"
Else
   j% = vxDeleteRange(OldLastRec& + 1, NewLastRec&)
   j% = vxClose()
End If
```

See Also

vxDeleteRec
vxZap

vxDeleteRec

Declaration

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

Purpose

Logically delete the current record.

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.

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

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 a valid numeric field name from the currently selected 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
vxInteger
vxLong
vxReplDouble
vxReplString

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 a valid field name from the currently selected 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

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

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 a valid field name from the currently selected 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.

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
vxInteger
vxLong
vxRecord
vxReplString
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 a valid field name from the currently selected 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

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 a valid field name from the currently selected 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

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.

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.

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. vxBase evaluates logical operators in the following sequence: .AND., .OR., and then .NOT. 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)" because the ".and." is evaluated first. 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)").

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
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.

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.

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
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

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.

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.

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
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 a valid field name from the currently selected 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.

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

vxIsMemo

Declaration

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

Purpose

Determine whether there is a memo attached to the defined field.

Parameters

FieldName is a valid memo field name from the currently selected database.

Returns

TRUE if there is a memo in the .dbt file, and FALSE if not.

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

vxMemoEdit

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

```
' only update if record is locked
' -------------------------------
j% = vxLockRecord()
If vxIsRecLocked() Then
   vxReplString("field1", (Text1.text))
   j% = vxWrite()
End If
```

See Also

vxLockDbf
vxLocked
vxLockRecord
vxUnLock

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.

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
   vxJoinReset
   AddProcedure
End If
```

See Also

vxClose
vxCloseAll
vxJoin

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
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
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 because vxBase automatically locks records as soon as they are read. However, long-winded intervening code between a record lock and processing an update to that record could necessitate an explicit lock on a record if another user could have removed all locks on the database during the processing of the intervening code.

Example

```
j% = vxGo(SaveRec&)
DoABunchOfStuff
If vxLockRecord() Then
   UpdateProc
Else
   MsgBox "Sorry. Can't lock the record"
End If
```

See Also

vxIsRecLocked
vxLockDbf
vxLocked
vxUnlock

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 a valid field name from the currently selected 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.

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

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).

FieldName is a valid memo field name from the currently selected 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).

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).

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.

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+. 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

vxCopy
vxIsMemo
vxPack

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 a valid memo field name from the currently selected 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 32k. You will require 64k (unformatted) or 96k (formatted) in text buffers to retrieve a string of this length. If you have monster memos, beware.

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.

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
vxMemoEdit
vxReplMemo

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.

Example

```
' Declare and build a user menu
' ------------------------------
Call vxMenuDeclare(19)     ' 19 items in the menu
```

See Also

vxBrowse
vxMenuItem

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

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
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
vxUseNtx

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
vxRecSize

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.

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 .$$$ is created. Every record is analyzed for the presence of valid memo block references and, if any are found, the old memo is copied to the new .$$$ 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.

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
vxDeleteRec

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

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.

It MUST be used in this fashion when editing a memo.

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).

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.

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.

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
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 files belonging to the current database are open.

Always use vxAreaDbf to ensure that the file is not open in **any active task**.

A meter bar window is presented that charts the progress of the reindex routine for each file being reindexed.

Multiuser Considerations

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

Example

```
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()
```

See Also

vxAreaDbf
vxPack
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 a valid date field name from the currently selected 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()
```

See Also

vxDateFormat
vxReplString
vxWrite

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 a valid numeric field name from the currently selected 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, "#####0.00") (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
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 a valid numeric field name from the currently selected 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, "####0") (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
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 the name of a valid logical type field in the current 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).

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

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 a valid numeric field name from the currently selected 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 2,147,438,647. 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, "#######0") (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
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 a valid memo field name from the currently selected 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).

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
vxMemoEdit
vxMemoRead

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 a valid field name from the currently selected 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
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 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.

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
vxSeekSoft
```

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.

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
vxSeek

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 when the file was opened or by vxAreaDbf.

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.

Example

```
OldDbf% = vxSelectDbf(AirtypesDbf)
CurrRec& = vxRecNo()
If OldDbf% > 0 Then
   j% = vxSelectDbf(OldDbf%)
End If
```

See Also

vxAreaDbf
vxAreaNtx
vxSelectNtx
vxUseDbf
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 undefined (like in CRASH).

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
vxNtxDeselect

```
vxSelectDbf
vxUseNtx
```

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 date convention set with vxSetDate is a SYSTEM wide function. All vxBase concurrent tasks use the same date format once this procedure is called. 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

vxDateFormat
vxDateString
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. Note that this is a SYSTEM wide command which affects every active vxBase task.

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

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.

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.

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.

Returns

Nothing.

Usage

Any call to vxSetLanguage sets a system wide global constant. All tasks currently running vxBase will switch languages. Future tasks will display menus, dialog boxes, and error messages in the language last selected by vxSetLanguage.

Example

```
Sub Form_Load()
  vxInit
  vxCtlGraySet
  Call vxSetLanguage(VX_FRENCH)
End Sub
```

See Also

vxSetDate
vxSetErrorCaption

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 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.

WARNING: This function sets a GLOBAL vxBase flag. Visual Basic applications written with vxBase will not run concurrently with applications written in other languages if vxSetString(1) is called. Visual Basic applications will terminate with a "Bad DLL Calling Convention" message.

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

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 if you have more than one printer defined for the same port. The user will still have to go to the control panel to effect this change.

Example

```
' PrSetup is a menu item or a button
' ---------------------------------
Sub PrSetup_Click ()
   Call vxSetupPrinter(VXFORM1.hWnd)
End Sub
```

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 numer 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.

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

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 a valid numeric field name from the currently selected 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

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 function. 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: VX_RED, VX_BLUE, and VX_GRAY.

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
vxJoin
vxTableField
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.

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. 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.

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 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.

Example

SEE THE EXAMPLES IN vxTableDeclare
ON THE PREVIOUS PAGE.

See Also

vxBrowse
vxJoin
vxTableDeclare
vxTableReset

vxTableReset

Declaration

```
Declare Sub vxTableReset Lib "vxbase.dll" ()
```

Purpose

Remove a 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

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) memory allocation error due to too many records in the database. Only worry about this one if your database record count exceeds 400,000 records.
(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 index key for an existing dbf record (most common cause).
(6) no dbf record for an existing index key.

Usage

Usually used in a file maintenance function. If the index does not pass, it should be reindexed (or the file should be packed).

Example

```
  If NOT vxTestNtx(NtxArea1) Then
     If Not vxReindex() Then
        MsgBox "Reindex unsuccessful!"
     End If
  End If
```

See Also

vxPack
vxReindex

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, the record buffer content is undefined until an explicit record operation is performed. This is usually vxTop.

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.

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

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 a valid logical field name from the currently selected 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

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

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.

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

vxIsRecLocked
vxLockDbf

```
vxLocked
vxLockRecord
```

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 is returned between 1 and 24 that defines the select area handle to the file 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 an attempt is made to open a vxUseDbf file with vxUseDbfRO with closing the the first instance, the file will not be read only. Only one instance of an open 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 contents of the record buffer are undefined. An explicit record positioning command must be issued to fill the record buffer (such as vxTop).

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
vxUseDbfRO
vxUseNtx
```

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, vxUseDbf defaults to ".dbf".

Returns

FALSE if the open attempt was not successful. Otherwise, an integer is returned between 1 and 24 that defines the select area handle to the file 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 contents of the record buffer are undefined. An explicit record positioning command must be issued to fill the record buffer (such as vxTop).

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, vxUseDbf 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.

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.

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.

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

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()
```

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.

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 probably does not exist. Is the path specification correct?

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.

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.

912 Index key does not exist
An index key for a specified record does not exist. The index is probably out of date and may be corrected with vxReindex.

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 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())

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 data 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 (100 chars)
The maximum length of a key is 100 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 (32767) exceeded
The maximum length of a memo is 32767 (signed integer max).

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 32767 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, 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 found too many decimal points (e.g., 34.56.7).

2010 Too many signs
vxBrowse onscreen edit of a numeric field 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 (512523 Alberta Ltd.).

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 $295.00 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 (512523 Alberta Ltd.) 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 Comsoft Inc. for a full refund.

vxBase (512523 Alberta Ltd.) 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 (512523 Alberta Ltd.) 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 (512523 Alberta Ltd.) 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 (512523 Alberta Ltd.) 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 (512523 Alberta Ltd.) 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 (512523 Alberta Ltd.) 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 71335,470. These numbers are for ordering from PsL only. vxBase (512523 Alberta Ltd.) 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 (512523 Alberta Ltd.).

For technical support, shipping status, direct licenses, and developer distribution licenses, contact vxBase (512523 Alberta Ltd.) at the address, phone, or FAX listed below.

When ordering from vxBase (512523 Alberta Ltd.), 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	$49.95 U.S.
DataWorks	$49.95 U.S.
vxBase Manual	$15.00 U.S.
Developer Pack*	$295.00 U.S.
U.S./Canada Shipping	$5.00 U.S.
Overseas Shipping	$15.00 U.S.

Canadian orders add 7% GST (GST# R1010183020)

*The Developer Pack 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 $295.00 and include your license number(s).

Mail, phone, or fax your order to:
vxBase (512523 Alberta Ltd.)
#200, 10310 - 176 Street
Edmonton, Alberta, Canada
T5S 1L3
Phone (403) 489-5994

Fax (403) 486-4335

Purchase orders accepted with prior approval.

Developer's Special Offer
Purchase the vxBase Developer's Kit for just $195.00 (a savings of $100.00) until March 31, 1992.