bBrowser Tips & Tricks

Tom Walden

DevCon USA 2003

Introduction

How many of you have been on the Visual Objects newsgroup and have asked your share of questions about how to accomplish some obscure feature with a DataBrowser or (Data)ListView object? Picking on my Aussie friends - Gary will quickly pop in and say that he can do that DataBrowser in just 12 lines of code on his handheld, then Phil will [snip] and say he can do it with a ListView in just 8 lines of code. But holy boomerangs, Geoff jumps on the tree with the wallabee to announce that he can do it in just two lines of code using bBrowser. But then Geoff gets frustrated when the other person asks "what is a bBrowser" and the boomerang knocks Geoff out of the tree.

But in all seriousness, bBrowser is to VO what tBrowse was to Clipper, and then some, well actually, a lot more. And although a "limited edition" version of bBrowser now ships with VO, I would highly recommend that you purchase the full version (source code is optional, and not necessary for our discussions here).

There really isn't much that you cannot do with bBrowser, but here is a condensed list of some of it's major features:

- Works with either a dbServer object or as an array browser.
- Can operate in cell or line mode.
- Can assign field specs to columns.
- Can operate in many cell data entry modes, such as Excel simulation.
- Logical fields operate as either check boxes or comboboxes.
- Multi-lines within individual cells.
- Individual cells/rows can all have conditional color settings.

What am I going to discuss in this paper?

This paper will give you an insight on "how" to implement various functionality within bBrowser. This information in this document is a culmination of my own source code, the VO Newsgroup, and from the bBrowser author himself. Initially I will discuss the history of bBrowser and it's future wish list. Then I will start with the very basics and gradually jump into harder subjects as the paper progresses.

History of bBrowser

How many of you have been on the Visual Objects newsgroup and have asked "what is a bBrowser?". Well, as in most third-party tools in (Clipper or) VO, somebody had a specific need for a feature that did not come with the language – so they wrote their own. And then it turns out to be such a nice product that lots of other people also want, that they then decide to sell it. We've all seen it happen many, many times throughout our programming careers. Thus the rest is history…

bBrowser was developed by a German - Joachim Bieler, which may not be too surprising since VO was developed mostly by Germans too. Unfortunately the first version of bBrowser was, you guessed it, written in German and came with German docs. I want ahead and bought myself a copy anyway after I saw the demo. I lived in Germany for 4 years as a kid, so I still had limited knowledge of the language – for the rest I just used translation software (which really didn't help much with technical terms). The product was quite usable regardless. Finally Joachim got enough pressure to produce an English set of docs, which I helped proofread. Obviously from that point on it became "the choice of browsers" for use in VO applications. But here is a chronological history of bBrowser:

- 1997
  The beginning of the development of a browser (for internal requirements) with the following priorities:
    - High speed when opening databases with many fields (> 80)
    - High speed when scrolling
    - Optimized vertical scrollbar at which the slider shows the relative position within the database
    - Higher flexibility:
        - Simple color conditions for columns and records
        - Simple interface for editing cell values
        - Automatic refresh of the displayed data, if other programs in the network change the data of a database.

    - § Simple freezing of data columns

- 1998, August:
  Presentation of the browser basic framework at the User Group of Hamburg (Germany). Jan Waiz persuades him to introduce the browser framework at the DevCon in Nuremberg (Germany).

- 1998, November:
  Presentation of the browser basic framework on the DevCon in Nuremberg (Germany)

- 1999, February:
  Further development of the browser basic framework to the complete product called "bBrowser" for the German market.

- 2001, March:
  bBrowser 1.3 is available as an international version with English documentation.

- The bBrowser product supports CA-Visual Objects 2.0b4, 2.5b-3 and 2.6

What is next for bBrowser?

Here is a list of future enhancements for bBrowser:

- Horizontally scolling of columns in pixel steps
- Combining from several columns to a column group. The following functionality is supported within a column group:
    - o Group caption over a set of sequential column captions
    - o Interactive moving of column groups with the mouse.
    - o The columns of a column group can be moved within the group.
- Support of oblique column captions
- Support of column values with RTF format
- Support of a TreeView column
- Conditions for editing and selecting (like the color conditions)
- Optimized support for SQL

Some Simple Tips and Tricks for the Novice

First of all, there is always some confusion concerning the use of the bBrowser Refresh() and Redraw() methods. FYI - it's not the redraw after the refresh that is important, it's that you do things in the correct sequence. If you change the state of the server in question, then you must do something to rebuild the browser row data. That will be caused by calling Refresh(), but equally you could also call Skip(0), GoTop(), etc. and not need to call Refresh(). Refresh() is only needed if you suspend notification, do stuff and then reset notification. If you have

done somthing like Append() inside suspended notification you even need to Notify() for a filechange. Refresh calls Redraw internally so you never need to call both together.

But if you change anything like captions, color conditions, etc., and don't do any record movement in the server, then you call Redraw(#FromColumn/Row) or Redraw() for the entire table. But be selective, redraw only that part of the browser you affect. Your code will of course run faster that way.

Okay, let's first learn one of the most commonly used bBrowser methods that you will be using – GetColumn(). You are, however, in fact creating an object of the bDataColumn class. You pass only a single parameter to this method, but it can be either a symbol name of the column, a string value of the column name, or the respective sequential number of the column itself. Thus a typical column change scenario would start something like this:

```
oCol                  := obBrowser:GetColumn( 5 )
oCol:Caption   := "What U want to change it to"
```

or, as an alternative method if you are wanting to change a caption on-the-fly:

```
obBrowser:ColumnList[ ColumnNo ]:Caption := 'What U want to
change it to'
```

One important note that you need to remember is this – it is always more efficient and more user-friendly to suspend notification to your bBrowser object prior to making any sort of display or physical changes to it. I don't typically enter these commands in the sample code in this document, but you still need to be cognizant about using it:

```
SELF:oDCbbPlayers:SuspendNotification()
…make all your codes changes
SELF:oDCbbPlayers:ResetNotification()
```

What if I want standard defaults and colors without having to set them for every bBrowser object?

That's an easy one - sub-class bBrowser! Then inherit from it in your bBrowser control properties. You could simply pass an array to it with pertinent information to setup each individual bBrowser instance.

How can I display a different color for each quarter of a year?

```
bColorCondition{ {|x| Month( x:Server:fieldget( #DateField ) ) < 4 }, oBrowser, oLightGreen, oBrush }
bColorCondition{ {|x| Month( x:Server:fieldget( #DateField ) ) < 7 }, oBrowser, oLightBlue, oBrush }
bColorCondition{ {|x| Month( x:Server:fieldget( #DateField ) ) < 10 }, oBrowser, oYellow, oBrush }
bColorCondition{ {|x| Month( x:Server:fieldget( #DateField ) ) < 13 }, oBrowser, oLightRed, oBrush }
```

I have a large database and it's taking a long time to load my bBrowser object, how can I speed that up?

The :Use() method literally takes a long time to load larger database files. If you are building the bBrowser object by hand (which I always do), then try adding these lines prior to your :Use() statement.

```
SELF:obBrowser:Rowmode := #Absolute
SELF:obBrowser:RowCountMode := #RecCount
```

How can I remove a context menu from a bBrowser object?

The following call can be used to remove the context menu from the data area, only. For the data area no context menu will be shown unless a general context menu was assigned with bBrowser:ContextMenu to the bBrowser.

```
self:obBrowser:SetContextMenu(, NULL_OBJECT)
```

How do I add a new data column on-the-fly (and delete my old column too)?

```
oGridColumn := bDataColumn{ SELF:oDCbbClients,
SELF:oClientServer, ;
     {|Server| Trim(Server:SomeFieldName)}, ;
     #NewColumnName}

oGridColumn:HyperLabel := HyperLabel{#OldColumnName}
oGridColumn:editable     := FALSE
oGridColumn:Selectable   := TRUE
oGridColumn:caption      := "Client Notes"
oGridColumn:width        := 160
oGridColumn:ValType      := "M"

// add column to browser
SELF:oDCbbClients:AddColumn(oGridColumn)
// show column at position 2
SELF:oDCbbClients:OpenColumn(oGridColumn, 2)
```

How do I allow a user to click on a column header so they can sort on that column?

First of all, this depends if you are using a dbServer or an ArrayServer to display the data. The methods are slightly different in nature, thus I will show you both scenarios. However, regardless of which server you are using, you still need to include a statement similar to this one in your code to turn on the "caption click" capability:

```
SELF:oDCbBrowser:EnableCaptionClick( TRUE )
```

First I will show you how to handle caption clicks using an ArrayBrowser. But regardless which server you are using, you will still need to create a CaptionClick() method for your class. Notice that there are NO "index tag names" involved in this process at all. That is because a bBrowser ArrayBrowser uses only a single index expression at a time, which obviously also saves memory too. Re-indexing an ArrayBrowser on-the-fly, even for thousands of records, is so fast anyway that you need not be concerned about time as an issue.

```
METHOD CaptionClick( oCol ) CLASS PlayersBrowser
```

```
// oCol = bColumnEvent object
// used to change the sort of the browser
LOCAL nRec     AS INT

SELF:oDCbbPlayers:SuspendNotification()

// We want to go back to our current record after indexing:
nRec := SELF:oDCbbPlayers:Server:recno

// Clear out our existins index:
SELF:oDCbbPlayers:Server:ClearOrder()

DO CASE
CASE oCol:Column:FieldSym == #PlayerName
     SELF:oDCbbPlayers:Server:CREATEORDER( "upper(
server:LastName + server:FirstName" ) )

CASE oCol:Column:FieldSym == #PlayerNr
    SELF:oDCbbPlayers:Server:CREATEORDER( "server:PlayerNr )"
)

CASE oCol:Column:FieldSym == #TeamNr
     SELF:oDCbbPlayers:Server:CREATEORDER( "server:TeamNr" )

CASE oCol:Column:FieldSym == #League
 SELF:oDCbbPlayers:Server:CREATEORDER( "str(server:League,2)"
)

OTHERWISE
     InfoMsg( SELF, "No Column Sort", ;
          "The column you have clicked-on is not available for
          sorting on." )

ENDCASE

SELF:oDCbbPlayers:Server:Commit()
SELF:oDCbbPlayers:Recalculate()
SELF:oDCbbPlayers:Redraw()
SELF:oDCbbPlayers:Server:GoTo( nRec )
SELF:oDCbbPlayers:ResetNotification()
```

Using a dbServer bBrowser object is very similar to the above. Simply remove the CreateOrder() statements above and replace them with respective SetOrder() statements. Of course you can leave out the :Server: code also as they won't apply now, including the GoTo() as VO will maintain the record pointer when changing index

tags.

How do I change the view/justification of a column caption?

Obviously there will be many a time that you will need to modify the features of a column caption. Typically you are changing the caption, the text color, or the alignment of the text itself. The sample below shows you how to do all of that. Note that the "caption" assignment below contains a crlf in it. This displays "Company" on the top line of a caption, and "Name" on the bottom line of the column caption. Don't forget to increase the height of your caption in the bBrowser object itself. The next line below is the CaptionView. In the sample below we are changing the text color of the caption and also centering the text. Again, any time we change column caption features, we need to include those last 3 lines (for each individual column).

```
oGrdColumn     := SELF:oDCbbPatterns:GetColumn(#CoName)
oGridColumn:Caption           := "Company"+crlf+"Name"
oGridColumn:CaptionView  := ;
     bViewStyle{Color{192,192,192},, BALIGN_CENTER}
SELF:oDCbbPatterns:Recalculate()
SELF:oDCbbPatterns:Refresh()
SELF:oDCbbPatterns:Redraw( #Caption, TRUE )
```

How can I display alternate colors on rows similar to the old green/white computer printouts?

Some people like their browsers to display their rows of data with alternating colors, similar to an old style printer output. This sample uses the Rows of the bBrowser, not the underlying data, as they're fixed to the view:

```
// Even
bCond := bColorCondition( ;
     {|x| x:GetRowNo() % 2 = 0}, obBrowser,, ;
     Brush{Color{COLORGREEN}} )
obBrowser:ColorCondition:Add( bCond )

// Odd
bCond := bColorCondition( ;
     {|x| x:GetRowNo() % 2 > 0}, obBrowser,, ;
     Brush{Color{COLORWHITE}} )
obBrowser:ColorCondition:Add( bCond )
```

How to react on the user pressing a key on the keyboard?

bBrowser also provides the capability to react to keyboard commands. This is accomplished via use of the bKeyCommand class. You simply define key combination to react to, and then define it in your PostInit() (or other) method. Then you also create a method that will be invoked when that key command is pressed. See the sample below for when the user presses the CTRL + DELETE key combination. Note that the keyboard command works only if the bBrowser object has current focus.

```
METHOD PostInit() CLASS myDataWindow
```

```
  LOCAL oKeyCommand AS bKeyCommand

  oKeyCommand := bKeyCommand{KeyDelete,, ;              True,,,
self, #RecordDelete}
  self:oDCBrowser:KeyCommand:Add(oKeyCommand)

METHOD RecordDelete() CLASS myDataWindow
  self:oDCBrowser:Delete()
```

How can I determine what the current column is, and then also get it's column object?

This is actually quite simple. You obtain the :CurrentColumn number, and then use that to get the bDataColumn object for that particular column.

```
LOCAL dwColNr    AS DWORD
LOCAL oCol     AS bDataColum

dwColNr   := obBrowser:CurrentColumn

IF dwColNr > 0
  oCol := obBrowser:GetOpenColumn( dwColNr )
ENDIF
```

How can I move a column programmatically?

To move a column on-the-fly is actually a two-setup process. First you want to close the column to be moved, and secondly you need to open it back up at it's new position. The following code sample moves the "City" column into column position number 3:

```
LOCAL oCol   AS bDataColumn

oCol      := obBrowser:CloseColumn(#City, True)

IF !Empty(oCol)
  obBrowser:OpenColumn(oCol, 3)
ENDIF
```

How should I properly delete selected records?

There is a not-so-right way and a correct-right way to delete a selected record from a bBrowser:server. But the key here, as always, is to suspend all notifications from/to the data server. This is to prevent the any other record movement from disrupting the process. As you will see in the sample below, this is handled more efficiently via the :SelectionFirstRow() and :SelectionNextRow() methods in bBrowser:

```
oServer   := obBrowser:Server
oServer:SuspendNotification()

iRecNr    := obBrowser:SelectionFirstRow()

DO WHILE iRecNr > 0
  oServer:Goto( iRecNr)
  oServer:Delete()       // you may need lock/commit/unlock
logic here, of course
  iRecNr  := obBrowser:SelectionNextRow()
ENDIF
oServer:GoTop()
oServer:ResetNotification()
oServer:Notify(NOTIFYFILECHANGE)
```

How do I create color condition using a function call?

There are sometimes situations where you want the color condition to be dependent upon certain expression evaluations that are not always available directly in a codeblock because they are too complex.
Sometimes the color condition is too complex and cannot be defined in a simple condition. In these cases a method can be called in the condition, that returns a logical value. For this in the 2. argument of the method bColorCondition:Init() the object must be passed on, in that the method is defined. For example such an object can be the window of the bBrowser.

The following code fragment demonstrates the proceeding:

```
oColorCondition := bColorCondition{"Server:CheckAge()",;
                SELF,;
                Color{COLORYELLOW},;
                Brush{Color{COLORRED}}}
self:oDCBrowser:ColorCondition:Add(oColorCondition)

self:oDCBrowser:Refresh()
self:oDCBrowser:Redraw()

METHOD CheckAge() CLASS PlayersBrowser
  LOCAL osPlayers     AS OBJECT
  LOCAL nAge            AS INT

  osPlayers    := self:oDCbbPlayers:Server

  nAge := Year(Today()) - Year( osPlayers:BIRTHDAY )

  RETURN ( nAge >= 10 )
```

How to create a complex color condition with several colors?

Based on the previous example, you can also return an array of colors instead of a logical value. However, this array must be in the following format:

auColorSpec [BCS_FOREGROUND]

[BCS_BACKGROUND]

[BCS_SELECTEDFOREGROUND]

[BCS_SELECTEDBACKGROUND]

[BCS_INACTIVESELECTEDFOREGROUND]

[BCS_INACTIVESELECTEDBACKGROUND]

```
METHOD CheckAge() CLASS myDataWindow
  LOCAL auColorSpec      AS ARRAY
  LOCAL osPlayers     AS OBJECT
  LOCAL nAge             AS INT

  osPlayers     := self:oDCbbPlayers:Server

  nAge := Year(Today()) - Year( osPlayers:BIRTHDAY )

IF nAge >= 10
  auColorSpec := ArrayCreate(BCS_ITEMS)
  auColorSpec[BCS_BACKGROUND] := Brush{Color{COLORRED}}
  RETURN auColorSpec
ELSEIF nAge >= 15
  auColorSpec := ArrayCreate(BCS_ITEMS)
  auColorSpec[BCS_BACKGROUND] := Brush{Color{COLORBLUE}}
  RETURN auColorSpec
ELSE
  RETURN FALSE
ENDIF
```

I'm using ADS for my RDD, but it seems really show with respect to bBrowser. What can I do?

Set your bBrowser object's :RowMode property to #Absolute:

```
SELF:oDCbBrowser:RowMode   := #Absolute
```

How do I assign a FieldSpec object to a column?

Assuming that you have already defined some FieldSpecs, it is as simple as this (of course you need to open the column and recalc/redraw it properly too):

```
oGridColumn    := SELF:oDCbbCptCodes:GetColumn( #SSN )
oGridColumn:FieldSpec    := fsSSN{}
```

How can I display multi-line column captions?

First assign a larger caption height to your bBrowser object itself:

```
SELF:oDCbbCptCodes:CaptionHeight := 30
```

Then define your respsective column captions when you configure them at load time:

```
oGridColumn    := SELF:oDCbbCptCodes:GetColumn( #Location )
oGridColumn:caption := "City" + crlf + "State"    // double
oGridColumn:caption := crlf + "State"   // blank top line
```

This sample shows similar code for a triple header caption:

```
SELF:oDCbbCptCodes:CaptionHeight := 45

oGridColumn    := SELF:oDCbbCptCodes:GetColumn( #SSN )
oGridColumn:caption := "Social" + crlf + "Security" + crlf +
"Number"
```

Is there a way for the use to double-click anywhere on a row and toggle a column's checkbox?

If you want the user to be able to toggle any given checkbox by double-clicking on any other cell in the same row, then you need to create a CellDoubleClick() event method to handle this. Of course, you could probably find other similar uses for this same method.

```
METHOD CellDoubleClick() CLASS SelectCodes

SELF:oDCbbChoices:Server:FIELDPUT( #SelChkBox, ;
     !SELF:oDCbbChoices:Server:FIELDGET( #SelChkBox ) )

SELF:oDCbbChoices:Server:commit()
SELF:oDCbbChoices:Refresh( #RefreshBuffer )
```

Slightly More Complex Tips and Tricks

Now we will move on to some more advanced topics, specifically some of the event methods that you will typically be using on a regular basis.

How can I complete the checkbox click process without having to change focus manually?

Unfortunately, bBrowser requires that you double-click a checkbox for the toggle event of (un-)checking a box to take effect. Or for the checkbox cell to lose focus to another row/column. My preferable method is to allow the user to accomplish this step with a single mouse-click. And to accomplish this, we only need to utilize the standard VO ButtonClick() event method. The process involves doing a GoTo() to the same record/row that we are on, thus simulating losing focus but really staying on the current row. Then we :Commit() the change to disk and refresh the buffer. The sample below shows you how:

```
METHOD ButtonClick(oControlEvent) CLASS SelectCodes
LOCAL oControl AS Control
LOCAL sSym AS SYMBOL
oControl := IIf(oControlEvent == NULL_OBJECT, NULL_OBJECT,
oControlEvent:Control)
SUPER:ButtonClick(oControlEvent)
//Put your changes here
sSym := oControl:NameSym

DO CASE
CASE sSym == #Selected
     SELF:oDCbbChoices:Server:Commit()
     SELF:oDCbbChoices:Server:goTo(SELF:oDCbbChoices:Server:re
cno)
     SELF:oDCbbChoices:Server:Commit()
     SELF:oDCbbChoices:Refresh( #RefreshBuffer )
ENDCASE
```

How can I react to field or record changes for post-validation logic and such?

Okay, I haven't mentioned this method yet, but it is a VERY important one in your bBrowser development process. So important that you MUST include this event method in every dialog window class that you create.

The first parameter of this method tells you which type of notification has occurred, however you will typically only be concerned about the field and record change notifications. The second method parameter is used to tell you specifically which cell has been accessed and changed.

```
METHOD Notify( nChange, cDesc ) CLASS Ins2PayWindow
// Event handler for data entry in a bBrowser object //
control    (the EditFocusChange of bBrowsers)
LOCAL xFlag              AS USUAL
```

```
LOCAL nLineNr       AS INT

xFlag          := NIL

DO CASE
CASE nChange == NOTIFYINTENTTOMOVE
     xFlag          := TRUE

CASE nChange == NOTIFYRECORDCHANGE
     // Enter logic code here to respond to
     // any change that might have occurred
     // for a record/row.

CASE nChange == NOTIFYFIELDCHANGE
     // Enter logic code here to respond to
     // individual cells that have been updated.
     nLineNr   := SELF:oDCbbInsPay:Server:recno

     DO CASE
     CASE cDesc == #PMTAMT
          // Call a function to update some totals:
          SELF:UpdateTotals()

     CASE cDesc == #ACTIVE
          IF  !SELF:oDCbbInsPay:Server:FIELDGET( 8 )
               // Do something here…
          ENDIF

          SELF:oDCbbInsPay:Refresh( TRUE )

     CASE cDesc == #InsNote
          IF nLineNr = 1
               // Enter some logic code here if they
               // were specifically on the first row
          ENDIF

     ENDCASE
ENDCASE

RETURN xFlag
```

Okay, what about pre-validation logic too?

There is another event method called CellEdit() that you will typically have in most of your bBrowser dialog window classes. With this method you can restrict which individual cells the use can access or update. Maybe

you want to restrict access to cells in a given column, yet don't allow them access to those with information already in them (thus only the emty cells). Or maybe you want to invoke a ComboBox object for a given cell, or possibly a hyper-text search as they type info into a cell – similar to an SLE in operation. All those things are handled via this event method. The next section will discuss some samples, but I wanted to at least give you a shell here that you can insert into your dialog window's class. Note that this method could feasibly be called numerous times in the course of a cell edit process. You will use the oCol variable below to determine which particular column the user has selected.

```
METHOD CellEdit(oCellEditEvent) CLASS SomeClassName
LOCAL oCol          AS bDataColumn

oCol      := SELF:oDCbbGroups:GetOpenColumn(
oCellEditEvent:EditCell:Column )

// Normal sequence:  1 2 5 3 7 6
DO CASE
     // check the edit mode
CASE oCellEditEvent:EditMode = BEDIT_CREATE      // 1
     // This is used for pre-validation logic.
CASE oCellEditEvent:EditMode = BEDIT_INIT        // 2
     // This is used for field validation if we need it.
CASE oCellEditEvent:EditMode = BEDIT_END         // 3
     // This is used for post-validation logic.
CASE oCellEditEvent:EditMode = BEDIT_CANCEL      // 4
     // This is used for cancellation logic.
CASE oCellEditEvent:EditMode = BEDIT_SHOW        // 5
     // This is used for show logic.
CASE oCellEditEvent:EditMode = BEDIT_HIDE        // 6
     // This is used for hide logic.
CASE oCellEditEvent:EditMode = BEDIT_COMMIT      // 7
     // commit implemented
     SELF:oDCbbGroups:Server:Commit()
     SELF:oDCbbGroups:Refresh( #RefreshBuffer )

ENDCASE

RETURN TRUE
```

Tips and Tricks for Those Who Like Challenges

This section will show you some samples of some more complex operations possible with bBrowser.

How can I invoke a dropdown combobox when a user selects a specific cell/column?

bBrowser gives you the capability to invoke a pseudo-ComboBox object when a user selected a pre-determined cell. They can then select from this ComboBox and the value that you define will be stuffed into the cell that they were editing. This ComboBox is really just another dialog box with a small bBrowser object in it. You populate that ComboBox on-the-fly with either an array or a subset of some data file. The sample snippet of code below is actually part of a CellEdit() event method. You first determine if you are in the particular column that involves the ComboBox (in this case, the "State" column), then you call the GetOpenColumn() method with the parameters as show below. The only part you will typically need to change is the last parameter (in this case "#bStatesEdit), which is the class that is called to open and display your respective ComboBox class. The sample source code that I provide with this paper/session has a prime example of how to setup a typical ComboBox class.

```
METHOD CellEdit(oCellEditEvent) CLASS PlayerBrowser
LOCAL oColumn       AS bDataColumn

DO CASE
// check the edit mode
CASE oCellEditEvent:EditMode = BEDIT_CREATE
     // Open up a combobox for the "state" cell:
     oColumn :=
oCellEditEvent:Control:GetOpenColumn(oCellEditEvent:EditCell:C
olumn)
     IF oColumn:FieldSym==#State
          oCellEditEvent:EditControl :=
oCellEditEvent:Control:EditCreate(oCellEditEvent:EditCell:Colu
mn, oCellEditEvent:EditCell:Row,
oCellEditEvent:EditCell:RecNo, #bStatesEdit)
     ENDIF
```

How can I prevent a user from selecting a cell once data is entered in it (yet leave other cells in the same column editable)?

```
METHOD CellEdit(oCellEditEvent) CLASS PlayerBrowser
LOCAL oColumn       AS bDataColumn

DO CASE
// check the edit mode
CASE oCellEditEvent:EditMode = BEDIT_CREATE
     // If there is a value in the SSN field,
     // then don't let them edit it:
     IF oCellEditEvent:EditCell:Column = 8 .and. ;
               !Empty( SELF:oPlayersServer:FIELDGET( #SSN ) )
          RETURN FALSE
     ENDIF
```

Conclusion

What else can you do with bBrowser? Well, I haven't done any of these items myself, but I have heard of and/or seen them in action. You can also display a graphic image in a cell. You can drag and drop from another screen to into a bBrowser cell (all really just done with background logic). You can implement hyper-lookups into cells that will auto-fill matching data for the user. So don't hold back, be creative and inventive. There are very few limitations that you have in bBrowser.

What it really comes down to is that you need to study as many samples as you can, to learn from and exploit them. Also study the bBrowser Help file – it is a wealth of information and code snippets. And of course, do not forget to study the samples that come with bBrowser – they are very helpful learning tools. Plus you will see some more of bBrowser's capabilities, such as graphic images and multi-line text and simple bar graphs.

This white paper will eventually become public domain, so if you have some tips and tricks of your own that you would like included, please feel free to send them to me and I will include them in here.

The source code samples included with this paper show several ways to display various color conditions. One of the samples also shows you how to incorporate a ComboBox in a column. There is also an sleBrowser module included with the samples. This was written by Geoff Schaller I believe, and allows you to use a bBrowser object in similar fashion to a standard ComboBox, but from an SLE control instead. It also allows you to display several columns instead of a single one as in standard ComboBox controls in VO. And lastly, Jamal Assaf was kind enough to donate some code to show how he implemented a drap and drop feature from one bBrowser control to another in a separate dialog window.

One other note, if you have the "limited edition" version of bBrowser that comes with VO 2.6, then you really need to take advantage of the full product by purchase the upgrade. It is well worth the minimal fee that is charged. And if you really want to get resourceful, also purchase the source code too. I believe that you can purchase bBrowser from www.votools.com and www.bBrowser.com .

Biography

Tom Walden telecommutes for a business in Pennsylvania and also owns a consulting business on the side where he lives along the Space Coast of Florida. He lives and breathes CA-Visual Objects applications day and night, and his usage of it dates back to the original 1.0 alpha team of testers. Tom was also one of the co-authors for Ginny Caughey's "Using Visual Objects" 1.0 book. He has also spoken on various CA-Visual Objects topics at related computer conferences in Germany, Italy, Holland, and all over the USA.

He can be reached at twalden1@cfl.rr.com .