Help Contents

Browse Bound Control version 1.0

developed by Gabriel Oancea

The Browse control is a shareware application, that means you are free to use and test it for 30 days. After this period of time, if you decide you like it you have to register it with the author (see address and Compuserve ID below).

You are free to distribute the demo version of the control in any way you like, if you distribute it with all the documentation enclosed. You are not allowed to sell it.

For inquiries please contact Gabriel Oancea, Compuserve ID: 70404,655, or contact Delta Soft Inc., 12 Danton Court, Ajax, Ontario, Canada, postal code L1S 3G1. Telephone: (416) 619-2018.

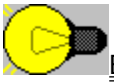
Description

The Browse control can be used in conjunction with the standard VB Data control to view the contents of any database file in a table format. It allows you to select the number and content of the data columns to be displayed, allows editing of the data in the browse window, with data pre- and post-validation - if editing for the respective column is enabled.

Please read the sub-chapters listed below in order to get accustomed with the Browse control.

Introduction	a short introduction to the control
Editing	how to handle editing
Other features	goodies

Introduction


Example

The way the control appears to the users is very similar with a browse created with Clipper TBrowse class, or with the standard xBase BROWSE command: each column has a column header, each row is a record, the user can scroll horizontally and verically either using scroll bars or with the keyboard. Columns can be frozen at the left side of the control window, etc. There are many improvements compared to the DOS based browses, all the standard Windows and VB features are available.

The RecordSource (of the Data control) can have as many records as you like, sorted and filtered as you like, the Browse will display them as provided by the Data control.

How it works:

Create the Data control on the form, you can make it invisible. Set the properties of the Data control, if required. For example you could set the database and record source.
Then create a Browse control and set the DataSource property to the name of the Data control. Set the other Browse properties you want to set at design time.

Now you need to write some initialization code, namely to tell the Browse control how many data columns you will display, and what is the field name and size for each of this columns. You can optionally set the column headers, column data/header alignment, enable edit for columns of your choice, set the number of columns to be "frozen" to the left, set the columns that are displayed in a special colour.

The initialization code can be writen either in the Form_Load() or after the Data control initialization / refresh procedure (for example if you set/change the RecordSource at runtime)

If editing is enabled you might want to write code to pre- and post-validation of data entry.

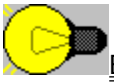
See the example application as well, the code is commented.

Introduction Example:

The example below assumes you have a form (**Form1**) on which there is a Data control Data1 and a Browse control **Brw1**. We assume that the Database is set to **BIBLIO.MDB** and the record source to the **Titles** table, also the DataSource property of the **Brw1** control is set to the Data1 control.

```
Option Explicit

Sub Form_Load()
   ' we want to browse 3 fields from the Titles table
   Brw1.Cols = 3           ' set the number of columns

   ' set the data width for each column (from 0 to Cols - 1, that is 0 to 2)
   Brw1.ColWidth(0) = 50   ' Title
   Brw1.ColWidth(1) = 20   ' Author
   Brw1.ColWidth(2) = 20   ' ISBN

   'Set the column headers for each column
   Brw1.Header(0)   = "Publication title"
   Brw1.Header(1)   = "Author"
   Brw1.Header(2)   = "ISB No"

   'Set the column data source: the field names
   Brw1.ColField(0) = "Title"
   Brw1.ColField(1) = "Author"
   Brw1.ColField(2) = "ISBN"

   ' Here you can set also the column alignment, edit enabling, special
   ' colouring, and other browse properties
End Sub
```

Editing

 Example

One of the most handy features of the control is on-the-spot editing, with data pre-validation and post-validation and an edit flag for each column.

How does the user see it: while browsing a file he/she can either press ENTER to start editing the current (highlighted) field, or just simply start entering the text. Press ENTER when done if you want to save the current field, press ESC to cancel.

All the nice features are there: Cut, Copy and Paste to/from the clipboard, undo, scrolling, etc. - like any normal text box. The normal xBase editing keys are available: press ENTER or PAGEUP / PAGEDOWN to exit edit mode and save, press ESC or click with the mouse outside the edit area to cancel changes. Press UP or DOWN arrows to save and move to the next record up or down.

How do you program it: very simple: just set the property ColEdit(x) = **True**, (where x is the column number for which you want to enable editing; you can enable editing of all columns, of one column or of none, *default is none)*. Then you write code for the EditWhen / EditValid event if you want to do some data validation. The control will take care of the rest (Click on Example above for details).

Some hints and requirements:

Write data validation routines, the control will not attempt any validation at all. That means that if the user enters incompatible data you will get a run-time error.
Do not enable for editing the key (indexed) fields, only if absolutely required
NEVER move the record pointer in the EditWhen / EditValid events !! Also do not request any Action from the browse while in one of this events, there is no need for the REFRESHLINE, this is done for you.

Editing Example

The example below assumes you have a form (**Form1**) on which there is a Data control Data1 and a Browse control **Brw1**. We assume that the Database is set to **BIBLIO.MDB** and the record source to the **Titles** table, also the DataSource property of the **Brw1** control is set to the Data1 control.

```
Option Explicit

Sub Form_Load()
    Dim k As Integer

    ' we want to browse 3 fields from the Titles table
    Brw1.Cols = 3           ' set the number of columns

    ' set the data width for each column (from 0 to Cols - 1, that is 0 to 2)
    Brw1.ColWidth(0) = 50   ' Title
    Brw1.ColWidth(1) = 20   ' Author
    Brw1.ColWidth(2) = 20   ' ISBN

    'Set the column headers for each column
    Brw1.Header(0)   = "Publication title"
    Brw1.Header(1)   = "Author"
    Brw1.Header(2)   = "ISB No"

    'Set the column data source: the field names
    Brw1.ColField(0) = "Title"
    Brw1.ColField(1) = "Author"
    Brw1.ColField(2) = "ISBN"

    For k = 0 To 2
       Brw1.ColEdit(k) = True       ' enable editing for all columns
    Next k
    ' Here you can set also the column alignment, special
    ' colouring, and other browse properties
End Sub

Sub Brw1_EditWhen(nCol As Integer, cField As String, lCancel As Integer)
  ' Check if editing can proceed for column nCol, if not set lCancel to True.
  ' The default action is to start editing.
  ' This event can be used for example to activate a pop-up window with a
  ' limited number of choices, instead of allowing the user to type in the
  data.
  ' The cField argument contains the value that will be edited, you can
  change
  ' this value as required - for example to a default value if the field is
  empty
End Sub

Sub Brw1_EditValid(nCol As Integer, cField As String, lOk As Integer)
  ' Validation routine for column nCol, set lOk to False if the data is
  invalid.
```

```
  ' The cField argument contains the current data as entered by the user;
  ' you can change this value as required.
  ' The default action is to change the data (the DataChanged property of the
  ' control will be set to True) and write it to the database as soon as the
  ' position in the database is changing.
  ' You should use this event to check for empty/duplicate index keys, etc.
  If nCol = 2 Then           ' we must check that the key field is not empty
    If Trim$(cField) = "" Then
      MsgBox "The ISBN cannot be blank!"
      lOk = False
    End If
  End If
End Sub
```

Other features of the Browse control

Some other important properties and events are listed below:

Ability to lock one or more columns at the left edge of the control window: by setting the LeftFrozen property to a number between 1 and Cols, you tell the browse control to 'freeze' the respective number of columns - that means that they will be visible no matter how the user scrolls the other columns. You can change the LeftFrozen property dynamically, at run-time - or let the user do it. This can be very handy if you would like to keep one / more column(s) always in view. Frozen columns can be edited, sized, etc. like any other normal column.

Highlighting columns: you can have selected columns displayed in a special color you can set up, all you have to do is set the ColorSpecial and ColorTSpecial at design or run-time then set the SpcColor(nColumn) = True and the column will be displayed using the two special colors as background / foreground. Useful for underlining some special column - as totals of some sort, or the editable columns if you choose.

Run time adjustable column widths: the user can change the column width by dragging the column separator with the mouse. For this, move the mouse on the column headers, between two columns: the cursor shape will change; press the right mouse button and keep it pressed, the column will appear delimited with a focus rectangle, size it to the right or left, release the mouse button when done. You can also set the column width from code using the ColWidthPix property (the width of each column in pixels). This can be useful to save and restore settings of a query the user has defined: when the control is unloaded (for example in the Form_Unload() save the column width in pixels to some variable(s), then upon exiting the program to a INI file. In the Form_Load() procedure (after setting the ColWidth - data width in characters) set the ColWidthPix to the saved values, eventually the location of the window on screen.

Design-time / run-time adjustable colors: you can set the colors for the header, normal, highlighted cell, frozen columns and special columns. You can offer the user the chance of setting them to his/her preferences (use for example the common dialog box control - Colors), and then you can save and restore them.

Find out when the user tries to move outside the current scope (past EOF or before BOF): the events HitTop and HitBottom are fired every time when the user tries to move outside the current limits. Also the HitTop and HitBottom properties are set to **True** when the user tries to move past EOF of above BOF. You can use these events / properties to ask if he wants to add a record, or just simply beep at him.

Reposition the record pointer / refresh the browse display from code: the Action property allows you to send requests to the browse control, like: RefreshLine, RefreshAll, GoTop, GoBottom, Up, Down, Left or Right. You can look at the Action property as a method, because no custom methods can be implemented in the actual version of the CDK.
You can also force the user to enter Edit mode from code, just set the **Action** property to BRW_ACT_EDIT, and the current column from the current record will be edited.

Browse Properties

The standard Visual Basic properties supported are:
NAME, INDEX, BORDERSTYLE, DATASOURCE, DATACHANGED, DRAGMODE, DRAGICON, ENABLED, FONTBOLD, FONTITALIC, FONTNAME, FONTSIZE, FONTSTRIKE, FONTUNDER, HEIGHT, HELPCONTEXTID, HWND, LEFT, MOUSEPOINTER, PARENT, TABINDEX, TABSTOP, TAG, TOP, VISIBLE, WIDTH. Please see your VB manual for details about any of these properties.

Custom Properties

Available at design time:

ColorNormal	Long	Background color of the normal data cells
ColorHeader	Long	Background color of the column headers
ColorHighlight	Long	Background color of the highlighted data cell
ColorFrozen	Long	Background color of the frozen columns
ColorSpecial	Long	Background color of the special colored columns
ColorTNormal	Long	Text color of the normal data cells
ColorTHeader	Long	Text color of the column headers
ColorTHighlight	Long	Text color of the highlighted data cell
ColorTFrozen	Long	Text color of the frozen columns
ColorTSpecial	Long	Text color of the special colored columns

Available at run-time only, read only at run-time

Rows	Integer	Total no of rows visible in the window
RowHg	Integer	Row height, in pixels
LineLenPix	Integer	Total browse line length, in pixels
LineLenChar	Integer	Total browse data line length, in characters
LeftCol	Integer	Left-most visible column no (from 0 to Cols-1)
RightCol	Integer	Right-most visible column
CurrCell	String	Highlighted cell text
LastLine	Integer	Last row of data in display
EmptyFile	Integer	TRUE if the database has no records, or no record matches the scope / filter
HitTop	Integer	TRUE if the user has tried to move above BOF
HitBottom	Integer	TRUE if the user has tried to move past EOF
HBarDim	Integer	Height of the horizontal scroll bar in pixels, 0 if no bar is required
VBarDim	Integer	Width of the vertical scroll bar in pixels, 0 if no bar is required

Available at run time only, read-write at run time

Header()	String	Columns header
Cols	Integer	Total number of columns in browse
ColWidth()	Integer	Array of column data width, array index starts at 0 to Cols-1
ColWidthPix()	nteger	Array of column pixel widths, array index starts at 0 to Cols-1
ColAlign()	Integer	Columns alignment (left, right or center)
SpcColor()	Integer	True if you want the column highlighted in the special color
ColEdit()	Integer	True if you want edit enabled for the column
ColField()	Integer	Valid field name for the column
Col	Integer	Current display column
Row	Integer	Current display row
LeftFrozen	Integer	No of columns to freeze to the left edge of the control window
Action	Integer	Tell the Browse control what to do

Browse Colors [Long]

```
[Var] = [Form!]Brw.ColorHeader     [ = LongExp ] ' header BackColor
[Var] = [Form!]Brw.ColorTHeader    [ = LongExp ] ' header ForeColor
[Var] = [Form!]Brw.ColorNormal     [ = LongExp ] ' normal cell BackColor
[Var] = [Form!]Brw.ColorTNormal    [ = LongExp ] ' normal cell ForeColor
[Var] = [Form!]Brw.ColorHighlight  [ = LongExp ] ' highlighted cell
BackColor
[Var] = [Form!]Brw.ColorTHighlight[ = LongExp ] ' highlighted cell
ForeColor
[Var] = [Form!]Brw.ColorFrozen     [ = LongExp ] ' frozen columns BackColor
[Var] = [Form!]Brw.ColorTFrozen    [ = LongExp ] ' frozen columns ForeColor
[Var] = [Form!]Brw.ColorSpecial    [ = LongExp ] ' special columns BackColor
[Var] = [Form!]Brw.ColorTSpecial   [ = LongExp ] ' special columns ForeColor
```

The colors can be changed at design time by either typing in a hex value, or by selecting the color from the standard VB color box (click on the '...' button or double-click on the property name to activate it).

You can also change colors at run time as a response to some user selection - for example selecting another color from the CommDlg Colors Selection dialog.

NOTE:
if a color is set both to special and frozen status, the frozen color is used.

Example:

```
' set a flashing background for the current cell in browse
' we create a timer Timer1 and set the interval to 0.3 seconds
' It is nor recommendable to use the timer like this,
' it is just an example for using the colors
Sub Timer1_Timer()
  Static lRed
  If lRed Then
    lRed = False: Brw.ColorHighlight = &HFF0000&     ' Blue
  Else
    lRed = True: Brw.ColorHighlight = &HFF&          ' Red
  End If
End Sub
```

Rows [Integer]

```
[Var] = [Form!]Brw.Rows
```

The Rows property is read-only at run time, and represents the no of rows visible currently, including the blank rows (if any). The header line is not included.

Example:

```
' adjust the height of the browse to fit 20 rows of data
Sub Form_Resize()
  ' if iconic don't bother
  If Me.WindowState = MINIMIZED Then Exit Sub
  ' some form resizing code here, make sure there is room enough
  If Brw.Rows <> 20 Then
    nOldScaleMode = Me.ScaleMode         ' save old scale mode
    Me.ScaleMode = PIXELS                ' defined in CONSTANT.TXT = 3
    ' recalc height: 20 rows+1 header line + Horz Bar height
    Brw.Height = (20 + 1) * Brw.RowHg + Brw.HBarDim
    Me.ScaleMode = nOldScaleMode         ' and restore scale mode
  End If
End Sub
```

RowHg [Integer]

```
[Var] = [Form!]Brw.RowHg
```

The RowHg property is read-only at run time, and represents the height (in pixels) of one data row, it will change when the font name/style changes

Example:

```
' adjust the height of the browse to fit 20 rows of data
Sub Form_Resize()
  ' if iconic don't bother
  If Me.WindowState = MINIMIZED Then Exit Sub
  ' some form resizing code here, make sure there is room enough
  If Brw.Rows <> 20 Then
    nOldScaleMode = Me.ScaleMode         ' save old scale mode
    Me.ScaleMode = PIXELS                ' defined in CONSTANT.TXT = 3
    ' recalc height: 20 rows+1 header line + Horz Bar height
    Brw.Height = (20 + 1) * Brw.RowHg + Brw.HBarDim
    Me.ScaleMode = nOldScaleMode         ' and restore scale mode
  End If
End Sub
```

LineLenPix [Integer]

```
[Var] = [Form!]Brw.LineLenPix
```

The LineLenPix property is read-only at run time, and represents the length of one full line of data (if all columns would be visible). It will change when the font name or style, no of columns, data width of a column or column width in pixels are changing.

Example:

```
' adjust the width of the browse to fit all columns in display
Sub Form_Resize()
  ' if iconic don't bother
  If Me.WindowState = MINIMIZED Then Exit Sub
  nOldScaleMode = Me.ScaleMode           ' save old scale mode
  Me.ScaleMode = PIXELS                  ' defined in CONSTANT.TXT = 3
  If Brw.LineLenPix + Brw.VBarDim <= Me.ScaleWidth Then
    Brw.Width = Brw.LineLenPix + Brw.VBarDim  ' reset the width
  End If
  Me.ScaleMode = nOldScaleMode           ' and restore scale mode
End Sub
```

LineLenChar [Integer]

```
[Var] = [Form!]Brw.LineLenChar
```

The LineLenChar property is read-only at run time, and represents the length, in characters, of one line of data. It is the sum of all column widths and it will change when the no of columns or data width of a column are changing.

LeftCol, RightCol [Integer]

```
[Var] = [Form!]Brw.LeftCol
[Var] = [Form!]Brw.RightCol
```

The LeftCol and RightCol properties are read-only at run time, and represent the left-most and respectively the right-most visible columns in the browse. They change as the user moves using the left / right arrow keys or the horizontal scroll bar. Useful when you need to know whether a column is visible or not, for edit purposes for example.

The LeftCol represents the left-most **NOT FROZEN** column (the frozen columns start with 0 to LeftFrozen - 1).

Example:

```
' we want to count all the empty EXPIRES date fields in a database,
' in a fancy way
' For this we set up a disabled browse Brw and write the function:
Function CountBlanks() As Integer
  Dim nCnt As Integer
  ' select the file and go top, we assume the browse is already setup
  ' and the EXPIRES field is in column 3, we have 2 columns frozen
  Brw.Action = BRW_ACT_GOTOP       ' go top
  Do While Brw.LeftCol < 3         ' put the EXPIRES column near the frozen
  cols
    Brw.Action = BRW_ACT_RIGHT     ' move one to the right
  Loop

  Do While Not Brw.HitBottom
    If Len(Brw.CurrCell) = 0 Then nCnt = nCnt + 1
    Brw.Action = BRW_ACT_DOWN      ' the user can see the cursor moving down
    fast
  Loop
End Sub
```

CurrCell [String]

```
[Var] = [Form!]Brw.CurrCell
```

The CurrCell property is read-only at run time, and represents the text in the current cell (that is the active cell - the one in the highlighted color). It changes every time the user moves in the browse, you can use the Change event to find out when it changed, for example in order to update a Label control caption. The character string returned by the CurrCell property is what the user gets to edit, if edit is enabled for the specified column, it will be passed in the EditWhen event to you for pre-validation.

Example:

```
' we want to show the contents of the current cell in the Label1 control
Sub Brw_Change( nRowCol As Integer )
  Label1.Caption = "Current Cell: " & Brw.CurrCell
End Sub
```

LastLine [Integer]

```
[Var] = [Form!]Brw.LastLine
```

The LastLine property is read-only at run time, and represents the last line of data in the browse control window. If there is enough data (records) in the database to fill all the rows, LastLine = Rows and a vertical scroll bar is present (VBarDim > 0), otherwise LastLine is smaller and no vertical bar is displayed (also VBarDim will be set to 0). It can be smaller, although there are enough records to be displayed, if you use the vertical scroll bar by dragging the thumb near the end of the file. In this case the control will position the file on the new record, and will place it on the first line in the browse; if there are less records to EOF than lines visible, then LastLine will be smaller than Rows.

If the file is empty (no records), then LastLine is 0, and EmptyFile will be set to **True**. It is recommendable to use EmptyFile to test for the empty file condition.

Example:

```
' shrink the browse if less records than no of lines visible in browse,
' so that we do not display blank lines
Sub Form_Load()
  ' Initialize the browse
  If Brw.LastLine < Brw.Rows Then
    ' Set the new browse Height
    Brw.Height = Brw.LastLine * Brw.RowHg + Brw.HBarDim + 1
  End If
End Sub
```

EmptyFile [Integer]

```
[Var] = [Form!]Brw.EmptyFile
```

The EmpyFile property is read-only at run time, and is set to **True** if the database file is empty - has no records or if there are no records meeting the criteria you used to set the RecordSource. If the EmptyFile is set to **True**, the LastLine is set to 0 and the data lines are cleared, no vertical bar is displayed, the current cell is set to "" (an empty string) and cursor (active cell) will not be allowed to move except on row 0 (the first row).

Example:

```
' save a set of values entered by the user, if the file is empty add a
record
Function MyRecSave() As Integer
  Dim lOk As Integer

  If Brw.EmptyFile Then
    lOk = AppendOneRecord()  ' add a blank record
  End If

  If Not lOk Then
    MsgBox "Cannot append record!"       ' append failed
    MyRecSave = lOk                      ' return FALSE
    Exit Function
  End If

  ' replace data...
  ' ...............
  MyRecSave = True
End Function
```

HitTop, HitBottom [Integer]

```
[Var] = [Form!]Brw.HitTop
[Var] = [Form!]Brw.HitBottom
```

The HitTop, HitBottom properties are read-only at run time, and they are set to **True** if the user has tried to move up before the beginning of file or before the first record in scope and, respectively past end of file or down past the last record in scope. In both cases first the HitTop / HitBottom events are fired, and then the two properties are set.
The two properties are useful if you use the `BRW_ACT_DOWN / BRW_ACT_UP` actions from code to move the record pointer, for example if you want to print all records - as in the example below.

Example:

```
' print all the records in the file
Function PrintMyQuery() As Integer
  If Brw.EmptyFile Then
    MsgBox "Nothing to print"
    PrintMyQuery = False
    Exit Function
  End If

  Brw.Action = BRW_ACT_GOTOP
  ' print some headers
  Do While Not Brw.HitBottom
    PrintOneLine                  ' print one data line
    Brw.Action = BRW_ACT_DOWN     ' skip one record and refresh the display
  Loop
  Printer.EndDoc
  PrintMyQuery = True
End Function
```

HBarDim, VBarDim [Integer]

```
[Var] = [Form!]Brw.HBarDim
[Var] = [Form!]Brw.VBarDim
```

The HBarDim, VBarDim properties are read-only at run time, and they represent the height of the horizontal scroll bar and the width of the vertical scroll bar of the browse control, respectively. They are 0 if the respective scroll bar is not required.
NOTE: the two properties are always in pixels, no matter what the ScaleMode of the container of the browse control is. If you use a different ScaleMode, and you plan to use them, you should transform the coordinates from pixels to your mapping mode.

The two properties can be used as an indicator of the presence of the scroll bars, and also they are useful for run-time resizing of the browse.

Example:

```
' adjust the height of the browse to fit 20 rows of data
Sub Form_Resize()
  ' if iconic don't bother
  If Me.WindowState = MINIMIZED Then Exit Sub
  ' some form resizing code here, make sure there is enough room
  If Brw.Rows <> 20 Then
    nOldScaleMode = Me.ScaleMode        ' save old scale mode
    Me.ScaleMode = PIXELS               ' defined in CONSTANT.TXT = 3
    ' recalc height: 20 rows+1 header line + Horz Bar height
    Brw.Height = (20 + 1) * Brw.RowHg + Brw.HBarDim
    Me.ScaleMode = nOldScaleMode        ' and restore scale mode
  End If
End Sub
```

Header [Array of Strings]

```
[Var] = [Form!]Brw.Header(nColumn%) [ = StringExp]
```

The Header property is an array of strings representing the column headers for each data column (field). By default the header is set to the name of the field (the ColField(nColumn%)).

If the column contents changes, you should re-assign the Header.

NOTES:

The Header string can be wider than the data width of the column, but it will be clipped to fit the size of the header cell in which is displayed.

Example:

```
' see the Introduction example
```

Cols [Integer]

```
[Var] = [Form!]Brw.Cols [ = IntegerExp]
```

The Cols property is an integer, available only at run-time, which sets the number of data columns displayed by the browse. It must be a positive number and no bigger then BRW_MAX_NO_OF_COLS (see Constants).

Assigning the Cols of the control is the first logical step of initialization, although is not preemptive. But when you change the Cols everything is resized / changed so it is a good practice to set the Cols first, and when you change it, change the others related properties as well. The next step would be to assign the columns data width (ColWidth) and then the columns Header.

For a detailed explanation about the browse initialization see Introduction and the example there.

ColWidth [Array of Integers]

```
[Var] = [Form!]Brw.ColWidth(nColumn%) [ = IntegerExp]
```

The ColWidth property is an array of integers, available only at run-time, which sets the width of the data displayed in each column, from 0 to Cols - 1, in characters. It must be positive and less than BRW_MAX_COL_WIDTH for each column (see Constants).

After assigning the Cols (number of columns) of the control, the next logical step of initialization is to assign the columns data width for each column and then the columns Header.

The value assigned for each column tells the control how much space to allocate for each column's data. Data retrieved by the browse will be assigned to each column using the values in the ColWidth array.

The ColWidth property can be dynamically changed, this will resize the internal buffer maintained by the control, and will also resize the LineLenChar property for the respective column. Please note that changing the ColWidthPix() or resizing the columns with the mouse will **NOT** affect the ColWidth!

For a detailed explanation about the browse initialization see Introduction and the example there.

ColWidthPix [Array of Integers]

```
[Var] = [Form!]Brw.ColWidthPix(nColumn%) [ = IntegerExp]
```

The ColWidthPix property is an array of integers, available only at run-time, which sets the width of each column (the displayed width in pixels not the data width in characters as ColWidth does), valid indexes are from 0 to Cols - 1. This property changes when the ColWidth changes and also when the user resizes the column with the mouse, or when the font changes. Changing this property will not affect how many characters are stored in the internal buffer for each column.

The default value is the column width in characters multiplied by the average character width for the selected font.

You would have no reason to set this property from code, except one case: you want to restore a previous state of a user defined query window, see the example below.

Example:
The following example stores the last position and other settings of a user defined query into a user type, which is in turn saved to a disk file or INI file when the user exits the application. We will just show the code to save the last status of the window and restore it from the sStatus variable. Saving and restoring to/from file are not shown, you can use either a random file or some WinAPI calls like Get/SetProfileString.

```
DefType MYSTATUS
  left                            As Integer
  top                             As Integer
  width                           As Integer
  height                          As Integer
  ColWdt(0 to BRW_MAX_NO_OF_COLS) As Integer
  ' other data as DBF name, index name, scope, ...
End Type
Global sStatus As MYSTATUS

Sub Form_Unload( Cancel As Integer )
  ' save form position and browse column widths
  sStatus.left   = Me.Left
  sStatus.top    = Me.Top
  sStatus.width  = Me.Width
  sStatus.height = Me.Height
  For k = 0 to Brw.Cols - 1
    sStatus.ColWdt(k) = Brw.ColWidthPix(k)
  Next k
  ' save other data to sStatus
End Sub

Sub Form_Load()
  ' initialize the browse, ...
  ' then restore status, or start with default values if no status saved
  If sStatus.height = 0 Then Exit Sub    ' no previously saved status, exit
  Me.Left   = sStatus.left
  Me.Top    = sStatus.top
  Me.Width  = sStatus.width
  Me.Height = sStatus.height
  For k = 0 to Brw.Cols - 1
```

```
    Brw.ColWidthPix(k) = sStatus.ColWdt(k)
  Next k
End Sub
```

ColAlign [Array of Integers]

```
[Var] = [Form!]Brw.ColAlign(nColumn%) [ = IntegerExp]
```

The ColAlign property is an array of integers, available only at run-time, which sets the way the column headers and data are aligned (justified) within the display rectangle. The headers and data can have different alignments (header left justified, data right justified for example). By default both the headers and the data are left aligned.

Available alignments are left (default), center and right.

You can set the column alignments to a combination of data + header alignment constants, for example `BRW_ALIGN_LEFT + BRW_ALIGNH_RIGHT` (see Constants for the values).

After assigning the number of columns (Cols) of the control and the column widths (ColWidth), you can set the column alignment for each column, as required.

Example:

```
Sub Form_Load()
  ' initialization code comes here

  Brw.ColAlign(0) = BRW_ALIGN_CENTER + BRW_ALIGNH_CENTER   ' both centered
  Brw.ColAlign(1) = BRW_ALIGN_LEFT + BRW_ALIGNH_CENTER     ' left, header
  right
  Brw.ColAlign(2) = BRW_ALIGN_RIGHT + BRW_ALIGNH_RIGHT     ' both right
End Sub
```

SpcColor [Array of Integers]

```
[Var] = [Form!]Brw.SpcColor(nColumn%) [ = IntegerExp]
```

The SpcColor property is an array of integers (valid values are **True** and **False**). This property, in conjunction with ColorSpecial and ColorTSpecial, is used to display the selected column(s) in a different color. You can set the background and foreground color of the special colored columns at design time or run time, then after initializing the browse, set the SpcColor to True for all the columns you want displayed in the special color. By default no column is specially colored and the two special color are the same as the normal colors.

You can use this property to display the editable columns in a different color, or to highlight a calculated column, for example.

If editing is enabled for the special columns, the color of the edit window is the same as the columns color.

Example:

```
Sub Form_Load()
  ' initialize the browse

  ' set color combination for the special columns
  Brw.ColorSpecial  = QBColor(12)  ' intense red background
  Brw.ColorTSpecial = QBColor(10)  ' intense blue text
  Brw.SpcColor(0)   = True   ' Display the first column in the special col.
End Sub
```

ColEdit [Array of Integers]

```
[Var] = [Form!]Brw.ColEdit(nColumn%) [ = IntegerExp]
```

The ColEdit property is an array of integers (valid values are **True** and **False**). This property, when set to **True**, will enable editing for the specified column. By default editing is disabled for all columns.

At run-time the EditWhen event will be fired prior to enter Edit mode, then an EditValid event is fired when the user leaves the Edit mode with intention to save.

You can prevent the user to enter Edit mode by setting the **lCancel** argument of the EditWhen event to **True**, if for any reason you do not want edit for current record.

You can force data saved to be valid by setting the **lOk** argument of the EditValid event to **False**, you can even return a suggested value in the **cField** string argument.

See Editing for more details.

NOTES:

Do not issue a BRW_ACT_REFRESHLINE in the EditValid, refreshing is done for you. Do not change the record pointer while in the events.

The number of characters allowed in Edit mode is equal to the ColWidth() - the data width for the respective column.

Example:

See the example at Editing.

ColField [Array of Strings]

```
[Var] = [Form!]Brw.ColField(nColumn%) [ = StringExp]
```

The ColField property is an array of strings (valid field names from the current work area). You have to assign this property in order to display anything but blanks. Assigning a wrong field name will trigger a run-time error.

Example:
See the example at Introduction

Col, Row [Integer]

```
[Var] = [Form!]Brw.Col [ = IntegerExp ]
[Var] = [Form!]Brw.Row [ = IntegerExp ]
```

The Col and Row properties represent the current column and row in the browse (where the highlighted cell is). The user can change any of them by moving with the arrows, clicking with the mouse, etc.

If you assign them, Col must be between 0 and Cols - 1, Row must be between 0 and LastLine - 1, otherwise a run-time error is generated.

By assigning Row and / or Col you force the control to move the highlighted cell there.

Example:

```
' this example shows a way to keep the user out of the frozen columns
Sub Brw_Change( nRowCol As Integer )
  If Brw.Col < Brw.LeftFrozen Then Brw.Col = Brw.LeftFrozen
End Sub
```

LeftFrozen [Integer]

```
[Var] = [Form!]Brw.LeftFrozen [ = IntegerExp ]
```

The LeftFrozen property sets the no of columns to be visible all the time, these columns start with column 0 to LeftFrozen - 1. The frozen columns will be excluded from the horizontal scroll, when the user moves to the right or left, either with the mouse or using the keyboard.

The LeftFrozen columns will be displayed in the ColorFrozen / ColorTFrozen combination, by default these colors are the same with the normal colors. Increasing / decreasing the LeftFrozen will dynamically add / remove fixed columns to the left edge. If a column is both Special color and Frozen the Frozen colors will take precedence.

If the combined width of all the frozen columns exceeds the control width, then the control cannot be scrolled horizontally. The LeftCol is the number of the left-most visible column excluding the frozen columns.

Example:

```
' freeze 2 columns to the left
Brw.LeftFrozen = 2
```

Action [Integer]

```
[Var] = [Form!]Brw.Action [ = IntegerExp ]
```

The Action property is the way you can send messages to the control, requiring it to perform some action. It does not hold any meaningful value - it is always 0.

Normally you do not require to refresh the Browse control, the Data control will take care of that every time you make a change to the data or reposition the database.

To request the control to perform a refresh line action - that is for example to move up, set the Action property to `BRW_ACT_UP`. All the `BRW_ACT_*` constants are defined in the include file `BRW_INC.TXT` and shown in the Constants section. You can merge this file with one of your modules, it is recommendable to use the constants, not the numeric values.

Available actions:

`BRW_ACT_REFRESHLINE`: Refresh the current line of data. Not required normally.

`BRW_ACT_REFRESHALL`: Refresh all the lines visible in the control. Not required normally.

`BRW_ACT_REFRESHBAR`: Refresh the data of the vertical scroll bar. Not required normally.

`BRW_ACT_GOTOP`: Move to the first record in the database and refresh all visible records.

`BRW_ACT_GOBOTTOM`: Move to the last record in the database and refresh all visible records.

`BRW_ACT_UP`: Move one record up (if posible).

`BRW_ACT_DOWN`: Move one record down (towards the end of file, if posible).

`BRW_ACT_LEFT`: Move one column to the left, if posible.

`BRW_ACT_RIGHT`: Move one column to the right, if posible.

`BRW_ACT_EDIT`: Enter edit mode for the current cell (Row and Col), if editing is enabled for the column. You should postion the cursor using the Row and Col properties before entering edit mode.

Browse Events

The standard Visual Basic events supported are:
DRAGDROP, DRAGOVER, GOTFOCUS, KEYDOWN, KEYPRESS, KEYUP and LOSTFOCUS. Please see your VB manual for details about any of these events.

Custom Events

Change(nRowCol as Integer)	- row, col or record no has changed
HitBottom()	- the user hits the end of file
HitTop()	- the user hits the beginning of file
EditWhen(nCol As Integer, cField As String, lCancel As Integer)	- data edit pre-validation
EditValid(nCol As Integer, cField As String, lOk As Integer)	- data edit post-validation

Change(nRowCol As Integer)

```
Sub Brw_Change( [Index As Integer, ] nRowCol As Integer )
```

This event is fired whenever the user changes the position of the highlighted cell, using the mouse or the keyboard (for example moving up, down, left, right, page up, page down, top of file, bottom of file, etc.).

The **Index** argument uniquely identifies the browse control from an array of browse controls; the **nRowCol** argument holds information about what has changed and can be one of the constants below (see Constants for values):

The record no has changed, the Row and Col are the same	BRW_CHANGE_REC
The record no and the Row have changed	BRW_CHANGE_ROW
The column no has changed, the same record no	BRW_CHANGE_COL
Both the row and column no have changed	BRW_CHANGE_BOTH

Write code to update the current cell in this event procedure, for example.

HitTop(), HitBottom()

```
Sub Brw_HitTop    ( [Index As Integer] )
Sub Brw_HitBottom( [Index As Integer] )
```

This events are fired when the user has tried to move before BOF or past EOF, either using the mouse or the keyboard. Both events can be ignored, you can set a Beep statement in the event procedure, or ask the user if he/she wants to add a new record (in HitBottom) to emulate the standard xBase BROWSE command.

The **Index** argument uniquely identifies the browse control from an array of browse controls.

Example:

```
Sub Brw_HitBottom()
  If MsgBox("Add new record?", MB_ICONQUESTION + MB_YESNO) = IDYES Then
    ' add a blank record and reposition the browse
    AppendOneBlank()
  End If
End Sub

Sub Brw_HitTop()
  Beep
End Sub
```

EditWhen(nCol As Integer, cField As String, lCancel As Integer)

```
Sub Brw_EditWhen( nCol As Integer, cField As String, lCancel As Integer )
```

This event is fired whenever the user tries to edit a field from column **nCol**, where **nCol** is a column enabled for editing. The **cField** argument is the current value of the field - the one that will be edited. The **lCancel** argument is a boolean flag (set to **False** by default), which tells the control if edit should be allowed; setting **lCancel** to **True** will not allow the user to enter in edit mode.

The **Index** argument uniquely identifies an element of a control array.

This event allows you to make a data pre-validation, before the user enters edit mode, and cancel the edit if you decide so, even if the column is enabled for edit.
See also Editing for details.

Example:
The following example will allow editing of the date field DUEDATE, in column 3, enabled for editing, only if the field DUEAMOUNT is greater than 0.

```
Sub Brw_EditWhen( nCol As Integer, cField As String, lCancel As Integer )
  If nCol <> 3 Then Exit Sub       ' check only column 3, DUEDATE
  If Data1.RecordSet.Fields("DUEAMOUNT").Value <= 0 Then
     ' no edit allowed
     Beep: MsgBox "The DUEAMOUNT must be positive!"
     lCancel = True
  End If
End Sub
```

EditValid(nCol As Integer, cField As String, lCancel As Integer)

```
Sub Brw_EditValid( nCol As Integer, cField As String, lOk As Integer )
```

This event is fired whenever the user tries to save an edited value for column **nCol**, where **nCol** is a column enabled for editing, and in this respect is much like a VALID clause in a standard xBase GET command. The **cField** argument is the value entered by the user - the one that will be eventually replaced. The **lOk** argument is a boolean flag (set to **True** by default), which tells the control if the value is Ok; setting **lOk** to **False** will prevent the control to exit edit mode, until a valid value is entered or the edit is cancelled.

You can set the **cField** to a string expression, for example as a default value, if the user has entered a wrong value in edit mode.

The **Index** argument uniquely identifies an element of a control array.
See also Editing for details.

Example:
In the following example we have two columns enabled for editing, DUEAMOUNT - column 2 and DUEDATE - column 3.

```
Sub Brw_EditValid( nCol As Integer, cField As String, lOk As Integer )
    If nCol = 2 Then              ' edit DUEAMOUNT
      If Val(cField) < 0 Then     ' no negative numbers allowed
        MsgBox "Please enter a positive value!"
        lOk = False
      End If
    ElseIf nCol = 2 Then                      ' edit DUEDATE
      If IsDate((cField)) Then Exit Sub       ' Ok, replace data
      MsgBox cField & " is not a valid date!"
      lOk = False
    End If
End Sub
```

Browse Constants

All the constants listed bellow are available as text in the include file **BRW_INC.TXT** provided.
IMPORTANT NOTE: do not change the values for these constants, you can and will get run-time errors. (For example increasing the maximum number of columns from 64 to 128 will generate a run-time error when you will set the columns to 100). The control was compiled using these values, changing them here will have no effect on the behaviour of the VBX.

Limitative Constants

Maximum number of columns in the browse	BRW_MAX_NO_OF_COLS	64
Maximum column width (in characters)	BRW_MAX_COL_WIDTH	256

Error Codes

Invalid index referrence to one of the array properties	BRW_ERR_BADINDX	81
Invalid number of columns	BRW_ERR_NO_OF_COLS	2700
Invalid column no	BRW_ERR_COL_NO	32702
Invalid row number	BRW_ERR_ROW_NO	32703
Invalid data column width (characters)	BRW_ERR_COL_WIDTH	32704
Invalid column width (pixels)	BRW_ERR_COL_WIDTHX	32705

Available actions (for the Action property)

Refresh current line	BRW_ACT_REFRESHLINE	1
Refresh all visible lines / records (impl. RefreshBar)	BRW_ACT_REFRESHALL	2
Refresh the vertical scroll bar (not needed normally)	BRW_ACT_REFRESHBAR	3
Go to the last record in file / scope and refresh all	BRW_ACT_GOBOTTOM	4
Go to the first record in file / scope and refresh all	BRW_ACT_GOTOP	5
Skip one record up, if not posible a HitTop is fired	BRW_ACT_UP	6
Skip one record down, if not pos. a HitBottom is fired	BRW_ACT_DOWN	7
Move to the next column to the right (if any)	BRW_ACT_RIGHT	8
Move to the next column to the left (if any)	BRW_ACT_LEFT	9
Edit current cell, if editing is enabled	BRW_ACT_EDIT	10

Possible values of the nRowCol argument of the Change event:

The record no has changed	BRW_CHANGE_REC	0
The record no and the Row have changed	BRW_CHANGE_ROW	1
The column no has changed	BRW_CHANGE_COL	2
Both the row and column no have changed	BRW_CHANGE_BOTH	3

Column text display alignment for the ColAlign propriety

The column text is left justified (default):	BRW_ALIGN_LEFT	0
The column text is centered	BRW_ALIGN_CENTER	1
The column text is right justified	BRW_ALIGN_RIGHT	2
The column header text is left justified (default):	BRW_ALIGNH_LEFT	4
The column header text is centered	BRW_ALIGNH_CENTER	8
The column header text is right justified	BRW_ALIGNH_RIGHT	16

Keyboard Summary

Vertical movement:

UP	**UP**
DOWN	**DOWN**
CTRL-UP	**FIRST LINE IN WINDOW**
CTRL-DOWN	**LAST LINE ON WINDOW**
PAGEUP	**PAGE UP**
PAGEDOWN	**PAGE DOWN**
CTRL-PAGEUP	**TOP OF FILE**
CTRL-PAGEDOWN	**BOTTOM OF FILE**

Horizontal movement:

LEFT ARROW	**LEFT**
RIGHT ARROW	**RIGHT**
HOME	**LEFT-MOST VISIBLE COLUMN**
END	**RIGHT-MOST VISIBLE COLUMN**
CTRL-LEFT	**PAGE LEFT**
CTRL-RIGHT	**PAGE RIGHT**
CTRL-HOME	**LEFT-MOST COLUMN**
CTRL-END	**RIGHT-MOST COLUMN**

Edit keys:

ENTER	**Enter edit mode, if edit for current column enabled**
Any alphanumeric key	**Same as Enter + replace field contents with the entered character**
ESC	**Cancel edit**
PAGEUP / PAGEDOWN, ENTER	**Exit edit mode with saving, stay on the same row**
UP / DOWN ARROW	**Save field and move to previous / next row (record) - same column**