

The Ultimate Grid

In a project I inherited from another developer, I needed a way for a user to look up a record from another table or data source. A ComboBox was not the right solution because there were 30,000 to 40,000 records to choose from. I wanted to offer the user incremental search as well. What I needed was a lookup grid that could be dropped onto a modal form.

The project included 60 forms that needed lookup capability, so a generic solution was called for.

To meet these needs, the `cntGrid` class was born.

Structure of the Solution

`cntGrid` is a container class consisting of a grid and a checkbox. Each column of the grid uses a textbox from a custom class called `IncSeek` instead of the Visual FoxPro base class. `IncSeek` provides incremental search capability. The check box determines case-sensitivity for incremental search. Clicking a column header in the grid reorders the grid based on that column. A second click on the header reverses the order. A right-click in the grid calls a short cut menu that lets the user zoom into the highlighted record. There's also a form class (Form Pick_List) designed to work with the `cntGrid` as a lookup form. If the grid is on a lookup form, the form is released when the user zooms into a record.

The solution involves the following classes and behavior:

CntGrid is a container class containing a grid and a check box.

Grid_For_Display_Records is a grid class that lets the user reorder grid contents by clicking the column header. It shows the header of the chosen column in a different color.

IncSeek is a TextBox class that performs incremental search. It displays the characters entered so far in the column's header. In a lookup form, double click or Enter selects the highlighted record and releases the form, while Escape releases the form without a selection.

RightClick calls a shortcut menu containing Cut, Copy, Paste, Clear, and Select All. If there is a form to zoom into the highlighted record, the context menu also contains a Zoom It item. If the zoom form is already active, choosing Zoom It refreshes the form to display the chosen record.

The Buttons class

contains OK and Cancel buttons.

CaseSensitive is a CheckBox that determines whether incremental search is case sensitive.

Form_PickList is a form class specially designed to support the `cntGrid` class. It resizes the grid and itself, based on the field Sizes. The form has OK and Cancel buttons to finish the lookup. It can be resized by the user. If a record is selected, the form returns .T. and refreshes the calling form when it's released.

Figure 1 shows `cntGrid` used on a of a regular page frame with a case-sensitivity checkbox:

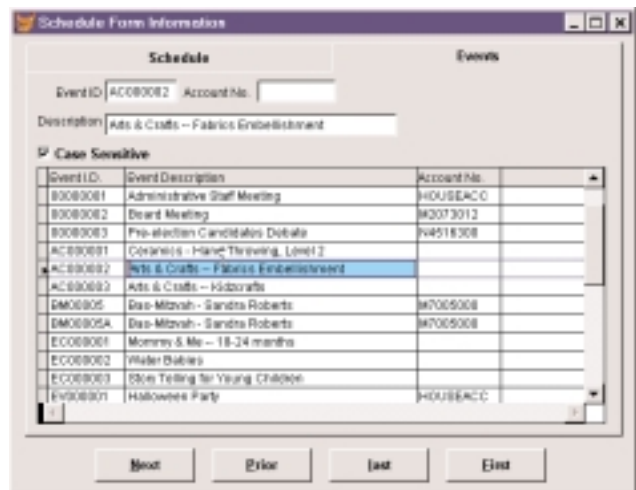


Figure 1: The Ultimate Grid at Work – The grid provides a quick way to find the right record.

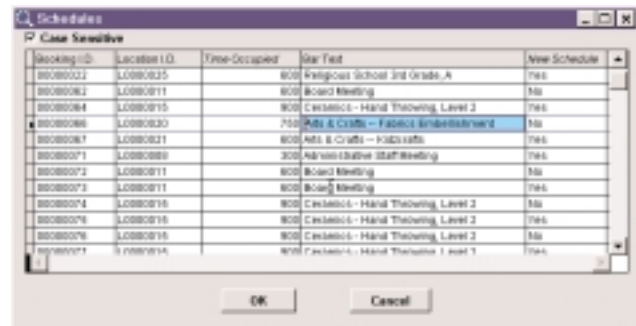


Figure 2: Using the ultimate grid for lookups – a form like this might be called from a field on another form to select a record.

Design Issues

Since it's not possible to subclass a grid without presetting the number of columns, there are two options: add columns on the fly or subclass the grid and add columns

at design time, then use only the columns needed. I chose the second option as my tests showed it to be faster. The cost this way comes at instantiation when the columns, headers, and textboxes are created, only to be later hidden.

My tests show that adding columns and all they need on the fly takes more processing time.

My grid class has 16 columns; if you need more, just set ColumnCount higher in your class. The code doesn't care how many columns there are in your grid.

In each column, remove the default text box and substitute one based on the IncSeek class (described above). Make sure the name of text box in all columns is txtIncSeek. Set ControlSource, RecordSourceType, and RecordSource to None. Insert one line of code in the header's Click event for each column:

```
This.Parent.Parent.ReOrder(This.Parent)
```

This way we pass the reference object of the column into the grid ReOrder method.

Another alternative for design:

I also tested the grid, while adding a column on the fly. There are some minor differences in code. I created a class just for the column's header.

```
DEFINE CLASS GridHeader AS Header
    FontItalic=.T.
    FontSize=9
    Caption=""

    PROCEDURE CLICK()
        This.Parent.Parent.ReOrder(This.Parent)
    ENDPROC
ENDDEFINE
```

Also had to add the following code before the FOR.. END-FOR code.

```
IF ! 'GRDCOL' $ SET('PROCEDURE')
    SET CLASSLIB TO FcgCtrls ADDITIVE
    SET PROCEDURE TO GrdHdr ADDITIVE
ENDIF
```

I had to add the following lines of code within the FOR.. ENDFOR in the container class.

```
.Columns[m.lnJ].AddObject("txtIncSeek", "IncSeek")
.Columns[m.lnJ].CurrentControl="txtIncSeek"
.Columns[m.lnJ].Visible=.T.
.Columns[m.lnJ].RemoveObject("Header1")
.Columns[m.lnJ].AddObject("Header1", "GridHeader")
```

On average it took 170 to 175ms to load a lookup form with 5 columns including 2 calculated fields.

If I used the class with the above code: It took about 175ms compare to 135ms, if all columns are pre define in the class. The resource hit is also higher, if I add an object on the fly.

Even though having the columns not as pre defined, I believe the need for speed and performance is also important.

```
This.grdPickList.ColumnCount=ALEN(taBehavior,1)
```

The above line will create only the columns that are needed.

Also I had to reset the ColumnCount to 0, since the moment the grid knows of the alias it puts columns to accommodate the table. Those columns latter to be removed. (not hidden). Thus I can start with Column1 Column2 etc....

```
DEFINE CLASS GridColumn AS Column

    ADD OBJECT Header1 AS GridHeader
    ADD OBJECT txtIncSeek AS IncSeek

    Visible=.T.
    CurrentControl="txtIncSeek"
ENDDEFINE

DEFINE CLASS GridHeader AS Header
    FontItalic=.T.
    FontSize=9
    Caption=""

    PROCEDURE Click
        This.Parent.Parent.ReOrder(This.Parent)
    ENDPROC
ENDDEFINE
```

There is a difference of 40ms between the three classes. If we are using hundreds of thousands records it's going to be noticeable.

Still I would work with the class that provides better performance and less resource hit.

Setting Up the Grid

Most of the heavy duty work is done by the container class' SetUpGrid method, called when the whole thing is instantiated.

The Container Class has two custom properties:

IsItLookUpForm indicates whether the class is sitting on a Lookup form or a different kind.

IsItCheckBox—indicates whether the case-sensitivity checkbox is visible .

aAppClassArrayName—Gets the array name of the application class or any array that holds the form's name and the object reference for any open forms within the application. The array or the object reference that holds the array must be PUBLIC.

aAppClassArray[1,0]—Gets a copy of the application class array if exist, or any array within the application. The array name is taken from the aAppClassArrayName property.

cMovePointer—Holds the name of the move pointer method within the calling or a regular form.

The SetupGrid method

has four parameters; only the first is required.

The first parameter is a two-dimensional array, **taBehavior** with four columns and one row for each grid column. It holds the headercaption, ControlSource and, optionally, tag for the column. If there is no tag, the third array column can contain the word "LOCATEKEY"; in that case, incremental search uses LOCATE rather than SEEK. If the third column of the array is empty, incremental search is disabled for the corresponding grid column.

The fourth array column can be used to specify the column's width in characters. If it's omitted, the class figures out the column's width based on the field size.

tcTable indicates what table the grid should display. If m.tcTable is omitted, the grid uses the current alias.

The array **taCaseSensitive** indicates, for each column, whether it's case sensitive or not.

tlIsChk determines whether a checkbox for case-sensitivity ever appears.

```
*****
** Program   : GridSetUp for the cntGrid Class
** Author    : Doron Farber
**           : E-MAIL:doron@dfarber.com,
**           : Tel: 516-796-6545
** Created   : Nov 20, 1998
** Copyright : The Farber Consulting Group, Inc
** Purpose   : Size the grid and bound fields,
**           : to its columns.
** Parameters: taBehavior - Array that gets
**           : header Caption, ControlSource,
**           : tag name. Substitue tag name
**           : with "LOCATEKEY", performs search
**           : with LOCATE
**           : m.tcTable - LookUp table
**           : m.taCaseSensitive - An array to
**           : indicate if field is case
**           : sensitive or not
**           : m.tlIsChk - A flag to indicate if
**           : check mark is installed
** MemVars   : m.lnJ - Counter for the FOR ...
**           : ENDFOR loop
**           : m.lnGridWidth - grid width
**           : m.lnFldSize - field size
**           : m.lcAlias - alias name
**           : m.lnFontWidth - average character
**           : width in pixels
**           : m.lnTextWidth - length of a
**           : character expression with respect
**           : to the average character width
**           : for a font.
** Called From : ThisForm.Init()
*****
LPARAM taBehavior,m.tcTable,taCaseSensitive,;
      m.tlIsChk
LOCAL m.lnJ,m.lnGridWidth,m.lnFldSize,m.lcAlias,;
      m.lnFontWidth,m.lnTextWidth

EXTERNAL ARRAY taBehavior
IF EMPTY(taBehavior)
  =MESSAGEBOX("You Must Pass All Grid Info Within;
  the taBehavior Array",16, "Error")
RETURN
ENDIF

WITH This.grdPickList
  *** Assign number of columns
  .ColumnCount=ALEN(taBehavior,1)
  *** Make sure array is populated, if passed as
  *** parameter it is the same
  DECLARE taCaseSensitive[.ColumnCount,1]

  *** Used a lookup table
  IF ! EMPTY(m.tcTable)
    .RecordSource=tcTable
  SELECT (m.tcTable)
ENDIF
```

```
m.lnGridWidth=0
*** Calculate average character width in pixels
m.lnFontWidth=FONTMETRIC(6, .Column1.FontName,;
      .Column1.FontSize)
*** Calculate the length of a character
*** expression with respect to the
*** average character width for a font. (just
*** for one character)
m.lnTextWidth=TXTWIDTH('a', .Column1.FontName,;
      .Column1.FontSize)
*** It could be default alias or lookup table
m.lcAlias=ALIAS()

FOR m.lnJ=1 TO .ColumnCount
  *** Install the column caption
  .Columns[m.lnJ].Header1.Caption = ;
  taBehavior[m.lnJ ,1]

  IF EMPTY(taBehavior[m.lnJ,4])
    *** Install the column field
    .Columns[m.lnJ].ControlSource = ;
    m.lcAlias+"."+taBehavior[m.lnJ,2]

    m.lnFldSize=;
    FSIZE(taBehavior[m.lnJ,2],m.lcAlias)
    IF m.lnFldSize<;
      LEN(.Columns[m.lnJ].Header1.Caption)
      m.lnFldSize=;
      LEN(.Columns[m.lnJ].Header1.Caption)
    ENDIF
  ELSE
    *** Install the column's calculated field
    *** expression
    .Columns[m.lnJ].ControlSource = ;
    taBehavior[m.lnJ,2]
    m.lnFldSize=taBehavior[m.lnJ,4]
  ENDIF

  *** Install the tag if any
  .Columns[m.lnJ].txtIncSeek.cColumnTag = ;
  taBehavior[m.lnJ,3]

  *** Establish the column width
  .Columns[m.lnJ].Width=;
  m.lnFontWidth*m.lnTextWidth*m.lnFldSize+5

  *** Calculate the grid size
  m.lnGridWidth=;
  .Columns[m.lnJ].Width+m.lnGridWidth

  *** Install case sensitive with each column
  *** text box
  .Columns[m.lnJ].txtIncSeek.lCaseSensitive=;
  taCaseSensitive[m.lnJ,1]

  *** If search is enable font is not italic
  IF ! EMPTY(taBehavior[m.lnJ,3])
    .Columns[m.lnJ].Header1.FontItalic=.F.
  ENDIF
ENDFOR
ENDWITH

WITH This
  IF .lIsItLookUpForm
    *** Calculate the grid size
    .grdPickList.Width=m.lnGridWidth+40
    ***Make the container same width as grid
    .Width=.grdPickList.Width

    IF m.tlIsChk
      .lIsThereChk=.T.
      .chkCaseSensitive.GetCase(1)
    ELSE
      .chkCaseSensitive.Visible=.F.
      .grdPickList.Top=0
      .grdPickList.Height=.Height
      .Top=6
    ENDIF
  ELSE && IF .lIsItLookUpForm

    IF m.tlIsChk
      .lIsThereChk=.T.
      .chkCaseSensitive.Left=0
      .grdPickList.Top=22
      .grdPickList.Height=.Height-.grdPickList.Top
      .chkCaseSensitive.GetCase(1)
    ELSE
      .chkCaseSensitive.Visible=.F.
      .grdPickList.Top=0
      .grdPickList.Height=.Height
    ENDIF
    *** Get the grid size, based on the
    *** container size
    .grdPickList.Width=.Width
  ENDIF && IF .lIsItLookUpForm
ENDWITH
```

The Grid_For_Display_Records Class has several custom properties:

cNewHeaderColor holds the background color to use for the header of the current sort column.

nActiveCol tracks the active column.

nPriorHeaderColor holds the previous background color of the header's of the current sort column.

oPriorColumn holds an object reference to the current sort column.

The Reorder method

sets the current search order based on the header clicked. It also changes the header's background color. A reference to the current sort column is stored in *oPriorColumn*. Using this object reference, we reset all sorting properties of the *IncSeek* textbox in that column. In *VFP 5*, this property causes a problem that prevents the form from completing closing and keeps the *Unload* method from firing.

Setting *oPreviousColumn=NULL* in the form's *Destroy* method solves the problem, but breaks encapsulation. In *VFP6*, the problem is gone.

```
*****
*** Program   : Reorder
*** Author    : The Farber Consulting Group
***           : Inc,By Doron Farber
*** Purpose   : Change the tag based on current
***           : index assigned into the cColumnTag
***           : property(within the IncSeek class)
***           : paint the current header with any
***           : other color, and re-paint previous
***           : active index(header) with its
***           : original color
*** Parameters: m.toColumn - Gets the reference
***           : object of column from the header
***           : click method
*** Memvars   : m.lnNowRec - Current record
***           : m.lcCurrTag - Current tag
*** Called From: The header click method
*** Calling   : None
*** Return    : None
*** Notes     : None
*****
LPARAM m.toColumn
LOCAL m.lnNowRec,m.lcCurrTag
m.lnNowRec=RECNO()

WITH This
*** Is it using the LOCATE or SEEK ?
*** If not abort the header click.
m.lcCurrTag=m.toColumn.txtIncSeek.cColumnTag
IF UPPER(m.lcCurrTag)="LOCATEKEY" OR ;
  EMPTY(m.lcCurrTag)
  RETURN
ENDIF

ThisForm.LockScreen=.T.
*** If there is no value in cWhatOrder it
*** means first time click on this column's
*** header. When click on another header, it
*** resets it's value to
*** .txtIncSeek.cWhatOrder=.F.
IF EMPTY(m.toColumn.txtIncSeek.cWhatOrder)

  *** Check if any previous column header was
  *** pressed
  IF ! ISNULL(.oPriorColumn)
    WITH .oPriorColumn
      ***Restore previous header's color
      .Header1.BackColor=This.nPriorHeaderColor

      *** Hold the header BackColor before it was
      *** pressed
      .nPriorHeaderColor=;
      m.toColumn.Header1.BackColor
```

```
*** Reset previous column properties and tag
IF ! EMPTY(.txtIncSeek.cPreviousTagCol)
  IF .txtIncSeek.lIsItDescendingCol
    SET ORDER TO TAG ;
    (.txtIncSeek.cPreviousTagCol);
    DESCENDING
  ELSE
    SET ORDER TO TAG ;
    (.txtIncSeek.cPreviousTagCol);
    ASCENDING
  ENDF
  .txtIncSeek.lIsItDescendingCol=.F.
  .txtIncSeek.cPreviousTagCol=.F.
  .txtIncSeek.cWhatOrder=.F.
ENDIF
ENDWITH
ENDIF

*** Hold the reference object of the current
*** column, so we can reset its properties
*** when another header is pressed.
.oPriorColumn=m.toColumn

*** Hold the header BackColor before it was
*** pressed
.nPriorHeaderColor=m.toColumn.Header1.BackColor

*** Setup the new pressed header
WITH m.toColumn.txtIncSeek

  *** Paint the new pressed header with a
  *** new color
  m.toColumn.Header1.BackColor=;
  EVAL(This.cNewHeaderColor)
  SET ORDER TO (m.lcCurrTag)
  *** Get the ORDER(), so when clicking on
  *** another header, we can reset its ORDER()
  *** to its original state. Descending or
  *** Ascending. Since this form uses Default
  *** DataSession, we do not want change the
  *** DE of the calling form. This would apply
  *** only when calling a lookup form

  .cPreviousTagCol=ORDER()
  IF DESCENDING()
    .lIsItDescendingCol=.T.
    .cWhatOrder="ASCENDING"
  ELSE
    .cWhatOrder="DESCENDING"
    .lIsItDescendingCol=.F.
  ENDF
  IF ! EMPTY(.cColumnCaption)
    .ClearSearch()
  ENDF
ENDWITH

ELSE && EMPTY(m.toColumn.txtIncSeek.cWhatOrder)

  *** Change the order to Descending or
  *** Ascending It must be second header
  *** click or more
  WITH m.toColumn.txtIncSeek
  IF .cWhatOrder=="DESCENDING"
    .cWhatOrder="ASCENDING"
    SET ORDER TO TAG (m.lcCurrTag) DESCENDING
  ELSE
    .cWhatOrder="DESCENDING"
    SET ORDER TO TAG (m.lcCurrTag) ASCENDING
  ENDF
  ENDF
  ENDF && EMPTY(m.toColumn.txtIncSeek.cWhatOrder)

  LOCATE
  .Refresh
  ThisForm.LockScreen=.F.
ENDWITH
```

The IncSeek Class uses these custom properties

cColumnCaption Holds the column caption.

cColumnTag Holds the tag for the column.

cNowAlias Holds current alias.

cNowTag Holds current tag.

cSearchBuffer Holds the key press as a buffer.

lCaseSensitive indicates whether the search is case sensitive.

lIncSeekMode indicates whether the search is successful.

lIsSearchActive turns search mode on and off.

nFirstRec	Gets the first record before search is performed.
nOldRec	Holds the record before the search is done.
cPreviousTagCol	Holds the previous column's tag.
cWhatOrder	It is a flag to tell us first time click on this column's header or not.
lIsItDescendingCol	Let us know if tag is descending or ascending.

The KeyPress method

handles keystrokes when the textbox is read-only (and therefore, set for incremental search). The following keystrokes are trapped:

BACKSPACE removes the last character from the search string and searches for that new string.

DEL does nothing.

HOME moves the pointer to the first record.

END moves the pointer to the last record.

ENTER selects the current record and

ESCAPE sends a release message to the form without selecting a record. It works only if the object is on a lookup form.

ENTER and **ESCAPE**, set the container's lOkSelected property, which is referenced by the lookup Form. (The actual code for releasing the form is in the Destroy method of the form_pick_list class. When it fires, it checks the lOkSelected property.)

```
*****
** Program      : KeyPress
** Purpose      : Trap the key press
** Parameters   : nKeyCode - Gets ASCII value
** Called From  : Fired when key is pressed
** Calling      : This.SearchString(),
**              : This.ClearSearch()
*****
LPARAMETERS nKeyCode, nShiftAltCtrl

WITH This
  *** Is it under search mode ?
  IF .lIsSearchActive
    .nOldRec=RECNO()
    DO CASE
      CASE BETWEEN (m.nKeyCode, 32, 126)
        NODEFAULT
        IF LEN(.cColumnTag)>0
          .cSearchBuffer=;
          .cSearchBuffer+CHR(m.nKeyCode)
          .SearchString()
        ENDIF

        *** BACKSPACE key is pressed
        CASE m.nKeyCode==127
          NODEFAULT
          IF LEN(.cColumnTag)>0
            IF LEN(.cSearchBuffer)==0
              .cSearchBuffer=""
            ELSE
              .cSearchBuffer=LEFT(.cSearchBuffer,;
                LEN(.cSearchBuffer)-1)
            .SearchString()
          ENDIF
        ENDIF

        *** Do nothing when DEL key is pressed
        CASE m.nKeyCode==7
          NODEFAULT

        *** When user pressed the HOME key
        CASE m.nKeyCode==1
          ThisForm.LockScreen=.T.
          GO TOP
          IF .nFirstRec>0
```

```
          .ClearSearch()
        ENDIF
        .Parent.Parent.Refresh()
        ThisForm.LockScreen=.F.
      NODEFAULT

      *** When user pressed the END key
      CASE m.nKeyCode==6
        ThisForm.LockScreen=.T.
        GO BOTTOM
        .Parent.Parent.Refresh()
        IF .nFirstRec>0
          .ClearSearch()
        ENDIF
        ThisForm.LockScreen=.F.
      NODEFAULT

      *** When user pressed the ENTER key
      CASE m.nKeyCode==13
        IF .Parent.Parent.lIsItLookupForm
          ThisForm.lOkSelected=.T.
          ThisForm.Release()
        ENDIF

      *** When user pressed the ESCAPE key
      CASE m.nKeyCode==27
        IF .Parent.Parent.lIsItLookupForm
          ThisForm.lOkSelected=.F.
          ThisForm.Release()
        ENDIF
      OTHERWISE
        IF .nFirstRec>0
          .ClearSearch()
        ENDIF
      ENDCASE
    ENDIF
  ENDWITH
  && IF .lIsSearchActive
```

The SearchString Method is the heart of the incremental search, here the actual search is being performed. This method responsible to display searched characters on the column's header, and determines if SEEK or LOCATE is used, or if it is case sensitive or not.

```
*****
***.Program.....: SearchString
***.Author.....: The Farber Consulting Group Inc, By Doron Farber
***.Created.....: Nov 20, 1988
***.Purpose.....: Perform an Incremental Search
***.Parameters..: None
***.Memvars.....: m.lcSearchVal - Gets the search value
***.....: m.lnCurrRec - Gets current record
***.....: m.lcDisplayHeader - Shows actual characters pressed
***.Called From.: This.KeyPress()
***.Calling.....: None
***.Return.....: None
***.Notes.....: None
*****
LOCAL m.lcSearchVal,m.lnCurrRec,m.lcDisplayHeader

WITH This
  IF EMPTY(.cSearchBuffer)
    IF .nFirstRec>0
      GO .nFirstRec
    ENDIF
    .ClearSearch()
    RETURN
  ENDIF

  *** First time key press
  IF .nFirstRec==0
    .nFirstRec=RECNO()
    .cNowAlias=ALIAS()
    .cNowTag =ORDER()
    *** Gets the caption of the column header and hold it
    .cColumnCaption=This.Parent.Header1.Caption
  ENDIF
  m.lnCurrRec =RECNO()

  *** It may use the current alias of the calling form.
  *** If it is empty current alias assumed
  IF ! EMPTY(.Parent.Parent.RecordSource)
    SELECT (.Parent.Parent.RecordSource)
  ENDIF

  *** Get actual pressed characters such as: AbCdee
  *** Since it could be converted to UPPER() for Not
  *** Case Sensitive
  m.lcDisplayHeader=.cSearchBuffer
```

```

*** Check if it is a case sensitive
IF .lCaseSensitive
m.lcSearchVal=.cSearchBuffer
ELSE
m.lcSearchVal=UPPER(.cSearchBuffer)
ENDIF

IF UPPER(.cColumnTag)<>"LOCATEKEY"
SET ORDER TO .cColumnTag
SEEK m.lcSearchVal
ELSE
SET ORDER TO 0
LOCAL m.lcType
m.lcType=VARTYPE(EVAL(This.ControlSource))
DO CASE
CASE m.lcType=="C"
IF .lCaseSensitive
LOCATE FOR EVAL(This.ControlSource)=m.lcSearchVal
ELSE
LOCATE FOR UPPER(EVAL(This.ControlSource))=m.lcSearchVal
ENDIF
CASE m.lcType=="N"
LOCATE FOR LTRIM(STR(EVAL(This.ControlSource)))=m.lcSearchVal
ENDCASE
ENDIF

*** Display pressed key into the header caption
.Parent.Header1.Caption=m.lcDisplayHeader

IF EOF()
IF BETWEEN(m.lnCurrRec,1,RECCOUNT())
GO m.lnCurrRec
ENDIF
ELSE
IF This.nOldRec<>RECNO()
.lIncSeekMode=.T.
ENDIF
ENDIF

IF ! EMPTY(.cNowTag)
SET ORDER TO .cNowTag
ENDIF

IF ! EMPTY(.cNowAlias)
SELECT (.cNowAlias)
ENDIF

ENDWITH

```

The Zoom2Form Method is called from the short cut menu. The actual code to perform the zoom service is found within the cntGrid object. See the ChkAnyArray and ZoomInToForm methods in the full code.

```

*****
** Program      : Zoom2Form
** Author       : The Farber Consulting Group Inc,
**              : By Doron Farber
** Created      : Nov 20, 1988
** Purpose      : Call either a form or a lookup
**              : form if exists.
** Parameters   : m.tcNowAlias-Gets the alias name
**              : taForms-Gets the frmName and SCX
**              : name as an array
** Called from: This.GetMenu()-From the bar
**              : selection, of the ShortCutMenu.
**              : See also This.ShortCutMenu()
** Calling      : None
** Return       : None
** Notes        : None
*****
LPARAM m.tcNowAlias,taForms
EXTERNAL ARRAY taForms
This.Parent.Parent.Parent.ZoomInToForm;
(m.tcNowAlias,@taForms)

```

Using the grid

Here's an example showing a call to a look-up form based on the form_PickList class. We assume the same table as the calling form, and include the case-sensitivity check box. You can see the results of this call in figure 2 above.

```

LOCAL ARRAY aBehavior[1,1],aCaseSensitive[1,1]
DECLARE aBehavior[5,4]
aBehavior[1,1]="Booking I.D."
aBehavior[1,2]="cSch ScheduleID"
aBehavior[1,3]="ScheduleID"

aBehavior[2,1]="Location I.D."

```

```

aBehavior[2,2]="cLoc LocationID"
aBehavior[2,3]="LocationID"

aBehavior[3,1]="Time Occupied"
aBehavior[3,2]='(EsfMMSch.nSch_EndTime-;
EsfMMSch.nSch_StartTime)*5'
aBehavior[3,3]=" "
aBehavior[3,4]=15

aBehavior[4,1]="Bar Text"
aBehavior[4,2]="EsfMMSch.cSch_BarText"
aBehavior[4,3]="LOCATEKEY"
aBehavior[4,4]=35

aBehavior[5,1]="New Schedule"
aBehavior[5,2]='IIF(EsfMMSch.lSch_NewSch,;
"Yes","No")'
aBehavior[5,3]=" "
aBehavior[5,4]=12

DECLARE aCaseSensitive[3,1]
aCaseSensitive[1,1]=.F.
aCaseSensitive[2,1]=.F.
aCaseSensitive[3,1]=.T.

DO FORM PLISTa WITH aBehavior,"Schedules",.F.,;
aCaseSensitive,.F.

IF ! m.llIsCancel
RETURN .F.
ENDIF

ThisForm.txtTimeOccupied.Value=;
EsfMMSch.nSch_EndTime-EsfMMSch.nSch_StartTime

```

In the example shown in figure 1, the container is on a page frame. In this case there is no need to pass the form's caption.

```

LOCAL ARRAY aBehavior[1,1],aCaseSensitive[1,1]

DECLARE aBehavior[3,4]
aBehavior[1,1]="Location I.D."
aBehavior[1,2]="cLoc LocationID"
aBehavior[1,3]="LocationID"

aBehavior[2,1]="Bar Style"
aBehavior[2,2]="nsch barstyle"
aBehavior[2,3]="BarStyle"

aBehavior[3,1]="Bar Text"
aBehavior[3,2]="cSch BarText"
aBehavior[3,3]="LOCATEKEY"

DECLARE aCaseSensitive[3,1]
aCaseSensitive[1,1]=.F.
aCaseSensitive[2,1]=.F.
aCaseSensitive[3,1]=.T.

ThisForm.pgfLocation.Pagel.cntGrid.GridSetUp;
(@aBehavior,.F.,aCaseSensitive,.T.)

```

The full source code, including some additional methods for the IncSeek class, as well as the Form_PickList class is on this month's Professional Resource CD There's also an example of an application on the PRD. As a sample try the booking form as a calling form, and the location form as page frame one.



Doron Farber is the president of The Farber Consulting Group, Inc, based in Long Island NY. He's been programming in Xbase dialects since 1983. His firm develops applications in Visual Basic and Visual FoxPro cold fusion, also provides Web Site development solutions.

Doron developed a commercial field based application for the collection of building deficiencies, while the software provides the recommendations. 516-796-6545, doron@dfarber.com